Maintaining Unstructured Case Bases

Kirsti Racine[1] and Qiang Yang[1]

School of Computing Science
Simon Fraser University
Burnaby BC V5A 1S6,
Canada
Email : kracine@cs.sfu.ca, qyang@cs.sfu.ca
Web : http://fas.sfu.ca/ǩracine

Abstract. With the dramatic proliferation of case based reasoning systems in commercial applications, many case bases are now becoming legacy systems. They represent a significant portion of an organization's assets, but they are large and difficult to maintain. One of the contributing factors is that these case bases are often large and yet unstructured; they are represented in natural language text. Adding to the complexity is the fact that the case bases are often authored and updated by different people from a variety of knowledge sources, making it highly likely for a case base to contain redundant and inconsistent knowledge.
In this paper, we present methods and a system for maintaining large and unstructured case bases. We focus on two difficult problems in case-base maintenance: redundancy and inconsistency detection. These two problems are particularly pervasive when one deals with an unstructured case base. We will discuss both algorithms and a system for solving these problems. As the ability to contain the knowledge acquisition problem is of paramount importance, our methods allow one to express relevant domain expertise for detecting both redundancy and inconsistency naturally and effortlessly. Empirical evaluations of the system prove the effectiveness of the methods in several large domains.

1 Introduction

A pervasive, yet relatively ignored, problem inherent in using case-based reasoning is that of case-base maintenance. A case base is usually constructed over a long period of time, during which cases that solve approximately the same range of problems are entered, by different case authors at different times. As well, a case base may be the result of the union of several different smaller case bases, or the result of "scanning in" raw material from large quantities of literature. Similarly, a company's use for any given case base may change with time. For example, the cases for fixing a certain type of printer in an organization will become outdated when the company acquires a fleet of new printers for replacement. As the case base grows, errors within the case base become increasingly difficult to detect. The result can be contradictions or inconsistencies within a case base. These problems can potentially harm the performance of a case based reasoning

system. All these reasons contribute to the need to update and reorganize a case base during its lifetime.

A case-base maintainer must be responsible for several different tasks. First, as time passes, cases may become redundant simply because there are more powerful cases in the same case base. In addition, some cases may contain inconsistent information either with other parts of the same case or with the background knowledge. A need then arises for identifying these cases and deciding whether to eliminate them. Second, a large case base implies that the cases are not used uniformly. Some cases are used more often than others, and this usage distribution can be dependent on many different factors, including time, the company's asset distribution and business strategies. A dynamic case base requires constant reorganization, so its most frequently, most recently accessed cases are easily presentable to the user. This requirement suggests a hierarchical organization structure for the case base. A complex aspect is that this structure must respond to the continuous change in the user environment. A final aspect of case base maintenance is in the ability for a system to identify and suggest solutions to "inconsistent cases." A case consists of a description of a target problem and a solution. If the case description and solution contain errors, it may lead to contradiction in the solution of a case. This problem will render a case solution unusable by the user. Thus, a case-base maintenance system should have the ability to identify inconsistent cases and parts of a case that are inconsistent with each other.

The problem of case-base maintenance is akin to that of software maintenance. It is now well known that as a software system is constructed, a major portion of an organization's resources is devoted to "software maintenance" in its entire life cycle; estimates put this effort at 50 to 70 percent of the total cost for developing and using the software in its life cycle [MO83, LS81, LST78]. We conjecture that the same amount of effort will be experienced by organizations exploiting case base reasoning systems.

2 Previous Approaches

The previous research in maintaining case-base systems has addressed many different aspects of the cleaning problem. [Aha91], by David Aha, presents several case-based learning (CBL) algorithms which are tolerant of noise and irrelevant features. These algorithms can predict feature values in future cases and, thereby, detect anomalies or possible errors in the data. However, CBL algorithms make several assumptions about the structure of the data, including the requirement that an explicit structure consisting of feature-value pairs be given. Also, features must be uniformly important across all cases for these approaches to work. Moreover, these methods concentrate on local, single-case level solutions. Further research conducted in this area concentrates on detecting discontinuities in case bases [ST96]. However, this research is still focused on very well structured cases, the cases are actually stored in a relational database. Predicting that a case is discontinuous involves examining the relationships between attributes

across all cases.

Another area of case-base maintenance is concerned with *optimization*. Smyth and Keane [SK95] suggested a competence-preserving deletion approach. The premise of this approach is that each case in the base should be classified according to its competence. These classifications are made according to two key concepts: *coverage* and *reachability*. Coverage refers to the set of problems that each case can solve. Reachability is the set of cases that can provide solutions for each current problem. Cases that represent unique ways to answer a specific query are *pivotal* cases. *Auxiliary* cases are those which are completely subsumed by other cases in the base. In between these two extremes are the *spanning* cases which link together areas covered by other cases and *support* cases which exist in groups that support an idea. The deletion algorithm then deletes cases in the order of their classifications : auxiliary, support, spanning and then pivotal cases.

An unresolved issue is how these auxiliary cases are identified, and *what* will be done once they are found. In addition, in our experience, we found that simply deleting cases from a legacy case base is a very dangerous endeavor; since cases represent significant assets of an organization, deleting them could represent a possible loss to the company. In addition, Smyth and Keane's theory does not address the issue of detecting erroneous cases.

3 The Case for Case Base Maintenance

In this section we further clarify the case-base maintenance problem using an industrially relevant domain — the computer printer trouble-shooting domain. We show through the use of this domain that case maintenance is a serious problem not only in theory, but also in practice.

3.1 Two Types of Cases

The majority of the work in case based reasoning has concentrated on cases with well defined features. These cases have a relational structure, where each feature is more or less a field in a relational database. In reality, however, formulating a case into a structured format requires extensive knowledge engineering. For a given domain, the user has to first determine the important features to use to represent each case. Then a decision has to be made on the type of values for each feature. The process of authoring knowledge in this feature-value format requires extensive maintenance when a new feature is discovered and inserted, or when an existing feature becomes irrelevant.

In industrial practice, a majority of the case bases come directly from either unstructured text documents, which are scanned in, or end-users' verbal description. These cases may have generic features such as *problem description* and *problem solution*, but each of these features probably will not be further partitioned down to a relational level. As an example, in a computer-printer repair domain, a case might be described as:

```
Problem: Paper continues jamming laser printer due to dirty
and/or sticky internals.
SOLUTION : The internal components of the laser printer are
dirty and perhaps gummed up. There is also a possibility the
paper
is sticking together. Running regular gummed labels through a
laser printer is a key source of the problem because the high
heat
melts the gum labels.
```

Structured, relational cases often lend themselves to maintenance. Each attribute is associated with a set of values. The cases can be scanned and values that appear infrequently for a particular attribute can be modified or brought to the user's attention. Alternatively, integrity constraints can be specified ensuring that each value entered is a legal one for that attribute. Unstructured cases, on the other hand, are more problematic. Often the cases can not be reduced to a set of variable value pairs so even range checking can be a complex problem. A case-base management agent must be able to account for unstructured cases as well as structured cases.

3.2 The Inconsistent-Case Problem

As a case base grows larger, the number of inconsistent cases will inevitably increase as well. A case can be inconsistent in two different ways:

1. A case can be inconsistent with the background knowledge in an application domain. For example, due to a mis-spelling, a case-base maintainer in a medical domain might have entered "the patient is 200 years old". This is inconsistent with the knowledge that all humans are no older than 115 (if the Guinness Book of World Records is to be believed!).
2. A case can be inconsistent because sections of the it contradict each other. For example, a case from printer-repair domain may have an inconsistent solution requiring the user to both repair and replace the printer.

The medical case-base example above presents an instance of a soft constraint violation. A *soft* constraint violation could occur when a uncommonly occurring feature value is found in a case. In this situation, a warning is desired to bring this item to the users' attention. The printer example, however, demonstrates a hard constraint violation. *Hard* constraint violations are logical contradictions. A self cleaning agent must be able to identify both types of constraint violations.

3.3 The Redundant-Case Problem

With a large legacy case base a need arises to detect if two cases are equal or if one case subsumes another by some criteria determined by the background knowledge. A special case is when two cases are considered equivalent; that is, all attribute values are identical.

An example of redundancy in the printer-repair domain is displayed in Table 1. It demonstrates the difficulty of identifying redundant cases when the cases are unstructured. A string comparison of the two cases presented will detect some similarities, but there are significant differences between the cases.

Case 1
CASE NAME: Envelopes jam laser printer due to glue. SOLUTION: Normal envelopes and laser printers do not get along together. Problems include poor glue heat tolerance.
Case 2
CASE NAME: Paper continues jamming printer due to sticky internals SOLUTION: Envelopes do not work very well with laser printers. The high heat melts the gummed labels.

Table 1. Example of redundant cases in the printer repair domain

4 Maintenance Algorithm

4.1 Overview

Our approach to solving the maintenance problem for unstructured case bases is to integrate an agent within a case based reasoning system. In order to minimize the knowledge acquisition bottleneck, the agent allows unstructured cases to be processed as well as the structured ones. We first use an information-retrieval based algorithm to parse the cases by mining key words and important keywords and key phrases from the unstructured text. These keywords and phrases will offer the basis for subsequent modules to operate on. After the information-retrieval step, we then use a specialized redundancy-detection and inconsistency-detection module to manage the case base.

4.2 Keyword and Phrase Retrieval from Unstructured Cases

We use information retrieval techniques to partially automate the normalization process. The specific steps in this process are:

1. Remove the stop words. Stop words are those words proven to be poor indexers, such as "the" and "of". They typically comprise between 40% - 60% of the words within a document[SM83].
2. Collapse words using a domain thesaurus. In this application, the thesaurus is used to standardize terms. For example, "sega unit" and "sega player" may both appear in a case-based reasoner designed to diagnose cable failure.

These can both be reduced to "sega player" in order to facilitate string matching.

3. Identify significant terms through statistical measures. Keywords are those words which appear frequently within a small set of cases and infrequently across all other cases[FBY92] [SM83]. The key word, the weight of the key word within the file and the documents in which the key word appears are retained.
4. Identify key phrases. Phrases are groups of more than one word which have high inter-case cohesion[SM83]; if one word appears in a case, then the other words have a very high probability of also appearing. Identified phrases must appear in $>$ T cases, where T is a standard threshold or user specified.
5. Generate an inverted index for the entire case base and for the key words. Our inverted index is a list of the terms that appear in the case base, the document number in which the term appears and the weight of the term within the document. This last measure is the frequency of the term within the case.

An example of the information retrieval process applied to one case in the printer-repair domain is shown in Table 2.

Step 1: Read in Original Case
CASE NAME: The printed page is black. CASE SOLUTION: The printed page is black due to an unseated toner cartridge Reseat the toner cartridge and reprint the document. To reseat the toner cartridge: 1.) Turn the laser printer off. 2.) Open the top by pressing button to release latch. NOTE: Some printers require removing the paper tray first.
Step 2: Case After Stop Words Removed
CASE NAME: printed page black CASE SOLUTION: printed page black unseated toner cartridge reseat toner cartridge reprint document reseat toner cartridge turn laser printer press button release latch printers require removing paper tray first
Step 3: Key Words And Phrases
KEY WORDS: toner, cartridge, tray, press, button, release, latch PHRASES: toner cartridge, page black, paper tray, reseat toner cartridge.

Table 2. Example of Information Retrieval Techniques Applied to Incoming Case

Information retrieval techniques facilitate the comparison of cases. Cases are "normalized" allowing the similarities or differences between cases to become

more pronounced. The normalized representation of the case can be used by retrieval schemes also to better solve the user's problem.

4.3 Guidelines for Inconsistent Cases

Addressing the Knowledge Acquisition Problem We chose to use string based rules to represent guidelines. These rules are very close to natural language, and the matching with the underlying case base is done through a string-based fuzzy matching algorithm. This method offers medium speed, but string-based rules are easy for the user to understand and easy for the expert to supply. An additional advantage is that string based rules can be easily modified by the user in case of spelling errors, irrelevant information or difficult wording. For this reason, we call these rules "guidelines".

Guideline Representation String-based guidelines are simply impossible combinations of key words or phrases. Therefore if K represents the keyword set, rules can be expressed as $k_1 \wedge k_2 \wedge ... \wedge k_n$ where $\{k_i \in K | i = 0...n \wedge |K| = n\}$. An example guideline in the printer-repair domain is: {`Guideline: laser printer black ribbon`}

Laser printers do not use ribbons, so this combination of words should not appear in a case. Range inconsistencies can also be defined. These rules are referred to as adjacency rules: {`Guideline : channel < 89`}

Violations of string-based guidelines are detected by examining the inverted index of the incoming case. If all of the words within a guideline are detected within one case, that case is flagged as possibly violating the guideline. The case must then be examined to determine adjacency of the words within the guideline. In the second example, the word "channel" is located and then the word directly following is tested to determine if it is a number and it is greater than "89". If so, a 100% chance of contradiction is reported to the user. Using hashing functions greatly reduces the amount of time required to "test" a case for consistency.

4.4 Solving the Redundancy Problem

Once each case has been given a standard description or profile, redundancy and subsumption can be partially identified.

Equivalence and Pure Subsumption First consider the case where two cases have the exact same string representation - clearly they are equivalent for all intents and purposes. However, that is not the interesting situation. We also detect situations where both the descriptions and the solutions of two cases are close to each other by our nearest-neighbor similarity function. These cases are then presented to the user for further examination.

Cases can also be redundant because they are subsumed by others. The advantage of identifying pure subsumption is that the user can be presented

with two redundant cases and the system can explain that Case 1 subsumes Case 2. In this way, the system can provide the user with information regarding which is the more powerful case. Consider *Subsumption Rule 1*:

Case 1 : Problems (p_1, p_2) Solution (s_1)
Case 2 : Problems (p_1, p_2, q_1) Solution (s_1)

q_1 can either be a keyword or a set of words containing a keyword. In this case, Case 1 subsumes Case 2. The sufficient conditions for solution s_1 have been established to be (p_1, p_2). The value of q_1 is irrelevant. Once the first two premises hold, the solution can be offered to the user. Once this scenario has been detected, the system allows the user to view both cases and highlights the unnecessary condition. As it is possible that Case 1 is an inconsistent case, the fact that it subsumes Case 2 does not mean that Case 2 should be summarily deleted from the case base. The user must examine both cases and decide on the suitable course of action.

Similarly, consider *Subsumption Rule 2*:

Case 1 : Problem: (p_1, p_2) Solution: (s_1)
Case 2 : Problem: (p_1, p_2) Solution: (s_1, s_2)

Similarly to the first example, s_2 can either be a key word or a set of words containing a key word. In this case, Case 1 again subsumes Case 2. If (s_1) is sufficient to solve Case 1, then it is sufficient to solve Case 2. Any additional information or suggested actions are extraneous.

There is one more possibility (*Subsumption Rule 3*):

Case 1 : Problem: (p_1) Solution: (s_1, s_2)
Case 2 : Problem: (p_1, p_2) Solution: (s_1)

In this instance, the system generates a third case:

System Generated Case : Problem (p_1) Solution (s_1)

This new case subsumes both Case 1 and Case 2 by the previous two subsumption rules.

Example We now present an example of how to apply the subsumption rules. Consider the two cases in Table 3. Both cases have the same solution, but Case 2 contains extraneous problem descriptions. By Rule2, Case 2 is subsumed by Case 1. With the end-user's authorization, Case 2 can then be eliminated from the case base.

5 Empirical Testing

We have implemented the agent architecture in the framework of the CaseAdvisor system[1] developed at Simon Fraser University.

[1] To get an evaluation copy, contact http://www.cs.sfu.ca/cbr.

Case 1
Problem : Envelopes jam laser printer due to glue. Solution: Normal envelopes and laser printers do not get along well together. Problems include poor glue heat tolerance.
Case 2
Problem : Paper continues jamming printer due to glue in the internals. The gummed labels of the envelope have melted. Solution: Envelopes do not work very well with laser printers. Problems include poor glue heat tolerance.

Table 3. Detecting Redundancy in the Printer Repair Domain. In this example, Case 2 is subsumed by Case 1

CaseAdvisor™ is a case-based reasoning system implemented in C++ and operates on both the PC and the Internet environments as either a stand alone system or a client/server system. It's advanced functionalities includes visual case authoring, interactive problem resolution, interactive planning using decision forests, and case adaptation for sales automation. For case bases, it supports both flat file structures and ODBC database structures. CaseAdvisor comes in both an application version for the Windows and UNIX environments, and an API (application programming interface) version. So far, CaseAdvisor has been successfully applied to many different help-desk applications in industrial settings (for example, call center applications and financial package suggestions).

Our tests are aimed at establishing the validity of the agent-based approach to case-base maintenance. We hope to confirm through the experiments the following conjectures:

- The information-retrieval approach for processing unstructured cases is feasible for large case bases.
- The redundancy-detection module is capable of detecting most redundant cases.
- The inconsistency-detection module is capable of detecting intra-case inconsistencies through the use of string-based guidelines.
- Finally, the knowledge-acquisition bottleneck problem is adequately addressed by the agent.

5.1 Testing the Information Retrieval Module

Even the one time cost of normalizing a case base is not that expensive. The time required to remove the stop words from all of the cases applying the user defined thesaurus, the time to extract keywords and phrases from the cases and the time to build the inverted file structure were all measured. The information retrieval module was applied to 5 different case bases containing different types of data.

Each case was on average 0.3 kilobytes in size. The Sheffield LISA collection is a database of abstracts and titles extracted from The Library and Information Science Abstracts database from Sheffield University. We performed empirical testing on different components of the Sheffield LISA collection varying the case size from 500 cases to 8000. In all instances, the information retrieval module finished processing in less than 120 seconds. This proves that the self cleaning module can handle large case bases in a reasonable amount of time. When applied to an actual case base designed to diagnose cable failures, the information retrieval module completed processing in less than one second. Testing was completed on large test files to illustrate how the information retrieval module scales.

5.2 Testing the Redundancy-detection Module

The redundancy module is responsible for testing an incoming case for possible redundancy. If there is no possible redundancy, the case is simply added to the existing case base. If there is, the case is presented to the user along with the case causing the possible conflict. The user then determines which, if any, of the cases should be deleted from the case base.

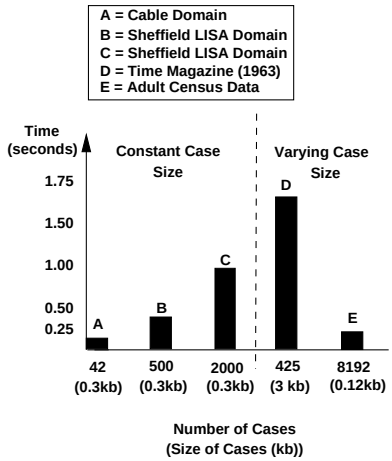

Fig. 1. CPU Time to Detect Redundancy

Figure 1 demonstrates that the algorithm to detect redundancy is efficient enough to be applied in a case authoring module. The average size of the case is also presented in the figure to illustrate the relative performance of the redundancy module is dependent on both the number of cases AND the typical case size. The case base with the largest number of cases, 8192, only needs approximately 0.25 seconds to check for redundancy due to the relatively small size of the cases. The results presented show that the redundancy module scales up to

large case bases quite efficiently. Again, the case base containing, on average, three (3) kilobyte cases took the longest period of time to test for redundancy. However, the system still performed the redundancy check in less than two seconds.

The next experiment involved using blind subjects to type in cases from the same data sources. Five (5) subjects were required to input cases and submit them to be added to the case base. Approximately 50% of the required cases to enter were, in fact, redundant. The data sources used were in the form of decision trees, rather than cases, to introduce a level of indirection. One branch of the decision tree is equivalent to a case. Figure 2 presents the results of this experiment.

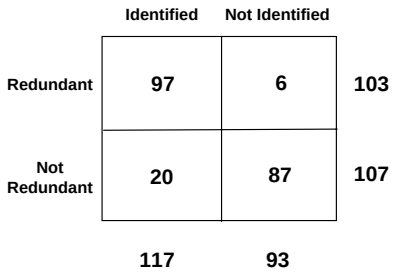

Fig. 2. Quality of Redundancy Module

Figure 2 demonstrates the efficacy of our redundancy module. 94% of the redundant cases were correctly identified by the application. Another encouraging statistic is that 83% of all cases identified as redundant were in fact redundant. Out of the 210 cases entered, 97 were correctly identified as redundant, 20 were falsely identified as redundant, 6 were falsely identified as not redundant and the remaining 87 cases were correctly classified. Using fuzzy string matching to determine redundancy allows for false positives. The threshold for identifying redundancy can be modified. However, this modification must be made at the expense of increasing the number of redundant cases that are not identified by the module.

5.3 System Design

The redundancy and inconsistency detection algorithms are now integrated as part of a larger case-base management module in CaseAdvisor system. Given a collection of text files containing case information, this module semi-automatically extracts the case base and performs redundancy and inconsistency testing and management. The module is also able to merge two case bases and accept a new case while detecting redundant and inconsistent cases.

6 Conclusions and Future Work

We maintain that case-base management should be taken seriously by every practitioner and researcher. Of high importance is the issue of how to contain knowledge acquisition costs while maintaining the case base. Our solution for this problem is a case base maintenance agent which can retrieve important information from a case base and then use this information to detect redundant and inconsistent cases. Our experiments confirmed that the approach can be used to address practical problems of large sizes.

One area of future work is conducting more experiments on the inconsistency detection algorithm. A first task of the experiments is to obtain more realistic guideline for inconsistency detection. These guidelines will be provided by the actual users of the system. With these guidelines we will be able to perform efficiency and usability analysis on the algorithm.

Acknowledgment

We wish to thank Christina Carrick, D. Edward Kim, Philip W.L. Fong and the other members of the Case based Reasoning Group at SFU for their comments. The authors are supported by grants from: Natural Sciences and Engineering Research Council of Canada (NSERC), BC Advanced Systems Institute and Canadian Cable Labs Fund.

References

[Aha91] D. Aha. Case-based learning algorithms. *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*, 1, 1991.

[FBY92] William B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice-HALL, North Virginia, 1992.

[LS81] B. P. Lientz and B. E. Swanson. Problems in application software maintenance. *Communications of ACM*, 24(11):763–769, 1981.

[LST78] B. P. Lientz, E. B. Swanson, and G. E. Tompkins. Characteristics of application software maintenance. *Communications of ACM*, 21, June 1978.

[MO83] R. J. Martin and W. M. Osborne. Guidance on software maintenance. National Bureau of Standards Special Publication 500–106, Superintendent of Documents, Washington DC, 1983.

[SK95] B. Smyth and M. Keane. Remembering to forget : A competence-preserving case deletion policy for case-based reasoning systems. *International Joint Conference on Artificial Intelligence*, 1:377–382, 1995.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval.* Computer Science Series McGraw Hill Publishing Company, New York, 1983.

[ST96] H. Shimazu and Y. Takashima. Detecting discontinuities in case-bases. *Proceedings of the Thirteenth National Conference on Aritifical Intelligence*, 1:690–695, 1996.

This article was processed using the LaTeX macro package with LLNCS style