Short Papers

Redundancy Detection in Semistructured Case Bases

Kirsti Racine and Qiang Yang

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 or semistructured; 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 semistructured case bases. We focus on a difficult problem in case base maintenance: redundancy detection. This problem is particularly pervasive when one deals with a semistructured case base. We will discuss an information-retrieval-based algorithm and an implemented system for solving this problem. As the ability to contain the knowledge acquisition problem is of paramount importance, our method allows one to express relevant domain expertise for detecting redundancy naturally and effortlessly. Empirical evaluations of the system demonstrate the effectiveness of the methods in several large domains.

Index Terms—Data mining, knowledge and case base maintenance, knowledge acquisition in expert systems.

1 Introduction

CASE-BASED reasoning is a problem solving and knowledge reuse technique that is gaining rapid industry acceptance and is increasingly used in commercial and industrial applications [5]. To solve a problem, a case-based reasoner recalls previous situations *similar* to the current one and *adapts* them to help solve the current problem. The existing problem descriptions and solutions, known as *cases*, are used to suggest a means of solving the new problem, to warn the user of possible failures that have been observed in the past, and to interpret the current situation. In many practical application domains, this technique is more effective in solving certain problems than *rule-based expert system* approaches since it can store entire problem-solving cases for later analysis, rather than asking the domain experts to encode their knowledge in the form of rule-like languages. There are many examples of successful case-based reasoning applications, most of which are in help desk application domains. For instance, COMPAQ applied a CBR system from Inference Corporation to a help-desk system for suggesting repairs to printers [9]. Many more examples can be found in the CBR literature [7], [6], [16].

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. 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 much redundancy within a case base. These problems can potentially harm the accuracy and the speed 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. As time passes, cases may become redundant simply because there are more powerful cases in the same case base. A need then arises for identifying these cases and deciding whether to eliminate them. Added to the problem of redundancy is the fact that many legacy case bases consist of nonrelational, textual data. How to efficiently maintain and update these semistructured case bases remains an open issue.

In the past, researchers have attempted to address various aspects of the case base maintenance problem. Aha [1] presents several case-based learning (CBL) algorithms which are tolerant of noise and irrelevant features. These algorithms classify the attributes of cases in a case base. The concept description of a training case can be used to predict attribute values in future cases, thereby detecting anomalies and filling in missing information. Other work in case-based learning include [12], [13], [14]. These algorithms can be seen to focus on learning and organization at the *feature* level. To provide maintenance support at the case level, Smyth and Keane [15] suggest a competence preserving deletion approach. Classifications are made according to two key concepts: *coverage* and *reachability*. A heuristic order is provided to delete cases according to the classification which promises to preserve the competency of the case base. Similarly, in expert systems research, researchers have developed frameworks for the detection of possible anomalies or redundancy [4], [10]. Like the case-based learning approaches, the researchers assume that the knowledge base is well-structured in either relational or rule-based formats.

In contrast with the previous work, we will focus on case bases that are semistructured. The cases in these case bases are described in natural language text, separated by several fields. Given these case bases, we tackle the problem case base maintenance with large sizes. The maintainer is assumed to function in the life cycle of a case base in an organization. Our main contribution consists of three linked aspects of case base maintenance. First, we present a method to normalize an unstructured case base using information retrieval techniques. Second, we explore redundancy detection algorithms that operate on semistructured case bases. These algorithms allow a case base to stay compact and correct, improving the efficiency and effectiveness of reasoning. Finally, we validate the power of the theory in a prototype system and show that the algorithms scale up very well with realistic, large and semistructured case bases. In addition, we point out the limits of our work and suggest future extensions.

1.1 Case Base Maintainer Design

Our approach to solving the maintenance problem for very large legacy case bases is to integrate a maintenance system within a case-based reasoning system. The system is used to automate time consuming tasks in case base maintenance, as discussed in Section 2. A block diagram of the system design for case authoring is shown in Fig. 1.

- *K. Racine is with IBM Canada Lab., 12 concorde Pl., Suite 500, Toronto, Ontario, M3C 3R8 Canada. E-mail: kracine@ca.ibm.com.*
- *Q. Yang is with the School of Computing Science, Simon Fraser University, Burnaby BC V5A 1S6 Canada. E-mail: qyang@cs.sfu.ca.*

Manuscript received 30 Sept. 1997; revised 25 June 1998; accepted 22 Mar. 2000.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 105734.

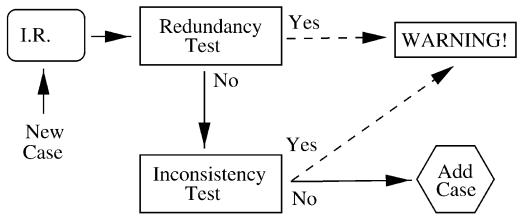

Fig. 1. Block diagram of agent architecture for case authoring.

The current dominant metaphor in case-based reasoning in terms of management is direct manipulation. That is, the user is required to supervise all events and to initiate all tasks. We suggest that the system should be designed to implement a complementary style of interaction, called indirect management [8]. As such, we will call our maintenance system an *agent*.

The indirect management approach mimics this interaction. Therefore, it seems that indirect management is the most reasonable type of management to apply to case base reasoning systems. Due to the nature of the domains to which case-based reasoning is applied, and the lack of domain models used in case-based reasoning, a fully automated management system could not be formally specified. Therefore, any actions that it performed may harm, rather than enhance, the competence of the case-based reasoner. An indirect management system, however, will still allow the user ultimate control while automating time consuming and difficult sections of the overall task.

In order to minimize the knowledge acquisition bottleneck, the agent must allow unstructured cases to be processed, as well as semistructured ones. The solution is an information retrieval-based algorithm to parse the cases by mining key words and important key word phrases from the unstructured text. These key words and phrases will offer the basis on which subsequent modules can operate.

The redundancy detection module will take an incoming case from the information retrieval (I.R.) module and determine whether it is redundant given the cases already existing in the case base. The redundancy detection algorithm relies on the successful completion of the information retrieval module in order to correctly identify the key words and phrases in the incoming case. The detected redundant cases will both be presented to the user along with a system suggestion explaining why redundancy was identified. The user can then choose to ignore the warning or delete one of the cases. The inconsistency detection module will use information from the inconsistency triggers and the cases and determine possible consistency problems. In this paper, we will focus on redundancy detection only.

In the next several sections, we will explain the algorithhm design for each of the three main modules.

2 The Information Retrieval Module

We use information retrieval techniques to normalize the cases. The purpose of applying this preprocessing module is to transform an unstructured case to a structured one so that we can perform subsequent algorithms for redundancy detection.

Information retrieval techniques have successfully been applied to case bases in the legal domain [3] and to large databases [11]. The architecture of an information retrieval system is simple. Each information retrieval system consists of a set of documents, a set of queries, and a similarity mechanism to determine which documents satisfy a query. It is typically difficult to directly compare two free-form text documents, so the similarity mechanism converts both the query and the set of information items into a standard format.

To illustrate design of the module, we will work through a cable TV example taken from realistic equipment troubleshooting cases at Rogers Cablesystems Ltd. Vancouver Call Center. In Appendix A, we show more examples of cases from that domain. The example is shown in Table 1.

2.1 General Algorithm

We describe the algorithm in the following steps:

Step 1. Removing the Stop Words. The first step in the information retrieval algorithm is to remove the stop words. Stop words are those words proven to be poor indexers, such as "the" and "of." These words do not add any meaning to the case. Stop words typically comprise 40-60 percent of the words within a document [11]. The application uses a general list of stop words generated for the English language used by the SMART system designed by Salton and McGill [11]. This list of stop words can be edited by the user in order to specialize it for a particular domain. For example, in a case-based reasoner designed to diagnose printer problems, the user may want to remove the words "printer" and "page" from consideration during the key word extraction phrase.

After the stop words have been extracted, the case in Table 1 will contain the information shown in Table 2.

Step 2. Domain Thesaurus. This function collapses 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 failures. These can both be reduced to "sega player" in order to facilitate string matching. The thesaurus can be edited iteratively as users become more familiar with the domain specific language. The user may choose not to use a thesaurus at all.

Step 3. The Stemming Algorithm. The stemming algorithm removes the suffixes and prefixes from each word in the case base. Stemming is used to reduce the number of distinct terms and to improve retrieval. There are a number of available stemming algorithms varying from removing almost all possible prefixes and suffixes to removing only those suffixes that pluralize a word. A reduced stemmer can typically be used

TABLE 1
Example Case in the Cable Domain

Case Name: no cable; black screen; tune local channel
Case Problem Description: This may be a problem with your TV or the cable system.
Case Solution: Once you have checked the electrical connections, tune the TV set to channel 3 and disconnect the cable. If there is still no reception, the problem is most likely in the TV set.

TABLE 2
Example of Removal of Stop Words

Case Name: no cable black screen tune local channel
Case Problem Description: problem tv or cable system
Case Solution: checked electrical connections tv set channel 3
disconnect cable still no reception problem tv set

in case-based reasoning as most cases are written in present tense to reduce the amount of typing required by the case author. However, if the case base is developed from existing data sources, a full stemming algorithm may be required.

The advantage of using a stemming algorithm is to further reduce the number of distinct words for consideration. A stemming algorithm will reduce the words "hook," "hooked," and "hooking" to the word "hook." This should increase the number of key words and phrases identified by the algorithm.

Step 4. The Inverted Index. After the preprocessing steps have been completed, the application generates an inverted index for the entire case base. The index is simply a listing of all terms that still remain in the set of cases, their weight within each document, and the document number in which they appear. The weight of a term within a document is simply a measure of the frequency that the term appears within that case. This measure provides information regarding the statistical importance of a term. Inverted indices may also contain information reflecting the position of the term within the case. However, due to the fact that this application was developed to handle large case bases, this information is not retained. The inverted index may already be quite large.

Step 5. The Key Word Index. After the inverted index is created, the next step in the algorithm is to build the key word index. Using the inverted index, this function identifies significant terms through statistical measures. Key words are those words which appear frequently within a small set of cases and infrequently across all other cases [2], [11]. This application uses the inverse document frequency measure to identify key words [11].

The key word, the weight of the key word within the file, and all document numbers in which the key word appears are retained in a key word index. The application retains all of the documents' numbers in order to facilitate redundancy detection in later stages.

The key word file can be edited by the user after its creation. The user may wish to add or delete some of the given key words. If the user adds a key word, the system identifies in which cases the new word appears and accordingly updates the key word file. At run time, the user can specify the number of key words to be identified and the minimum number of documents in which they must appear. The user is provided with the number of words and the number of cases within the document to facilitate a decision on these particular thresholds.

Step 6. The Key Phrase Index. The application also identifies key phrases using the inverted index. Phrases are groups of more than one word which have high intercase cohesion [11]; if one word appears in a case, then the other words have a very high probability of also appearing. The phrases and their corresponding weights are retained. Identified phrases must appear in > T cases, where T is a standard or user-specified threshold. Phrases can be more powerful than key words as they add some context to the statistical approach to information retrieval. To reduce the number of phrases identified by the algorithm and to increase their relative importance, there is an additional constraint that at least one word in the phrase must be a key word.

3 The Redundancy Detection Module

Once a standard description or profile has been generated for each case, redundancy and subsumption can be partially identified. The redundancy detection module receives the cases from the information retrieval module. These cases have been normalized: The stop words have been removed, the terms have been stemmed, the thesaurus has been applied, and the indices have been built.

3.1 Approximate Equivalence

An obvious instance of redundancy occurs when two cases have identical string representations. The use of information retrieval techniques to remove the stop words, stem each term, and reduce synonyms can assist in increasing the similarities between cases. However, it is still unlikely that two cases will have exactly the same representation. A more interesting situation occurs when the cases share similar case representations according to a fuzzy string matching algorithm. The matching is done with emphasis on the key words that the two cases share. If they are identified as redundant, they are then presented to the user for further analysis.

We apply a trigram matching algorithm to identify key word strings that are close to each other. Trigram matching is the pattern matching algorithm used for query retrieval in CaseAdvisor™ Problem Resolution [17]. The advantage of using this algorithm is that it is a fuzzy matching algorithm. Anomalies such as spelling, punctuation, and word order can be partially ignored by this algorithm.

Essentially, this algorithm divides the strings to be compared into trigrams (substrings of length three). If a trigram is found in both strings, a hit is recorded. The resulting "score" is the percentage of trigrams that existed in both strings given the total number of trigrams in the longer string. Thus, this algorithm is tolerant of spelling errors and word order. Using trigram matching on $S_1 = \text{cable}$ and $S_2 = \text{cabel}$, the comparison will result in a nonempty match.

Although a drawback of using trigram matching is that cases that are morphologically related will return high scores even if they are not semantically related, part of the problem has already been solved by the application of the thesaurus in the information retrieval module.

3.2 Domain-Independent Subsumption Rules

Cases can also be redundant because they are subsumed by other cases. In this situition, the subsumed cases can be removed from the case base without affecting the overall competence of the case-based reasoning system.

We adopt a uniform notation for representing a case. Let $\texttt{Case} = \langle (P), (S) \rangle$ be a case. In the definition, P represents a set of normalized key words or key phrases denoting a problem description for the case. Each element p_i of P represents a distinct problem keyword or key phrase such as "screen black" in a Cable TV troubleshooting case. S is the solution for the case; in the Cable TV example, S might be "Call Sony at 1-800-... ." In general, a solution S for a case is a sequence of steps such that, upon complete execution of S, the problems described in P can be solved.

We can now state our case-subsumption rule informally as follows: Suppose we are given two cases: $\texttt{Case}_1 = \langle (P_1), (S_1) \rangle$ and $\texttt{Case}_2 = \langle (P_2), (S_2) \rangle$. $\texttt{Case}_1$ subsumes $\texttt{Case}_2$ if $\texttt{Case}_1$ solves more problems than $\texttt{Case}_2$ (P_1 is a superset of P_2) and the solution of $\texttt{Case}_1$ requires fewer steps than that of $\texttt{Case}_2$. In this situation, S_1 is

a subset of S_2.[1] If $\texttt{Case}_1$ subsumes $\texttt{Case}_2$, we say that $\texttt{Case}_2$ is redundant.

The subsumption rule suggests two different redundancy algorithms. The first algorithm is designed to determine whether an incoming case is redundant given the cases in the case base. The second algorithm uses the entire case base as input and determines if redundancy exists within the case base. For both algorithms, we first apply a nearest neighbor algorithm to find out, for each case to be tested for redundancy, the candidate set of cases which are likely to be redundant. These cases can be found out using the native nearest-neighbor algorithm associated with a case base reasoning system. For a very large case base, this initial preprocessing reduces the set of cases we have to consider.

For the first algorithm, R_1, the worst case occurs when every single case existing in the case base is involved within a redundancy relationship with the incoming case. Let N be $\|CaseBase\|$. The number of key word index checks is $O(N)$ and the number of redundancy detections is $O(N)$. Therefore, the overall complexity is $2(O(N)) = O(N)$.

The second algorithm R_2, determining if redundancy exists at all in the case base, has an added layer of complexity. The worst case again exists when every single case is involved in a redundancy relationship with all other cases. For each of the N cases, R_2 calls R_1 as a subroutine. Thus, the overall complexity is $O(N) * O(N) = O(N^2)$.

Both algorithms try to provide the user with an explanation as to why the cases were identified as redundant. Often, the user does not have enough domain knowledge to isolate dependencies within cases. The two original cases are presented to the user along with the key words that exist in both cases, highlighting the similarities. If the two cases are considered candidates for merging, a notice is sent to the user including the application's suggestion.

Additionally, both algorithms rely on thresholds to determine when keywords extracted from cases are approximately identical. These thresholds are originally set by the application as default values, but can be changed by the user for each particular domain.

3.2.1 Redundancy Removal

When deciding to delete subsumed cases, the case-maintainer system should allow the user to view both cases and highlights the unnecessary condition. As it is possible that $\texttt{Case}_1$ is an incorrect or outdated case, the fact that it subsumes $\texttt{Case}_2$ does not mean that $\texttt{Case}_2$ should be summarily deleted from the case base. Rather than simply deleting the cases identified as subsumed, the application presents these cases to the user together with reasons why they are believed subsumed. This is because a typical user of the application may not be familiar enough with the domain to delete the case that offers more information. Perhaps the extra premise offers valuable information to the novice user that the case that subsumes it does not.

3.3 Merging Cases

A redundancy identification module should also be able to detect cases that are candidates for merging. For example, if two cases offer the same solution but slightly different problem descriptions, it is likely that the cases can be collapsed into one. Please note that if the differences within the problem description field are not considered significant by the application, then the system suggestion will state that the cases are essentially equivalent and the user may choose to keep either or both cases.

1. We have assumed that solution length is a measure of solution quality in this section. In some domains, there are other measures of solution quality, such as the cost of solution, etc. Also, it may be the case that a solution is longer because it contains more explanatory data. Extensions of our subsumption rules in these areas can be done; however, we will focus on solution length here for simplicity.

TABLE 3
Detecting Redundancy in the Printer Repair Domain

Case 1
Problem Description: envelopes jam laser printer due to glue.
Case Solution: Normal envelopes and laser printers do not get along well together. Problems include poor glue heat tolerance.
KEYWORDS : **envelopes jam laser glue heat** tolerance
Case 2
Problem Description: Paper continues jamming printer due to sticky internals
Case Solution: Envelopes do not work very well with laser printers. The high heat melts the glue.
KEYWORDS : **jam**ming sticky **envelopes laser heat glue**

3.4 Relaxed Redundancy Detection Rules and Implementation

To make practical use of the rules, their computation needs to be relaxed. The relaxation is aimed at catering to the fact that the cases may not be defined formally. Therefore, some margin of approximation must be left when detecting redundancy.

Table 3 illustrates the necessary information to detect redundancy, using an example from a printer-repair domain. Using the key words that have been extracted from each case, the first step is to determine the extent that the key words match. If Case 1 and Case 2 share more than some threshold T of key words, the two cases are considered further for redundancy. For example, in Table 3, Case 1 and Case 2 share five (5) key words. Each case has six (6) key words. Therefore, these cases share 83 percent of key words. The application initially sets the redundancy threshold to 80 percent, so if this threshold is not modified by the user, then the application will signal redundancy for these two cases.

This comparison of key words is facilitated by the key word index developed in the information retrieval module. All of the information is extracted from the index, rather than the cases themselves, in order to make this process quicker. If two cases "succeed" on the key word matching, a further check on the redundancy rules is performed on the entire case content (including descriptions and solutions) using fuzzy string matching. After this step, if the two cases have been identified as possibly redundant, both cases in their entirety are presented to the user, who determines if there is in fact redundancy and, if so, which case should be removed from the case base. The user is also offered the option of further editing one or both cases to make the distinction between them more apparent.

3.4.1 Related Work

Relating to the work by Smyth and Keane on competence-preserving methods for managing a case base, the subsumption rules defined above provide a significant operational advantage. In Smyth and Keane's work [15], all definitions of auxiliary, spanning, and support cases are defined in terms of problem coverage and reachability. These definitions have the problem of computational inefficiency since to compute the coverage and reachability of a case in terms of incoming problem queries is very expensive. Our subsumption rules, on the other hand, provide a problem-independent approach to finding redundant cases; instead of computing coverage and reachability for each case using all user problems, we compare the contents of the two cases directly. This direct comparison enables us to deduce whether one case makes another case redundant.

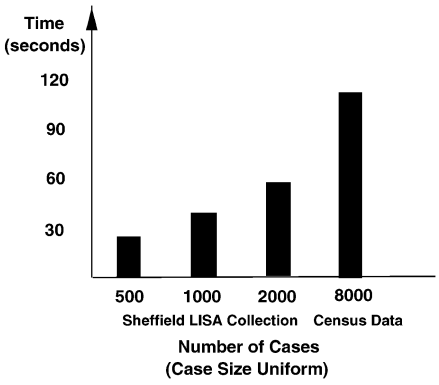

Fig. 2. CPU time to apply information retrieval techniques.

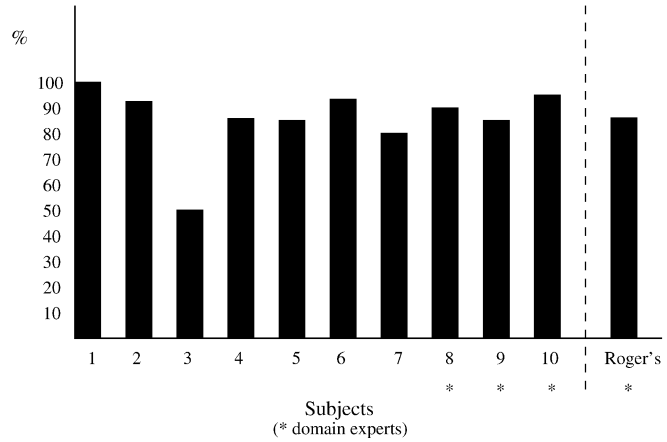

Fig. 3. Percentage of key words matching those found by Case Maintainer.

4 Empirical Testing

We have implemented the maintenance system in the framework of the CaseAdvisor™ system developed by the Case Based Reasoning Group at Simon Fraser University. CaseAdvisor™ is a case-based reasoning system implemented at Simon Fraser University.

Our tests are aimed at establishing the validity of our approach to case base maintenance. We hope to confirm through the experiments the following conjectures: First, The information retrieval approach for processing unstructured or semistructured cases is feasible for large case bases. Evidence supporting this conclusion will be based on a comparison of CPU time and case base sizes. Further experimentation was conducted to demonstrate the accuracy of the key word and phrase extraction algorithms. Second, the redundancy detection module is capable of detecting most redundant cases when cases are derived from one source. This will be shown through a controlled experiment where some redundant cases are introduced by the experimenter. Results as to the degree of these cases discovered by the agent will be used to justify this claim. All tests reported below were conducted on a Sun Sparc 5 workstation.

4.1 Testing the Information Retrieval Module

Testing was completed on large test files to illustrate how the information retrieval module scales up. Fig. 2 demonstrates that even the one-time cost of normalizing a case base is not that expensive. The time displayed is the CPU time required to remove the stop words from all of the cases, stem all of the terms, apply the user defined thesaurus, to extract key words and phrases from the cases, and to build the inverted file structure. The information retrieval module was applied to a number of 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. The empirical testing proves that the information retrieval module can handle cases of that size in a reasonable amount of time.

After testing the efficiency of the Information Retrieval Module, it was necessary to test the efficacy. Key words and phrases can be extracted from text files in a reasonable amount of time, but are they useful? To test the accuracy of the key word and phrase extraction procedures, the application was challenged by humans. Given the same information and the same text file, the key words and phrases that the subjects extracted were compared to those extracted by the application. To test the importance of these words and phrases, each one was used as a query to the case base developed by the Case Based Reasoning Group at Simon Fraser University for Roger's Cablesystems Ltd. in Vancouver, British Columbia, Canada.

When applied to an actual case base designed to troubleshoot Cable TV failures, the information retrieval module completed processing in less than one second. There were ten (10) subjects involved in this experiment. The text file was provided by Roger's Cable and contains 42 cases. On average, the subjects required approximately twenty (20) minutes to extract key words from the text file. All of the subjects had some knowledge of the cable domain, but only three (3) considered themselves experts. Despite this range of familiarity with the domain, the lists provided by the subjects showed great overlap. Fig. 3 illustrates the similarities between the key words generated by the subjects and the key words generated by Case Maintainer. Similarities were measured by exact matches and substring matches. As Case Maintainer performs term stemming, the terms "channels" and "channel" are both represented as "channel." Therefore, if the subject included the word "channels," it was marked as a match. The final list of key words was extracted from the case base developed at Roger's Cable.

Clearly, the automatic generation of key words is successful in the Cable TV domain. Without tweaking, the key words generated by Case Maintainer matched 87 percent of the key words generated by those familiar with the cable TV domain. The only list with which Case Maintainer did not match at least 80 percent of terms was a whopping 116 words provided by a subject with limited domain expertise and computer experience.

4.2 Testing the Redundancy Detection Module

The redundancy module is responsible for testing an incoming case for possible redundancy. Recall that the cases are first normalized by the I.R. module. 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. 4 demonstrates that the algorithm to detect redundancy is efficient enough to be applied in a case authoring module. At the time of this testing, the CaseAdvisor™ system could only output to flat files. The ability to output to ODBC (Open Data Base Connectivity) databases has recently been added to the system. Further testing will be required to determine the processing time the SQL (Standard Query Language) interface will add.

The average size of the case is also included in Fig. 4 to illustrate that 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, 8,192, 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

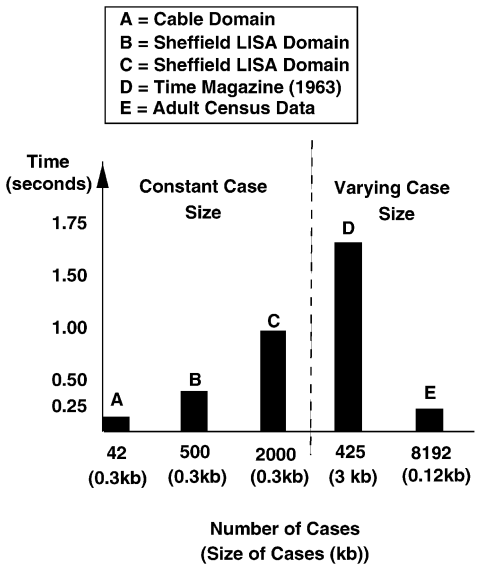

Fig. 4. CPU time to detect redundancy.

redundancy. However, the system still performed the redundancy test in less than two seconds.

The next experiment involved using subjects to type in cases from the cable domain. Five (5) subjects were required to input cases and submit them to be added to the case base. Approximately 50 percent of required cases to be entered were, in fact, redundant. The subject was not given any information regarding which cases had already been entered into the system. Table 4 presents the results of this experiment.

Table 4 demonstrates the efficacy of the redundancy module. Ninety-four percent of the redundant cases were correctly identified by the application. Another encouraging statistic is that 83 percent 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, six were falsely identified as not redundant, and the remaining 87 cases were correctly classified as not redundant. This means that 88 percent of the 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. An additional area of improvement is that all of the cases involved in this experiment were derived from the same source. As part of future work, it would be interesting to see how the above results generalize to cases authored by different case authors at different times.

TABLE 4
Quality of the Redundancy Module

	Identified	Not Identified	
Redundant	97	6	103
Not Redundant	20	87	107
	117	93	

5 Conclusions and Future Work

Case base management should be taken seriously by every practitioner and researcher in the case-based reasoning area. Of high importance is the issue of how to contain knowledge acquisition costs while maintaining the case base. Our answer to the question is a case base maintenance agent structure which can retrieve important information from a case base and then use the information to detect redundant cases. Our experiments strongly indicate that the approach can be used to address practical problems of large sizes.

One area of future work is exploring methods to dynamically reorganize a case base. We plan to design a case base organization agent in the future. The agent will also dynamically organize and reorganize the case base in a hierarchical manner so as to maximize usability. Cases that are required on a frequent basis will be quickly accessible by the case-based reasoner. This organization is done throughout the life time of the case base. As a case becomes more infrequently used, its ranking within the case base will decline.

Acknowledgments

We wish to thank David Aha, D. Edward Kim and Philip W.L. Fong and Christina Carrick for valuable comments. The authors are supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), an Ebco/Epic NSERC Industrial Chair Fund, BC Advanced Systems Institute, and Canadian Cable Labs Fund.

References

[1] D. Aha, "Case-Based Learning Algorithms," *Proc. 1991 DARPA Case-Based Reasoning Workshop*, vol. 1, pp. 147-158, 1991.
[2] W.B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.
[3] D. Gelbart and J.C. Smith, "Towards Combining Automated Text Retrieval and Case-Based Expert Legal Advice," *Law Technology J.*, vol. 1, pp. 19-24, 1992.
[4] H. Kaindl, "Verification and Validation of Knowledge-Based Systems Using Semiformal Representation," *Proc. AAAI 96 Workshop Verification and Validation of Knowledge-Based Systems*, pp. 7-16, 1996.
[5] J. Kolodner, *Case-Based Reasoning*. San Mateo, Calif.: Morgan Kaufmann, 1993.
[6] J. Kolodner and R.L. Simpson, "The MEDIATOR: Analysis of an Early Case-Based Problem Solver," *Cognitive Science*, vol. 13, no. 4, pp. 507-549, 1989.
[7] D. Leake, *Case-Based Reasoning—Expriences, Lessons and Future Directions.* AAAI Press/MIT Press, 1996.
[8] P. Maes, "Agents that Reduce Work and Information Overload," *Comm. ACM*, vol. 37, no. 7, pp. 31-41, 1994.
[9] T. Nguyen, M. Czerwinski, and D. Lee, "Compaq Quicksource—Providing the Consumer with the Power of AI," *AI Magazine*, 1993.
[10] A.D. Preece, "Towards a Methodology for Evaluating Expert Systems," *Expert Systems*, vol. 7, no. 4, pp. 215-233, 1990.
[11] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval*. New York: McGraw Hill, 1983.
[12] J. Shavlik, "Finding Genes by Case-Based Reasoning in the Presence of Noisy Case Boundaries," *Proc. 1991 DARPA Workshop Case-Based Reasoning*, vol. 1, pp. 291-303, 1991.
[13] H. Shimazu and Y. Takashima, "Detecting Discontinuities in Case-Bases," *Proc. 13th Nat'l Conf. Aritifical Intelligence*, vol. 1, pp. 690-695, 1996.
[14] E. Simoudis, "Using Case-Based Retrieval for Customer Technical Support," *IEEE Expert*, vol. 7, no. 5, pp. 7-13, 1992.
[15] B. Smyth and M. Keane, "Remembering to Forget: A Competence-Preserving Case Deletion Policy for Case-Based Reasoning Systems," *Proc. Int'l Joint Conf. Artificial Intelligence*, vol. 1, pp. 377-382, 1995.
[16] I. Watson, *Applying Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann, 1997.
[17] Q. Yang, E. Kim, and K. Racine, "Caseadvisor: Supporting Interactive Problem Solving and Case Base Maintenance for Help Desk Applications," *Proc. Int'l Joint Conf. Artificial Intelligence (IJCAI 97 Workshop on Practical Applications of CBR)*, Aug. 1997.