

Design and Evaluation of a Self Cleaning Agent for Maintaining Very Large Case Bases (VLCB)

by

Kirsti Racine

B.Sc.H Queen's University 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Computing Science

© Kirsti Racine 1997
SIMON FRASER UNIVERSITY
June 1997

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

APPROVAL

Name: Kirsti Racine
Degree: Master of Science
Title of thesis: Design and Evaluation of a Self Cleaning Agent for Maintaining Very Large Case Bases (VLCB)

Examining Committee: Dr. Bill Havens
Chair

Dr. Qiang Yang
Senior Supervisor

Dr. Jiawei Han
Supervisor

Dr. Robert Hadley
Examiner

Date Approved: _____

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

**Design and Evaluation of a Self-Cleaning Agent for
Maintaining Very Large Case Bases (VLCB).**

Author:

(signature)

(name)

(date)

Abstract

The objective of this thesis is to establish a theoretical and empirical framework for the design of an agent which can maintain very large unstructured case bases, ensuring that they remain current and useful. With the dramatic proliferation of case based reasoning systems in many commercial applications, many case bases are now becoming legacy data sources. While they represent a significant portion of an organization's assets, they are large and difficult to maintain. There are many sources for the complexity of the maintenance task: case bases are created over a long period of time and are updated by different people; high industry turnaround suggests that the authors of a case base may not still be with the organization; cases may be obtained from different, even geographically diverse sources; and finally, industry markets change, implying that the case bases are highly time and market dependent. These factors contribute to the difficulties in ensuring a case base is current and useful.

My solution to the maintenance problem is to develop a self cleaning agent which works with users in maintaining a legacy case base in a seamless fashion. This self cleaning module features a set of user entered guidelines for detecting redundant and outdated cases. The guidelines are written in a language that is easily manipulatable by any non-expert user. As the ability to contain the knowledge acquisition problem is of paramount importance, using this system allows one to express domain expertise naturally and effortlessly. Empirical evaluations of the system prove the effectiveness of the agent in several large industrial domains.

Contents

Abstract	iii
List of Tables	viii
List of Figures	ix
1 Background	1
1.1 Introduction to Case Based Reasoning	1
1.2 Areas of Research in Case Based Reasoning	3
1.2.1 The Case Base	3
1.2.2 Indexing	4
1.2.3 Retrieval	4
1.2.4 Adaptation	6
1.3 Case Based Reasoning and Other Techniques	7
1.3.1 Case Based Reasoning and Expert Systems	7
1.3.2 Case Base Reasoning and Databases	8
1.4 Application Examples	8
1.4.1 General Dynamics	8
1.4.2 Lockheed	9
1.4.3 Other Applications	9
1.5 Discussion	11
2 Case Based Management	12
2.1 Previous Approaches	14
2.1.1 Case Base Learning Approaches	14
2.1.2 Knowledge Discovery Research	17
2.1.3 Expert System Research	17
2.1.4 Case Base Maintenance Approaches	18

	2.1.5	Information Retrieval Applied to Case Based Reasoning . . .	21
	2.1.6	Agents and Indirect Managing	22
	2.2	Outline	23
3		The Case for Case Base Maintenance	25
	3.1	Two Types of Cases	25
	3.1.1	The Redundant Case Problem	27
	3.1.2	The Inconsistent Case Problem	29
	3.1.3	The Dynamic Reorganization Problem	30
	3.2	Criteria for a Competent Case Base Maintenance Agent	31
	3.2.1	Consistency	31
	3.2.2	Qualitative Competence Criteria	31
	3.3	Agent Design	32
	3.3.1	Case Authoring	32
	3.3.2	Problem Resolution	33
	3.4	Discussion	34
4		The Information Retrieval Module	35
	4.1	Why Information Retrieval?	35
	4.2	General Algorithm	36
	4.2.1	Removing the Stop Words	37
	4.2.2	Domain Thesaurus	37
	4.2.3	The Stemming Algorithm	37
	4.2.4	The Inverted Index	38
	4.2.5	The Key Word Index	38
	4.2.6	The Key Phrase Index	38
	4.3	Example	39
	4.4	Design Rationale	40
	4.5	Discussion	41
5		The Redundancy Detection Module	42
	5.1	Equivalence and Pure Subsumption	42
	5.2	Pure Subsumption	43
	5.3	Merging Cases	45
	5.4	An Example	45
	5.5	Redundancy Detection Algorithm	46

	5.5.1	Trigram Matching	47
	5.5.2	Check Incoming Case for Redundancy	48
	5.5.3	Check Entire Case Base for Redundancy	49
	5.6	Design Rationale	49
	5.7	Discussion	50
6		The Inconsistency Detection Module	51
	6.1	Addressing the Knowledge Acquisition Problem	51
	6.2	Guideline Representation	52
	6.3	Inter-case Representation	53
	6.4	Inconsistency Detection Algorithms	53
	6.4.1	The Intra-case Inconsistency Detection Algorithm	54
	6.4.2	An Example	54
	6.5	Design Rationale	56
	6.6	Discussion	56
7		Empirical Testing	58
	7.1	Testing the Information Retrieval Module	59
	7.2	Testing the Redundancy Detection Module	61
	7.3	Testing the Inconsistency Detection Module	64
	7.4	Testing the Competence of the Case Based Reasoner	65
	7.5	Discussion	71
8		System Development	72
	8.1	Information Retrieval Module	72
	8.1.1	Stop Word Removal	72
	8.1.2	Key Word Extraction	74
	8.1.3	Editing the Key Word File	75
	8.1.4	Phrase Extraction	75
	8.1.5	Editing the Phrase File	76
	8.1.6	Question Extraction	77
	8.2	Redundancy Module	77
	8.3	Inconsistency Module	79
	8.4	Adding a New Case	80
	8.5	Merging Two Case Bases	80
	8.6	Save the Case Base	80

8.7	Discussion	80
9	Conclusions and Future work	82
9.1	Contributions	82
9.2	Future Work	83
	Appendix: A.) A Decision Tree	85
	Appendix: B.) Cable Cases	87
	Bibliography	93

List of Tables

1.1	An Example Case	4
1.2	What are Companies Using CBR For? [adapted from [54]]	10
3.1	Example cases for automobile diagnosis and repair.	26
3.2	Example of redundant cases in the printer repair domain	28
4.1	Example of Information Retrieval Techniques Applied to Incoming Case	39
5.1	Candidates for Merging	45
5.2	Detecting Redundancy in the Printer Repair Domain	46
6.1	Example of a Normalized Case and the Guideline it Violates	55
7.1	Typical Case Extracted from the Sheffield LISA Collection	60
7.2	Sample Case and Automatically Generated Guidelines	64
7.3	First Case Retrieved Given Key Words For Each Case Base	67
7.4	Educational Status of Subjects from Competence Experiment	68
7.5	Competence of the Two Case Bases	69
7.6	Comparing the Case Bases	70
8.1	Example Case in the Cable Domain	73
8.2	Example of Removal of Stop Words	73
8.3	Example Inverted Index	74

List of Figures

1.1	The Process of Case Based Reasoning and Area of Focus	2
2.1	Illustrating Pivotal and Auxiliary Cases [adapted from Figure 1(a) in [48]] . .	20
3.1	Dimensions of Inconsistency Within a Case Base	30
3.2	Block Diagram of Agent Architecture for Case Authoring	32
3.3	Block Diagram of Agent Architecture for Problem Resolution	33
4.1	My Information Retrieval Agent Design For Case Based Reasoning [adapted from [41]]	36
7.1	CPU Time To Apply Information Retrieval Techniques	59
7.2	Percentage of Key Words Matching Those Found by Case Maintainer	61
7.3	CPU Time to Detect Redundancy	62
7.4	Quality of Redundancy Module	63
7.5	CPU Time to Detect Inconsistency	66
8.1	Thresholds for Key Word Extraction	75
8.2	Editing the Key Word File	76
8.3	Example of Possible Redundancy	77
8.4	Adding a sample case to the cable domain	78
8.5	Rule Violation in the Cable Domain	79
9.1	Sample Decision Tree for Redundancy Experiment	86

Chapter 1

Background

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 [21]. 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 problems than **rule based expert system** approaches, since it can overcome the so-called *knowledge acquisition bottleneck* by storing entire cases for later analysis, rather than asking the domain experts to encode their knowledge in the form of rule-like languages. Examples of successful case based reasoning applications are those where extensive previous knowledge exists in recorded forms, including a help-desk system for suggesting repairs to COMPAQ printers [32], and a system for assisting manufacturing design [19].

1.1 Introduction to Case Based Reasoning

Typically, the process of case based reasoning is described as a cycle. Figure 1.1 illustrates the process¹. The most effective mnemonic used to described this process is the four **Res** [1]:

¹Figure 1.1 is adapted from [1].

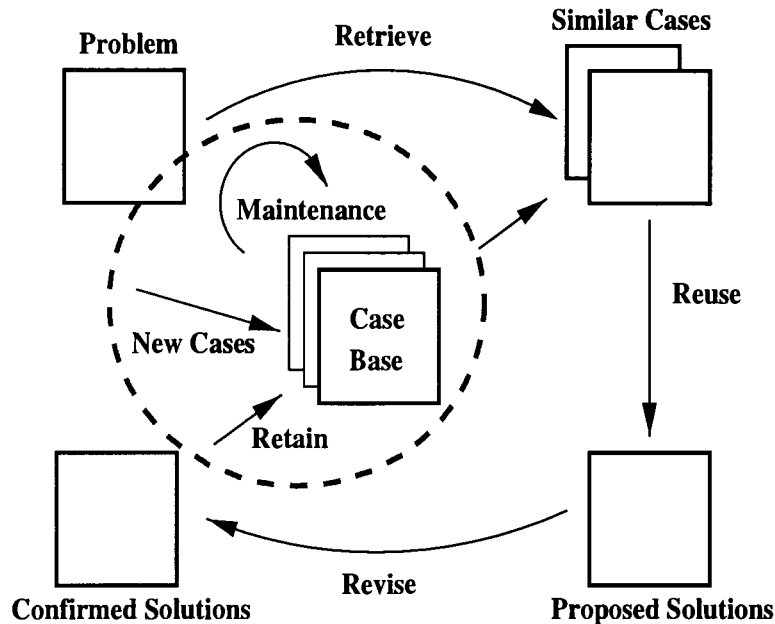


Figure 1.1: The Process of Case Based Reasoning and Area of Focus

1. **Retrieve** : Given the user's query, retrieve the most similar case(s) in the case base.
2. **Reuse** : Reuse the appropriate case to try to solve the problem.
3. **Revise** : Revise the current solution, if it is inadequate to solve the current problem.
4. **Retain** : Save the revised case as a new case in the case base.

Simplified, the process works as follows. The user formulates a query for the system; many systems use free form text queries, but the query structure is system dependent. The system uses the query to **retrieve** the appropriate case(s) that exist in the case base. Often, the system returns a list of cases that are given relative scores according to their similarity to the query. Either the system **reuses** the case with the highest score or the user is given the chance to select a case for **reuse** from a list of similar cases. If the current case will not appropriately solve the current situation, either the user or the system **revises** the case to fit the current problem. This newly generated case is **retained** in the case base, so that it will be accessible to the next user of the system.

In reality, most case based reasoning systems are retrieval and reuse systems [54]. Many systems return a number of similar cases given the user's query. Each of these cases is

associated with a score representing the similarity between the user's query and the case. Along with these cases are a number of questions that the user can answer to further focus the results. If the user is unsatisfied with the results produced by the query, s/he can answer the questions provided. As each question is answered, the scores for the cases are updated to reflect those answers. Generally, the case with the highest score is the one which the system believes is most suited to the user's query. In these systems, the cases are entered in a separate module; typically a set of cases is entered at the same time.

Clearly, case based reasoning is an interactive paradigm. The user is involved in much of the process. The user is given ultimate control over which case is selected for consideration and which questions s/he elects to answer to facilitate case selection. This is considered an advantage of case based reasoning [21, 54].

1.2 Areas of Research in Case Based Reasoning

1.2.1 The Case Base

The first step in developing a case based reasoner is to build a case base. A case base consists of a number of cases each of which describe an historical experience. At the very least, each case must comprise of a problem description detailing the situation in which the case occurred and a solution which presumably has some effect on that situation. Many different Artificial Intelligence techniques have been used to represent cases, such as frames, objects, predicates, semantic nets and rules [55]. Currently, there is no standard describing what information should be retained in a case [54]. Two factors that must be taken into account when deciding on a case representation are ease of acquisition and provided functionality [21].

The general structure of a case is standard, however. Each case is described by a collection of attribute value pairs. An example case is presented in Table 1.1. Knowledge elicitation in case based reasoning essentially is the identification of these attribute value pairs. Attributes may also be referred to as features. The main difference between cases represented in this manner is the granularity of the attribute value pairs. They can be very specific, such as the example in Table 1.1 or they can be very general. The latter type of case may just have two attributes *Case Name* and *Case Solution*, each associated with a paragraph of text.

Attribute	Value
Make	Mazda
Engine Size	8 cylinders
Price	12 300

Table 1.1: An Example Case

1.2.2 Indexing

Once the cases have been created, a representation approved and a case base begun, the developer must devise some method of indexing the cases to provide efficient retrieval. Some basic guidelines have been established by case based reasoning researchers. Essentially these guidelines state that indices should be [54]:

1. Predictive
2. Allow for an increasing case base
3. Concrete enough to be recognized in the future.

Traditionally indices used to be represented as pointers to cases. In case based reasoning, however, strategies such as applying importance values to cases or sections of cases have been used. Other indexing strategies include labeling cases with their important features. Real life case based reasoners such as CASEY, a case based reasoner developed to diagnose heart failures, not only store cases, but store the corresponding feature weights for the attributes within the cases [23]. FLEXICON, a case based reasoning system designed to train lawyers, stores case summaries or profiles as well as case histories [15].

1.2.3 Retrieval

Retrieval algorithms are responsible for retrieving the closest or most similar cases in a case base, given the user's query. Similarity can be measured in a number of different ways: nearest neighbour, induction, template retrieval or a combination of these strategies [4, 47, 54].

Case based reasoners are often used because they can provide quick solutions to the user's problems. It is critical that the retrieval strategy operates efficiently and correctly.

Nearest Neighbour

A nearest neighbour approach calculates similarity by matching a weighted sum of attributes. Each attribute must be assigned a weight that indicates its importance within the application domain. Problems with this approach include initializing the attribute weights, converging on the correct solution, and the retrieval time increasing linearly with the number of cases [54]. CASEY, a case based reasoner used to diagnose heart failures, uses this approach [23]. In each stored case, a set of attribute weights for that case is also stored. These can then be used in retrieval. CASEY employs a very rich case representation and includes background knowledge. However, some domains are so poorly understood that attribute weights cannot be extracted.

In Table 1.1, an example case is illustrated representing an automobile. A nearest neighbour algorithm may assign the attribute Price to have a higher weight than the Make or the Engine size. This would cause the value of the attribute Price to have a higher impact on the retrieval than the values of the other two attributes.

Induction

Inductive approaches usually make use of Quinlan's ID3 algorithm and its successor C4.5 [36] to determine which features are most important in discriminating cases. A decision tree is generated that may be used to organize cases in memory. This decision tree is a classification mechanism. Each branch represents an attribute value pair and each level of the tree represents an attribute. Additional information can be contained in the tree at each branch, including the cases that fit within the classification. This approach has been used to implement a case classification mechanism in the medical domain [47].

The disadvantage of using this approach is that the order of attributes is static. The top level of the tree always represents the same attribute and this approach is then useless if the value for that attribute is unknown. This reasoning can be applied at all levels of the decision tree. Typically this approach is most useful when there are a small number of attributes that dominate the case base [54]. In this case, a decision forest can be used; a number of decision trees each with a different ordering on the attributes. ReMind, a case based reasoning shell developed by Cognitive Systems Inc. and CBR Express developed by Inference Corporation both allow the user to use inductive retrieval approaches using decision trees [1]. In both systems, the user is allowed to edit the tree and to adjust values

to change attribute importance.

In Table 1.1, an example case is illustrated describing a car. An application using the induction process may decide that Price is the “most” important attribute and, therefore, assign that as the top level of the tree. If the user then types in a query that does not include the price, the application will need to prompt the user for a price range before continuing. This approach is effective when attributes have a very clear ranking of importance.

Template Retrieval

Template retrieval approaches return all cases that fall within specified parameters. In large case bases, a template retrieval algorithm may be applied before another retrieval technique to reduce the search space. Template retrieval algorithms work similarly to the SQL (Standard Query Language) used to extract information from databases. If a case base is stored in a data base, an SQL query can be generated to select only those cases which satisfy certain user and/or system set constraints.

Template retrieval is usually based on syntactic retrieval, as opposed to semantic retrieval. Although, syntactic retrieval is more superficial, some domains are so poorly understood that the use of semantic retrieval is impossible [21]. Successful applications that use template retrieval include the CYRUS system, a question and answer system with knowledge of the various travels taken by former US Secretary of State Cyrus Vance developed at Yale University. This system later served as the basis for CHEF [1], and CASEY [23] as well as others.

1.2.4 Adaptation

Adaptation (or revision) has presented difficulties for researchers. There are two main methods used to revise past cases [54]. The first, called transformational reuse, is to reuse the past case solution. The second is to reuse the past method that constructed the solution, referred to as derivational reuse.

In the first method, the case solution may not exactly match the current problem, but transformational operators exist such that when applied to the old solution, they can transform it into a new solution. This approach requires strong domain knowledge that can be represented as a set of transformation operators. CASEY, a case based reasoner developed at MIT to diagnose heart failure, uses this approach. CASEY’s case representation is very

rich providing a good depth of domain information. In CASEY, a new solution is built from an old one by rules that use a transformational operator as the action part of the rule [23]. New solutions are built according to differences between the retrieved case and the current case.

The second method, derivational reuse, examines how the problem was solved in the retrieved case. In this type of adaptation, the case must contain information justifying its current solution, the alternatives considered while generating the solution and the failed search paths. Case based reasoners using this approach then reconstitute these methods. During this process, the path that was taken in deriving the current solution will be avoided. The system will first take the successful alternatives and generate new subgoals. The Analogy/Prodigy system uses derivational adaptation [51]. Analogy/Prodigy is a case based architecture designed to facilitate learning and planning that can model many different domains. It uses two forms of knowledge: factual domain knowledge and control knowledge. The control knowledge drives the planning processes. Adaptation occurs by changing current plans according to the similarities between goals and initial solutions in other plans.

Both of these methods require a very rich case representation and a large amount of background knowledge. These methods are very domain dependent and the knowledge engineering required to implement them is expensive. As a result, very few case based reasoning systems use adaptation [54].

1.3 Case Based Reasoning and Other Techniques

1.3.1 Case Based Reasoning and Expert Systems

Traditional rule based expert systems have successfully been applied to many commercial applications. These systems use the current state of knowledge and background knowledge in the form of rules. The rules are applied until the system can reach a conclusion. One of the first expert systems developed was MYCIN, which was applied to the medical domain in order to aid doctors in deciding which therapy was appropriate for patients with bacterial infections such as meningitis. Despite the success of these types of systems, a number of problems have continually been reported [54]. The most difficult process in building an expert system is knowledge elicitation. This is often referred to as the knowledge acquisition bottleneck. It is difficult for expert system developers to extract domain information in an appropriate format for their system. Further problems occur once the expert system has

been implemented. These systems are often slow and unable to handle large volumes of information. In addition, they are very difficult to maintain [1, 54].

Case based reasoning was originally suggested to overcome these problems. The case based reasoning paradigm does not require an explicit domain model, meaning that elicitation is a task of gathering historical experiences or cases [21, 54, 55]. This also means that case based reasoning can handle domains where the problems are not fully understood. The difficulty in building a case based reasoner is usually reduced to identifying significant features that describe a task. Managing large information bases can be handled by developing a case based reasoning system that makes use of database techniques. Acquiring new knowledge occurs whenever new cases are added to the system. In this manner, case bases are easier to update than expert systems.

1.3.2 Case Base Reasoning and Databases

Case based reasoning can be closely intertwined with databases. Large CBR applications can use database techniques to efficiently store their data. In these systems, the case based reasoner acts as an intelligent layer operating above the data base. This layer provides capabilities such as fuzzy search and rich indexing support. Using this layer allows the user to extract relevant information from a database very quickly.

1.4 Application Examples

1.4.1 General Dynamics

The following example illustrates a domain in which case based reasoning produced better results than a typical rule based system. In the late 1980s, a US company, General Dynamics, decided to develop a knowledge based system to assist in building warships. One of the more difficult problems in this area is selecting appropriate mechanical equipment during the ship design. Many of the problems existed from ship to ship and were considered standard. However, there were also non-standard problems, particular to only one warship, which required a longer period of time to resolve.

General Dynamics eventually implemented a rule based system to address this problem. The system solved the standard problems efficiently, but was unable to model the non-standard problems [54]. Each time a non-standard problem was encountered, the system

developer was required to update the existing rules to model this new information. Eventually, the company decided that this maintenance was too expensive. They switched to a case based reasoning system three years after deploying the initial version of the expert system. General Dynamics estimates that the case based reasoning system solved more than 20 000 non-standard problems and saved the company over \$200,000 in the first year alone [34].

1.4.2 Lockheed

One of the first commercial applications of case based reasoning technology was implemented by Lockheed in Palo Alto [18]. Lockheed builds aircrafts which contain many elements that are made up from composite materials. These materials require curing in large autoclaves. Each one of these materials has different heating characteristics and must be cured properly. A mistake in the autoclave means that the material must be discarded. To further complicate the problem, many different materials are placed in a single autoclave and these materials may interact to affect the autoclave's heating and cooling characteristics [19].

The operators of the autoclave used to solve the problem manually using a set of previously successful layouts. The operators searched through the layouts to find the layout which most closely matched their current distribution of materials. Then the operators adapted the layout to fit their current problem. As this scenario describes the case based reasoning paradigm, it is hardly surprising that Lockheed decided to implement a case based reasoning system to assist the operators.

The system is called CLAVIER. In CLAVIER, each layout is a case, consisting of information describing the parts or materials and their relative positions on a table, the relative positions of the tables in the autoclave and statistics such as start, finish times and temperature. The system was designed in 1990 and has been in use since then. It now claims a 90% success rate in layout retrieval and has grown from 20 layouts to over 150 [19].

1.4.3 Other Applications

Table 1.2 illustrates the type of applications for which companies are using case based reasoning.

Application Type	Companies
Customer Service	Black and Decker London Electric National Westminster Bank
Help Desk	AT and T Broderbund Compaq
Quality Control	Volkswagon NEC Nestle
Information Management	Mitsubishi Electronic Corp. Swiss Bank Dun and Bradstreet
Fault Diagnosis	General Dynamics Matra Space Corporation DEC

Table 1.2: What are Companies Using CBR For? [adapted from [54]]

1.5 Discussion

This chapter was designed to provide the reader with an introduction to the case based reasoning paradigm. Several different avenues of current research within the field of case based reasoning were discussed. The chapter concluded with some examples of the practical applications of case based reasoning. In general, these systems can be designed more cheaply than expert systems and have a higher success rate in representing non-standard problems. Many corporations are turning to case based reasoning to automate various parts of their daily tasks, from customer service to quality control, to information management.

Chapter 2

Case Based Management

A pervasive, yet relatively ignored, problem inherent in using the case based reasoning approach is that of case base maintenance. A case base is usually constructed over a long period of time, during which cases are entered, by different case authors, at different times. As well, a case base may be the result of amalgamating several different smaller case bases, or it may be the result of “scanning in” raw material from large quantities of literature. Similarly, a company’s use for any given case base may change over 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. As the case base grows, errors within it 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 or not 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 the ability for a system to identify and suggest solutions to “inconsistent cases.” A case consists of a description of a target problem and its corresponding solution. If the case description and solution contain errors, it may lead to a 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 are 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 [26, 25, 29]. I conjecture that the same amount of effort will be expended by organizations exploiting case base reasoning systems.

My approach to addressing the problems described above is to design a maintenance agent which can sift through volumes of case data and alert the case base manager of situations where redundant or inconsistent cases occur. I will use information retrieval techniques to normalize the case base allowing for easier comparison between cases. Facilitating case comparisons will facilitate redundancy detection. If the case representation is normalized, redundancy can almost be reduced to simple string matching. A further advantage of using information retrieval techniques is that the background knowledge can also be normalized. Once this occurs, inconsistency detection can be achieved through string manipulation. Background knowledge is added to the case based reasoner in the form of string based rules.

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 importance within the case base will decline.

The agent design is guided by the principle that case maintenance itself should incur as little knowledge acquisition effort as possible. The approach outlined in this thesis is a continuation of the ideas we published in [37, 38] and it has been implemented as part of the CaseAdvisor™ system developed by the Case Based Reasoning Group at SFU. The application is called Case Maintainer.

2.1 Previous Approaches

Although research in the field of case based reasoning has repeatedly stated that there is a need for case base management [18, 22, 33], very few researchers have addressed the problem. By management, these researchers generally mean that the retrieval efficiency of a case based reasoner should not degrade over time, errors or inconsistencies should be detected, and redundancy should be eliminated or at least avoided. The previous research in this area has addressed different aspects of this problem, but no one has addressed the management problem as a whole. Most of the approaches outlined in this chapter were motivated through need.

2.1.1 Case Base Learning Approaches

David Aha, [2], presents several case based learning (CBL) algorithms which are tolerant of noise and irrelevant features. Case based learning algorithms are case based reasoning systems that focus on the topic of learning. Essentially, CBLs use a number of training cases as input and, in turn, output a *concept description*. The concept description is always composed of the case base, but may contain additional information such as attribute or feature weights. These weights indicate which attributes are considered important within the case based reasoning domain. The concept description can be used to predict attribute values in future cases, thereby detecting anomalies and filling in missing information.

Typically CBL algorithms emphasize similarity based retrieval and de-emphasize rich indexing schemes, which distinguishes them from CBR systems. These algorithms assume that cases are described using attribute value pairs. The attributes are divided into *predictor* and *goal* attributes. Predictor attributes are those which can be used to predict values for other attributes which are typically called goal attributes. For example, in the case described in Table 1.1, the attribute Price may be a predictor attribute and the attribute Engine Size the goal attribute. The CBL algorithm may learn that as the price of the car increases, so does the size of the engine. If an incoming case has a high value for the attribute Price, the algorithm will *predict* a high value for the goal attribute Engine Size. For the training cases, CBL algorithms will typically use the similarity function to calculate an initial similarity value between cases and then manually change this value to provide the best training set.

The simplest CBL algorithm, CBL1, predicts that the value for a given case's goal attribute is the most frequent goal value among its k most similar stored cases. Similarity

in this algorithm is measured as follows¹:

$$\text{Similarity}(C_1, C_2, P) = \frac{1}{\sqrt{\sum_{i \in P} \text{Feature dissimilarity}(C_1, C_2)}}$$

where P is the set of predictor features and

where $\text{Feature dissimilarity}(C_1, C_2) =$,

$(C_1(i) - C_2(i))^2$, if feature i's values are numeric

0, if $C_1(i) = C_2(i)$

1, otherwise

This defines similarity to be the inverse of the Euclidean distance for numeric attributes and a equal, not equal relationship for symbolic attributes. This algorithm uses a k nearest neighbour prediction function. Aha, [2], claims that the performance of a CBL algorithm is independent of its similarity function. This is perhaps because so much manual manipulation is required to achieve the best results.

Additional Learning Approaches

Much of the previous literature in case based learning relies on prediction based on specific attribute values [44, 45, 46]. The general approach to “fixing” noisy cases is to find all partial matches within the case base and extrapolate missing information or modify anomalies in the case. The matching is done using the case based retrieval mechanism. Once all of the similar cases have been located, they are analyzed to determine the most likely value for the missing information.

This extrapolation is usually done through statistical measures. If a set of values appear together with great regularity and a subset of these values is found in a case with missing information, the rest of the values in the set are applied to the case. Another possibility is to simply use the value in the most similar case. These methods are very similar to how missing information is extracted in databases [49]. The distribution of values for the current attribute is considered. Often, it is simply the modal, the average, or even the mean value that is selected. If the attribute is symbolic, it is typically the modal value that is used to fill in the missing information.

¹These definitions are from [2].

This approach is very effective in rigidly structured domains, such as a domain representing genes [44]. In the case based reasoner described in [44], DNA strands are represented as sequences of numbers. Genes or proteins are subsequences of these strands. Therefore, if one number is changed in the sequence, its subsequences may represent completely different genes. The focus is on extrapolating missing values and identifying outliers. All possible matches are found and only those strands which match over half of the proteins in the current strand are examined. This algorithm makes use of background knowledge that describes proteins and generates the most likely and consistent value from the similar cases.

Predicting missing values is not as easy in a case that has no explicit structure. In unstructured cases it is unclear what the attributes in a case may be. If there is no standard set of attributes, then these algorithms will be unable to learn how to predict attribute values. This is a serious limitation if developing a case based management system to work across different types of case bases.

An additional disadvantage in using these algorithms is that they will not necessarily perform well if feature importance is context sensitive [2, 4]. Essentially, context sensitive means that the importance of a particular feature may depend on the other attribute value pairs that appear within a case. For example, consider political decisions. Let person A typically vote Liberal, but always vote pro education. In the case where the Liberals attempt to raise tuition fees, the attribute value that person A always votes pro education will be of significant importance; it is likely that person A will vote against the Liberals. However, if the Liberals and all other parties have similar educational agendas, then the fact that Person A always votes pro education is irrelevant. Given the information available, the obvious prediction is that person A would vote Liberal. This type of behaviour is modelled by association rules in relational data bases [17]. However, relational tables do have an explicit structure and typically model very narrow attribute value pairs. The mining of association rules also requires background knowledge in the form of concept hierarchies which model each attribute at different levels of generality [17].

Context sensitive case weights have not yet been derived to handle this problem [56]. Features must be uniformly important across all cases for these approaches to work effectively as each attribute is given static consideration when extrapolating missing values.

Essentially, the research described above focuses on learning and organization at the *feature* level. Given a missing or odd attribute value, these algorithms predict the missing value based on the information contained in the case base. These approaches are limited to

missing or odd attribute values. A case base management system must be responsible for the overall competence of a case based reasoner. This requires the detection of redundant and inconsistent information at a *case* level rather than a *feature* level. That is, I compare entire cases and determine whether an entire case is redundant or inconsistent.

2.1.2 Knowledge Discovery Research

The knowledge discovery process was developed to discover interesting information patterns within large databases [11]. The first three steps of the process are as follows: understanding the domain the database represents, identifying the interesting attributes, and preprocessing the data [11]. This last step is necessary to improve the results of the knowledge discovery process. Missing information and outliers are detected and repaired, to reduce their effect on the information patterns generated to represent the data. The techniques used to preprocess the data, data cleaning algorithms, are very similar to the case base learning algorithms discussed above.

Data cleaning algorithms require training data to generate profiles of the information within the database. Essentially, they generate *information patterns*, which describe the data at a higher or more general level. As new data is collected, it is compared to these information patterns [16]. Outliers can then be identified through simple comparisons to the known or probable range of values for a particular attribute. If a case is missing a field, the information patterns are used to generate the most likely match by examining the most similar pattern(s).

As with case base learning algorithms, training data is needed to generate descriptions or information patterns, which are then used to predict values in incoming data. Essentially, the reasons data cleaning approaches are not suitable for case base management mirror those reasons for which case base learning algorithms are disqualified. Although data cleaning techniques can handle context sensitive data, they rely on structured data and focus only on consistency management. A case base management agent must be able to handle unstructured data and perform redundancy detection.

2.1.3 Expert System Research

Researchers in the field of expert systems have also conducted extensive research in the area of system maintenance. However, most current approaches focus on implementation

representations such as rules or frames, which typically require formal representations to be applicable [20]. This is often impossible in case base systems as case based reasoning is most effective when applied to poorly structured domains or domains where expertise is explained by example [54].

Other work in this area has concentrated on developing a framework for the detection of possible anomalies or redundancy [35]. The detection of these anomalies requires knowledge regarding the expected behaviour of the knowledge base. When the expected and actual behaviour differ, an anomaly is detected. These anomalies are then referred to as **potential errors**, as they may actually be correct. Again this system hinges on prediction. Prediction may be impossible in case based reasoning where no explicit domain model is required. Also, loosely structured cases will again affect the competence of these types of algorithms.

A further limitation is that this framework can only be applied to rule-based systems [53]. While research is continuing in this area to relax that constraint, in order to be able to apply the framework to a case based reasoning system, one would need to formally specify the cases. This contradicts the largest advantage of case based reasoning; no domain model is required to build a case based reasoning system. In a poorly structured domain, formally specifying the cases could lead to missing information or very large cases. Missing information could occur as some cases simply will not break down into well defined attribute value pairs. Also, if the cases can be partitioned into well defined attribute value pairs, but were not previously, this may mean that each case shares a very small number of attributes. This would imply that a very large number of attributes would be required to give each case a standard representation.

This research has generated definitions of redundancy and subsumption that can be adapted for a case base reasoning system [7, 53]. Algorithms have been established to identify these types of interdependencies within rules. Although not a perfect match, these algorithms provide a first step toward solving the case base management problem.

2.1.4 Case Base Maintenance Approaches

Most of the previous research in the area of case base maintenance is concerned with *optimization*. Due to the large size of some case bases, it is necessary to delete cases as time goes by or retrieval stages become increasingly expensive [48]. This issue is called the *swamping problem*. Remember that case based reasoning is an incremental strategy. Cases are added to the case base, but they are typically not removed. The strategy of deciding which cases

to delete is similar to the question of management. Some researchers advocate a random deletion policy [28]. This is a very simple, inexpensive policy and is completely domain independent. Simply randomly select and delete a case from the case base. A slightly more complicated approach is to calculate the frequency with which each case is retrieved and delete those that are not frequently accessed [30]. Ironically, this policy degrades the competence of the case base more than the random deletion policy [48]. The problem with both of these approaches is that “important” cases can be deleted by mistake. In other words, a case that is necessary to answer a query or set of queries may be deleted from the system. Also, cases represent experience and should be carefully reviewed before being removed from the system.

To overcome this problem, Smyth et. al [48], suggest a competence preserving deletion approach. The premise of this approach is that each case in the case 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 be used to provide solutions for each current problem.

Smyth et. al provide formulas for these two concepts²:

Definition 1: Coverage

Given a case base $C = \{c_1, \dots, c_n\}, \forall c \in C$

$$Coverage(c) = \{c' \in C : Adaptable(c, c')\}$$

where $Adaptable(c, c')$ means that c can be adapted from c' within a given cost limit.

Definition 2: Reachability

Given a case base $C = \{c_1, \dots, c_n\}, \forall c \in C$

$$Reachability(c) = \{c' \in C : Adaptable(c', c)\}$$

The obvious problem with these definitions is deciding when one case can be adapted to fit another case. Smyth et. al provide no details as to how the adaptability function is computed.

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

²Definition 1 and 2 are from [48].

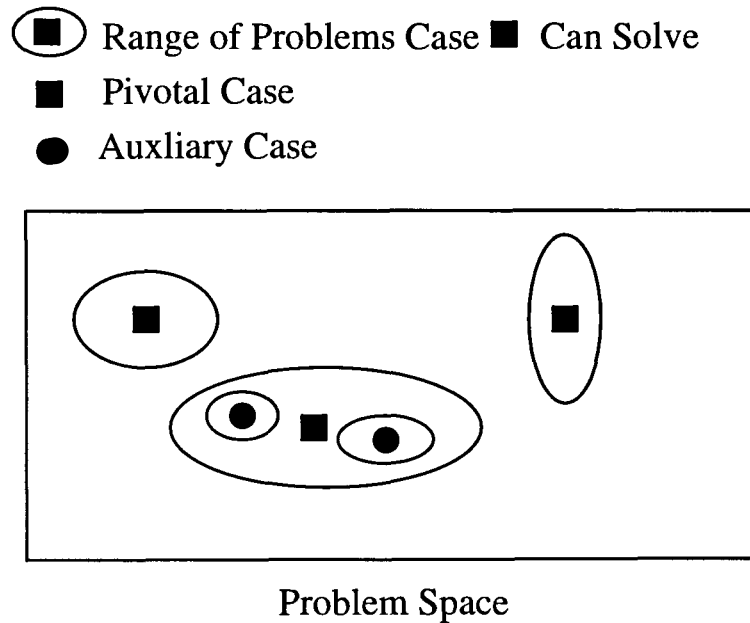


Figure 2.1: Illustrating Pivotal and Auxliary Cases [adapted from Figure 1(a) in [48]]

two extremes are the *spanning* cases which link together areas covered by other cases, and *support* cases which exist in groups to support an idea. The deletion algorithm then deletes cases in the order of their classifications : auxiliary, support, spanning and then pivotal cases [48]. A limitation is that the case classification is dependent on the concepts of coverage and reachability. For example, the definitions provided for pivotal and auxiliary cases read as follows³:

Definition 3: Pivotal Case

$Pivot(c)$ if and only if $Reachable(c) - \{c\} = \{\}$

Definition 4: Auxiliary Case

$Auxiliary(c)$ if and only if $\exists c' \in Reachable(c) - \{c\} :$

$Coverage(c) \subset Coverage(c')$

Figure 2.1 illustrates the differences between pivotal and auxiliary cases.

Similarly, the definitions for spanning and support cases also rely on the concepts of coverage and reachability. Thus, everything hinges on the definition of the function Adaptable.

³These formulae were extracted from [48].

As stated earlier in this thesis, adapting cases requires extensive knowledge engineering and may not be possible for all case bases. Smyth et. al concede that it is an expensive strategy, but point out that it is a one time cost and should be able to be absorbed by the system. In their evaluation of their algorithm, they restrict the size of the case base and the size of the problem space and manually identify the category in which each case falls. However, the knowledge required to adapt the cases may simply not exist. Also, in a large case base, this manual adaptation will be very time consuming. Therefore, applying this approach to large and dynamic case bases may be an impossible endeavour.

The approach by Smyth et al. is motivated by the need to delete cases in order to maintain the case base at a reasonable size. The solution it offers can be considered to be one at a meta-level. In other words, no mention was made about *how* auxiliary cases are identified, and *what* will be done once they are identified. Pivotal cases may be “important” or they may simply contain anomalies that distinguish them from the rest of the case base. This approach, therefore, may harbour inconsistent cases more rigorously than other types of cases. A final problem with all of the deletion approaches is that case based reasoning systems rely on cases to successfully resolve problems. Cases represent accumulated experience that the users have acquired over time and as such can be considered assets to a company or organization. Deleting a case from a case base without notifying the user may reduce the set of queries that the application can satisfy. In this case the competence of the case based reasoner has been degraded rather than improved.

On a final note, further motivation for case base management can be found in Smyth’s paper. Empirical testing was completed on a relatively small case base consisting of only 50 cases. The researchers concluded that five (5) cases were auxiliary cases [48]. Even in this small case base, 10% of the cases were identified as redundant. This suggests a need for redundancy detection in case based reasoning systems.

2.1.5 Information Retrieval Applied to Case Based Reasoning

Information retrieval approaches have long been suggested as viable methods for pattern matching. It is stated in [10, 58] that information may be compressed by searching for and removing redundant information, and that information retrieval techniques can facilitate this search. This provides a basis for suggesting that information retrieval techniques can be used to identify redundancy within a case base.

Information retrieval approaches have previously been applied to case based reasoning

systems in order to facilitate retrieval. These techniques have been applied with great success to the law domain [15, 40]. Law cases can be partitioned into four different components : facts, concepts, cases, and legislation. Applying information retrieval techniques to this domain has significantly improved relevant retrieval[15, 40]. Each case is given a summary through the use of information retrieval techniques. In [15], this summary consists of all of the concepts, cases, statutes and facts that were extracted from the case. Following this information is a list of the important paragraphs which are identified according to citation patterns (i.e what type and how many citations appear in the paragraph), paragraph position in the document and the paragraph length. Although this approach is specific to the law domain, the idea of producing normalized summaries for each case is very attractive. Normalized representation of cases can facilitate both comparisons between cases and between cases and background knowledge.

Information retrieval techniques have successfully been combined with case base reasoning systems in the past. They have enhanced retrieval in several different systems [10, 15, 40]. However, they have not been used to implement redundancy or inconsistency checking. They have only been applied to case bases in the area of case retrieval. My approach is to use these techniques to not only enhance retrieval, but to enhance the overall competence of the case base. Information retrieval techniques facilitate retrieval by normalizing the cases, thus allowing an application to draw easier comparisons between them. Both redundancy and inconsistency detection require some method of comparing knowledge. Therefore, by facilitating comparison, one can facilitate management.

2.1.6 Agents and Indirect Managing

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. I suggest that an agent should be designed to implement a complementary style of interaction, called indirect management [27]. According to *Webster's Dictionary*, an agent is "a means or an instrument by which a guiding intelligence can reach a result"⁴. Recent research in the area of agent development suggests a parallel definition where an agent is an application which acts in place of another application or the user with permission [14].

Indirect management means that the user and the agent cooperate in an attempt to

⁴Taken from the online version of *Webster's Dictionary* available at SFU

reduce the quantity of information exposed to the user. My vision is to design an agent that will automate or semi-automate some of the actions for which the user is currently responsible. These tasks include key word extraction from the cases, redundancy identification and inconsistency detection. Instead of the user being responsible for the entirety of each of these tasks, the agent will perform the tasks and then submit the solution for the user's approval. The idea of using agents to delegate certain tasks was first introduced in [31]. It is now a widely accepted idea across many areas of software development, and applying this idea to case based reasoning seems viable.

Case based reasoning is an interactive problem solving paradigm. The user is involved in almost every step of the case base process [21, 55]. The user initiates communication with the case based reasoner by typing in a query. The application then uses the query to locate the most similar cases. In some case base reasoners, a list of questions may also be returned. At this point, control is returned to the user. The user may or may not choose to answer questions, to select the case with the most favourable weighting, or to select the case with the least favourable weighting. At each step in the process, control is returned to the user. The application merely weeds out what it considers to be extraneous information.

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.

2.2 Outline

In this thesis, a theoretical and empirical framework for the design of an agent to perform case base management is presented. The algorithms that are used to implement this agent are discussed along with preliminary empirical results demonstrating the efficiency of the agent and its ability to semi-automate redundancy and inconsistency detection.

Chapter 3 introduces the need for case management, based on inherent problems encountered while using the case based reasoning paradigm. The chapter outlines the problems

that can arise without a management agent. Following that, my criteria for evaluating the competence of a case based reasoner are outlined. A management agent must not violate these criteria. The agent architecture is presented at the end of the chapter.

Chapter 4-6 discusses the design of the agent. Each module in the agent is discussed. Chapter 4 explains the motivation for using the information retrieval techniques and the basic strategies employed. Chapter 5 outlines the algorithms used for redundancy detection and Chapter 6 illustrates the techniques developed to detect inconsistency. These three chapters are designed to allow the reader to gain an understanding of the strategies the agent employs for case base maintenance.

Empirical results are presented in Chapter 7 to evaluate the performance of Case Maintainer. The preliminary results indicate that Case Maintainer can effectively semi-automate the management process. Factors affecting performance are identified and a discussion regarding the control of these factors to yield higher efficiency is included.

Chapter 8 presents an Annotated Example illustrating the system at a global level. It is intended to provide the reader with an idea of the scope of this thesis. The example uses a real life domain which is presently in use at Roger's Cablevision. The chapter includes actual screen layouts from the Case Maintainer captured during processing this domain. The entire process is presented.

Finally a summary of the thesis is given in Chapter 9, along with a discussion of possible future avenues of research.

Chapter 3

The Case for Case Base Maintenance

3.1 Two Types of Cases

The majority of research on case based reasoning has concentrated on cases with well defined attributes. These cases have a relational structure, where each attribute is more or less a field in a relational database. For example, in an auto-repair domain, a case base may have the structure of the case presented in Table 3.1.

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 attributes to use to represent each case. Then a decision has to be made regarding the range of legal values each attribute may have. The process of authoring knowledge in this attribute value format requires extensive maintenance when a new attribute is discovered and inserted, or when an existing attribute becomes irrelevant. In addition, unstructured documents rarely break down into obvious attribute value pairs. By reducing each case to this structure, the meaning or the purpose of the case can be lost in the translation.

In industrial practice, a majority of the case bases come directly from either unstructured text documents or end-users' verbal description. These cases may have generic attributes such as *case problem description* and *case solution*, but each of these attributes probably will not be further partitioned down to a relational level.

Case based reasoning was adopted as a paradigm to solve a range of problems that

Attributes	Make	Model	Model Year	Engine Type	Mileage	Case Problem Desc.	Case Solution
Case 1	Mazda	626	1988	20l EFI	12 498	Engine stalling	Clean fuel injector
Case 2	Honda	424	1987	2.8L	67 183	No good mileage	Replace gas pump

Table 3.1: Example cases for automobile diagnosis and repair.

could not be handled by rule based systems[21, 54]. It is most effective when applied to poorly structured domains or domains where an expert teaches by example. Case based reasoning can also be applied to poorly understood domains providing some type of legacy data source exists. Highly structured domains can generally be effectively modelled by a rule based system, but rule based systems are not flexible enough to handle poorly structured or understood domains [54]. Therefore, if a case based reasoning system is used to model an appropriate (poorly understood or structured) domain, it may be very difficult to generate cases in a relational format.

Consider the following example of a free form case used in a printer repair domain. This case actually exists in a case base used to diagnose printer failures in a Hewlett Packard laser printer.

CASE PROBLEM DESCRIPTION : Paper continues jamming laser printer due to dirty and/or sticky internals.

CASE 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 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. Much research has been conducted on relational databases to learn how to better specify and apply integrity constraints [49]. Case bases containing structured cases can use this research to maintain consistency within their case base.

Unstructured cases are more problematic. Often the cases cannot be reduced to a set of attribute value pairs, so even range checking can be a complex problem. Reducing the above example of a printer repair case to a set of attribute value pairs would require extensive knowledge engineering and may not truly represent the case's meaning. Further problems occur between the cases themselves. Differences in vocabulary, punctuation and the level of detail used to describe a case can cause difficulties when evaluating a case base.

Obviously, a case base management agent must be able to account for unstructured cases as well as structured cases.

3.1.1 The Redundant Case Problem

In a large legacy case base, redundancy identification requires the ability to detect two equal cases, if one case subsumes another, or if two cases can be merged. At the rapid rate that industry is changing, it is possible that two previously distinct case bases will need to be merged. In this case, it is critical to develop a mechanism that can collapse the redundant cases into a representative case for the class of problems that the cases can solve. This mechanism must have the ability to explain why the cases were identified as redundant, so that the user can make an informed decision to resolve the problem.

Current redundancy testing involves submitting the new case as a query to the case base reasoner. If a reasonable solution is returned, the case is not entered. However, in practice, cases are often entered by a module separate from the problem resolution module. Case authors enter a set of cases at the same time and then test the system. The iterative type of testing described above may not be feasible for a large case base. A further problem is that redundancy is not always obvious. Case authors may not be domain experts, and thus, not familiar with the domain jargon. In addition, the range of problems that a large case base reasoner can solve may be wide. Manual, iterative testing for redundancy may be very time consuming. Finally, companies may already have their data available in a different format, where redundancy may not be obvious. Some companies already have their data collated in decision trees, where there may be great overlap. Redundancy within free form

<p>Case 1 CASE PROBLEM DESCRIPTION: Envelopes jam laser printer due to glue. CASE SOLUTION: Normal envelopes and laser printers do not get along together. Problems include poor glue heat tolerance.</p>
<p>Case 2 CASE 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 gummed labels.</p>

Table 3.2: Example of redundant cases in the printer repair domain

text is not always obvious. There are differences in vocabulary, depth of detail, and even punctuation. Therefore, a mechanism to detect redundancy and offer an explanation for that identification is critical.

An example of redundancy in the printer-repair domain is displayed in Table 3.2. 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.

Subsumption is identified by detecting sufficient conditions. For example, if case A requires case problem description (x, y, z) to reach case solution (u) while case B requires only case problem description (x) to provide the same solution, (u) , then B subsumes A. Case problem description (x) is a sufficient condition to offer case solution (u) .

The iterative nature of case based reasoning means that the size of the case base is always increasing¹; storage space and retrieval efficiency will eventually become critical issues. Therefore, a case base management agent should have the ability to detect whether two cases can be merged. Two cases may be candidates for merging if they are similar enough to share common attribute values, but also have a small number of critical differences. Merging is suggested by the application if two cases share some percentage of common key words and a common field, yet have at least one significant difference. Obviously, if the two cases do not share one significant difference, the application will suggest that the two cases are equivalent and one should be deleted.

The algorithms for equivalence testing, subsumption and merging will be discussed in

¹See Figure 1.1

more detail in the next chapter.

3.1.2 The Inconsistent Case Problem

As a case base grows larger, the number of inconsistent cases will inevitably increase as well. Databases often contain inconsistent or “harmful” data [16, 48]. I conjecture that case bases can be worse, as relational databases in particular often have more structure than a case consisting of free form text.

A case can be inconsistent in a number of different ways:

1. A case can be inconsistent with the background knowledge in an application domain. For example, due to a typing error, 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 it contradict each other. For example, a case from a printer-repair domain may have an inconsistent solution requiring the user to both repair and replace the printer.
3. A case can be inconsistent with another case because combinations of fields may contradict each other. For example, in the medical domain, one case may suggest that 0.3cc of morphine as treatment for a broken arm, 0.7cc of morphine for a broken leg and 1.0cc of morphine for a broken neck. However, a patient who has a broken arm, leg and neck should not receive 2.0cc of morphine as treatment.

The first two examples described above are instances of intra-case inconsistency. The case is either self contradictory or it conflicts with known background knowledge. The third example is an instance of inter case inconsistency. Each case by itself is correct. However, if the cases are combined in an additive relationship, the result can be deadly. Inter-case inconsistency can occur when combining two or more cases and is more difficult to detect than intra-case inconsistency. Most of the work presented in this thesis focuses on intra-case inconsistency, but inter case inconsistency will also be discussed.

Inconsistency can further be partitioned into *soft* and *hard* contradictions. The first medical case base example above presents an instance of a soft constraint violation. A soft constraint violation could occur when a uncommonly occurring attribute value is found in

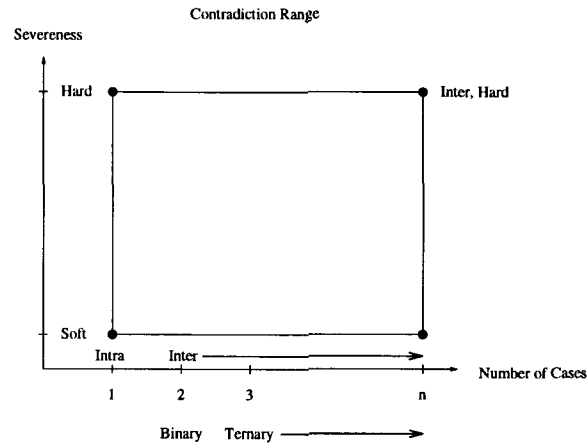


Figure 3.1: Dimensions of Inconsistency Within a Case Base

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. For example, a case requiring a user to perform actions (go forward) and (go backward) simultaneously, contains a hard constraint violation.

Inconsistency can arise in a variety of ways. A self cleaning agent must be able to identify different types of violations.

3.1.3 The Dynamic Reorganization Problem

A large case base requires constant reorganization. For example, in the printer-repair domain, a case base may initially contain only the cases relevant to solving an institution's line printer problems. With the acquisition of new laser printers, new cases need to be designed and entered, and the requests for accessing the line printer cases may dramatically decrease. Cases are very rarely evenly used in a case base system. Cases that are consistently and frequently accessed should be quickly available to the user. An intelligent case base management agent should be able to detect these cases and offer solutions to respond to the user's demands quickly.

The focus of this thesis is on redundancy and inconsistency detection. As a result, I do not focus on the dynamic reorganization issue in this thesis.

3.2 Criteria for a Competent Case Base Maintenance Agent

Given the above listing of management problems, a comprehensive case base agent must be able to semi-automate the solution. Since there are many different ways to design such an agent, there is a need to first define the criteria required for a “competent” case base management agent.

3.2.1 Consistency

In addition to coverage and reachability as defined by [48], I define a third criterion called *consistency*. Consistency can be defined in many different ways. A single case may be consistent with the background knowledge, if it “makes sense” in the context of the knowledge. Similarly, two cases must be consistent with each other when both are used in a composite solution. The former is called intra-case consistency, while the latter is called inter case consistency. In an automobile-repair domain, a case is inconsistent with the background knowledge if an engine type is not available given the particular model of a car. In the same domain, an engine-diagnosis case is inconsistent with an exhaust-diagnosis case if they result in incorrect explanations for the problem of a car. In the scope of this thesis, I primarily focus on intra-case consistency.

Simply stated, the competence of a case base can be measured by how often and how quickly the case that is retrieved is the case in the case base that answers the query most effectively.

3.2.2 Qualitative Competence Criteria

My qualitative criteria for a competent case base maintenance agent is as follows:

- It will maintain a high degree of intra-case consistency for the entire case base. A case that conflicts with background knowledge or contradicts itself may not provide a reasonable solution to the user.
- It will maintain a high degree of coverage for the case base. The case base reasoner must still be able to answer the set of queries that it was designed to answer. Coverage is related to retrieval accuracy.
- It will maintain a low degree of reachability for the case base. The higher the reachability of a case base, the higher the number of auxiliary or redundant cases. These

cases can interfere with retrieval efficiency.

3.3 Agent Design

My approach to solving the maintenance problem for very large legacy case bases is to integrate an agent within a case based reasoning system. The agent is used to automate time consuming tasks in case base maintenance as discussed in Chapter 2. A block diagram of the agent design for case authoring is shown in Figure 3.2.

3.3.1 Case Authoring

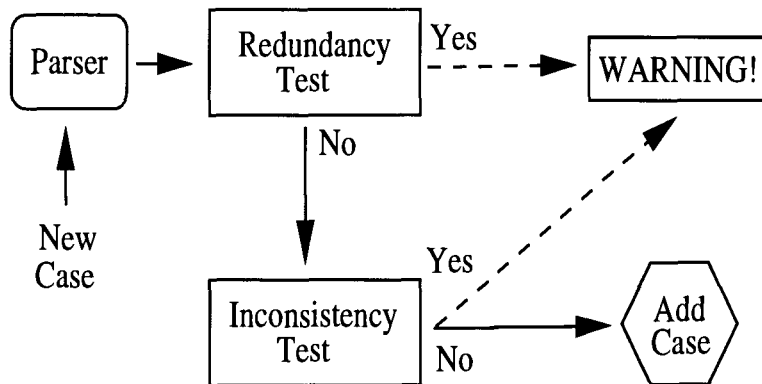


Figure 3.2: Block Diagram of Agent Architecture for Case Authoring

In order to minimize the knowledge acquisition bottleneck, the agent must allow unstructured cases to be processed as well as structured 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 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 guidelines and the cases and determine possible consistency problems. The inconsistency detection module is independent of the redundancy module; the redundancy module can fail to work and the inconsistency module can still be applied. However, as redundancy is more common than inconsistency, I have chosen to search for inconsistency only after a case has successfully passed through the redundancy detection module. Again, the inconsistency detection module relies on the information retrieval module.

For those cases with sufficient evidence of inconsistent content, an alert will be generated for the user to take action. This alert will identify the case involved, the guideline involved and where the possible inconsistency occurs within the case.

3.3.2 Problem Resolution

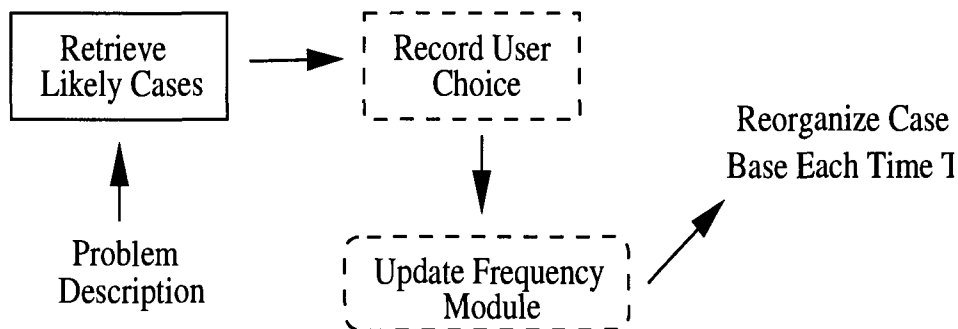


Figure 3.3: Block Diagram of Agent Architecture for Problem Resolution

During problem resolution, a case base organization module will detect the frequency of usage for the cases and the last date accessed and use this information to determine a hierarchy of cases. The levels of the hierarchy are organized such that the most accessed and useful cases are presented first. In the event that the usage of a particular case increases, the module will promote the case to a higher level of the hierarchy. Likewise, if a case's usage goes down it will be demoted. The module extracts the last date the case was accessed so that cases with high frequency will be demoted quickly if they have not been accessed for some time. This is because cases that remain in a case base longer will typically have a high frequency, but may become obsolete. Continued usage of a case ensures that it is still relevant and needed. Frequency statistics do not provide sufficient reason to delete a

case altogether. Studies have proven that this approach degrades the competence of a case based reasoning system quickly [30]. Therefore, only with the user's approval will a case finally be deleted from the entire case base.

3.4 Discussion

This chapter explained the motivation for a case base management agent. Unstructured cases are typically used in case based reasoning systems, yet most research has concentrated on structured cases[2, 53]. Unstructured cases can cause more management problems than structured cases.

This chapter also provides definitions of the redundancy and inconsistency problems within a case base. Redundancy is further partitioned into equivalence and subsumption and both ideas are discussed. Limited storage space may require that the application reduce the number of cases in the case base. Strategies for identifying cases that are candidates for merging are introduced. Methods of providing solution suggestions to solve redundancy are also discussed.

The redundancy algorithm requires a method of comparing cases which, in turn, suggests a need for a method for normalizing cases. Further argument for normalizing cases occurs when considering inconsistency detection. Background knowledge must be acquired in some format whereby it can be compared to a case.

In addition, the iterative nature of case based reasoning suggests that a case base is continually growing. A management agent should be able to dynamically identify cases that have been used frequently and recently. Cases that have not been accessed for a long period of time or those that are used infrequently will not be stored in working memory.

Criteria is identified that must be satisfied to maintain a competent case based reasoner. A case base management agent must not violate these criteria. Following this, the design of the agent is discussed at a general level, providing the reader with an overview of the system. The following chapters provide more detail as to how each module within the agent reaches its desired goal.

Chapter 4

The Information Retrieval Module

4.1 Why Information Retrieval?

I use information retrieval techniques to normalize the cases. These techniques have successfully been applied to case bases in the legal domain [15] and to large databases [41].

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.

In a case based reasoning system, the case base can be considered a set of documents and an incoming case can be considered a query. The similarity mechanism converts both the incoming case and the set of documents (the case base) into a standard format. This mapping allows the system to compare the new case to the case base which can facilitate redundancy detection. It is called an indexing language. See Figure 4.1 for a graphical representation of this system as applied to case based reasoning.

The information retrieval architecture can also be applied to inconsistency detection. In this case, the background knowledge can be thought of as the query and the case base as the set of documents. Using the indexing language, the background knowledge can now be compared to the case base to detect inconsistency.

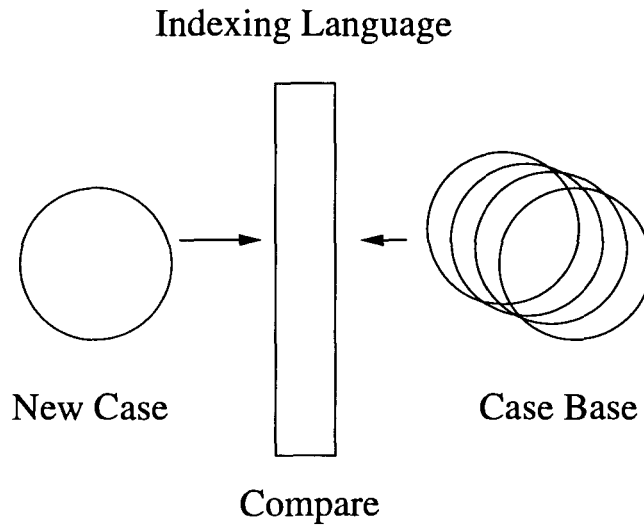


Figure 4.1: My Information Retrieval Agent Design For Case Based Reasoning [adapted from [41]]

4.2 General Algorithm

The specific steps in the information retrieval algorithm used by Case Maintainer are:

1. Remove the stop words.
2. Collapse words using a domain thesaurus.
3. Remove the suffixes and prefixes from each term.
4. Build an inverted index.
5. Build a key word index.
6. Build a key phrase index.

The output of this algorithm is a internal, normalized array of cases. If the redundancy and inconsistency detection modules have not been activated by the user, the application builds a case base from this array. As well, three flat files representing the inverted index, the key word index, and the key phrase index are generated. These files are represented in binary form to reduce the storage space required.

4.2.1 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 between 40% - 60% of the words within a document [41]. The application uses a general list of stop words generated for the English language used by the SMART system designed by Salton [41]. 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.

4.2.2 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.

4.2.3 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 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.

4.2.4 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.

4.2.5 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 [13, 41]. This application uses the inverse document frequency measure to identify key words [41].

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 which cases the new word appears in 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.

4.2.6 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 inter case cohesion [41]; if one word appears in a case, then the other words have a very high probability of also appearing. The phrases and their

<p>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.</p>
<p>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</p>
<p>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.</p>

Table 4.1: Example of Information Retrieval Techniques Applied to Incoming Case

corresponding weight 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.

4.3 Example

An example of the information retrieval process applied to one case in the printer-repair domain is shown in Table 4.1.

4.4 Design Rationale

Information retrieval techniques have been extensively studied since the 1960s. As a result, many systems are available for adaptation. I chose to use the stop word and stem removal algorithms discussed in [13]. The strategies for extracting key words and phrases were adapted from [41]. I then built the inverted index using these algorithms.

The literature in Information Retrieval offers many possibilities concerning what information should be stored in the inverted index [13, 41]. I have chosen to store a minimal amount of information. Additional information that could be stored in the indices includes adjacency information. This information could facilitate both redundancy and inconsistency detection, but would increase the size of the inverted index. As the focus of this thesis is on large case bases, this additional storage cost may be insupportable.

The key word and key phrase index are structured differently than the inverted index. The list of documents in which the term or phrase appear is also retained in the index. This allows the application to cluster cases containing a high percentage or number of key words. This feature will allow for easier redundancy detection.

The critical path in a case base reasoning system is considered to be Problem Resolution. The time required to author cases, while important, is not as critical as having real time assistance once the case base is operational. The information retrieval, redundancy and inconsistency modules have limited impact on the Problem Resolution module.

As the redundancy and inconsistency detection modules both require input from the information retrieval module, it is designed to always return a result. Statistical information is not as interesting or effective if the case base size is too small. As a result, with small case bases, < 20 cases, the information retrieval module extracts key words and phrases locally to each case. That is, the stop words are removed, the remaining terms are stemmed and whatever remains is added to the key word index. Similarly, if the case base contains only disjoint cases, the key words will be extracted in the same fashion. Thus, the key word index will always contain information. The degree of relevance of this information will vary, but the redundancy and inconsistency detection modules will still be able to function.

4.5 Discussion

Information retrieval techniques facilitate the comparison of cases. These techniques consider all cases before extracting key words and phrases. Therefore, the key words represent the important words considering all of the data stored in the entire case base rather than just the current case. Many case base systems work on a local basis. Key words are extracted from each case independently, ignoring the information contained in other cases. Cases are “normalized” allowing the similarities or differences between cases to become more pronounced. This normalized representation of each case can be used by retrieval schemes to better address the user’s problem.

The main advantages of using information retrieval techniques are that the cases can now be compared to one another with greater ease and key words can distinguish one case from another. Thus, identifying a connection or similarity between cases is more likely. This increased power of comparison can facilitate both redundancy and inconsistency detection.

Chapter 5

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.

5.1 Equivalence and Pure Subsumption

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 to 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.

In other words, redundancy is identified as follows:

- Let C represent the case base.
- Let K represent the set of key words.

- $A \in C$ and $B \in C$.
- $k(X) = \{k_i | i = 1..n \wedge k_i \in K \wedge X \in C\}$. $k(X)$ is the set of key words that exist in case X.
- Redundancy can then be considered when $|k(A) \cap k(B)| > \theta$. Redundancy may exist when the intersection of the key words from case A and case B exceeds some threshold θ .
- If the two cases share more than T key words, then perform a string matching algorithm on the two cases. If redundancy seems to still be a consideration, the matching algorithm returns a score greater than 80% and presents both cases to the user.

5.2 Pure Subsumption

Cases can also be redundant because they are subsumed by other cases. Algebraic rules can identify subsumption. The advantage of identifying 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 indicate which is the more powerful case.

Consider : *Subsumption Rule 1:*

Case 1 : case problem description (p_1, p_2) case solution (s_1)

Case 2 : case problem description (p_1, p_2, q_1) case solution (s_1)

Here, q_1, p_1, p_2 and s_1 can either be a key word or a set of words containing a key word. Consider p_1 and p_2 to be premises of the case problem description. In this case, Case 1 subsumes Case 2. The sufficient conditions for case 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. If 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.

Each subset of the problem description and the solution that is considered must contain a key word. If it does not, then prepositional phrases, articles and other extraneous information can be considered sufficient conditions. Not only would this result in an increase in the detection of subsumption, but it would result in the application suggesting that prepositional phrases and articles are sufficient conditions to declare subsumption. The advice then offered to the user would be useless.

Similarly, consider *Subsumption Rule 2*:

Case 1 : case problem description (p_1, p_2) case solution (s_1)

Case 2 : case problem description (p_1, p_2) case solution (s_1, s_2)

As in 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 that I have considered, *Subsumption Guideline 3*. The term guideline is used as this *rule* is not logically entailed, unlike the first two subsumption rules. However, in my experience, this scenario usually does involve two redundant cases.

Case 1 : premises (p_1) solution (s_1, s_2)

Case 2 : premises (p_1, p_2) solution (s_1)

In this instance, the system generates a third case:

System Generated Case : premises (p_1) solution (s_1)

Case 1, Case 2 and the System Generated Case are presented to the user for consideration. The new case generated by the system may subsume both Case 1 and Case 2. Although this guideline does not logically follow from Subsumption Rule 1 and Rule 2, experience has shown that case authors tend to include extraneous information as they are typically not domain experts. Therefore, from Case 1, observe that (p_1) may be a sufficient condition to offer solutions (s_1, s_2) . However, Case 2 stipulates that (s_1) is sufficient to solve both p_1 and p_2 . Presumably, therefore, s_1 should be sufficient to solve p_1 . Of course, there are cases when this rule fails. However, I believe that this scenario is typically an example of redundancy.

Rather than simply deleting the cases identified as subsumed, the application presents these cases to the user. This is because the typical user of the application may not be familiar enough with the domain to delete the case that offers more information. Maybe the extra premise offers valuable information to the novice user that the case that subsumes it does not.

<p>Case 1</p> <p>Case Problem Description: no reception on low band</p> <p>Case Solution: Check there is no splitter on the cable. Fine tune the TV channels if problem continues and then unplug the TV for 30 seconds and plug it back in. If problem still continues, generate a trouble ticket.</p>
<p>Case 2</p> <p>Case Problem Description: no reception on high band</p> <p>Case Solution: Check no splitter on cable. Fine tune TV channels if problem continues. Unplug TV 30 seconds, replug. Problem? Generate trouble ticket.</p>

Table 5.1: Candidates for Merging

5.3 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.

Consider the example illustrated in Table 5.1. These are actual cases that were isolated by the redundancy mechanism in the cable TV failure diagnosis domain.

5.4 An Example

Table 5.2 illustrates the necessary information to detect redundancy. 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 key words, the two cases are considered further for redundancy. For example, in Table 5.2, Case 1 and Case 2 share five (5) key words. Each case has six (6) key words. Therefore, these cases share 83% key words. The application initially sets the redundancy threshold to 80%, so if this threshold is not modified by the user, then the application will signal redundancy for

Case 1
Case 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
Case 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 : jamming sticky envelopes laser heat glue

Table 5.2: Detecting Redundancy in the Printer Repair Domain

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 matches the cases on entire case content 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 redundancy and if so, which case should be removed from the case base. The user is also offered the option of editing one of or both cases to make the distinction between them more apparent.

5.5 Redundancy Detection Algorithm

There are 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 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 $N = ||CaseBase||$

Key Word Index Check = $O(N)$

Redundancy Detection = $O(N)$

Overall Complexity: $2(O(N)) = O(N)$

The second algorithm, 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.

Let $N = ||CaseBase||$.

For each case $C (1 \dots N)$,

Perform R_1 .

Overall Complexity: $O(N) * O(N) = O(N^2)$

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. These thresholds are originally set by the application, but can be changed by the user.

5.5.1 Trigram Matching

Trigram matching is the pattern matching algorithm used for query retrieval in CaseAdvisor™ Problem Resolution. 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. The drawback of using trigram matching is that cases that are morphologically related will return high scores even if they are not semantically related.

5.5.2 Check Incoming Case for Redundancy

1. Run **Algorithm R_1** to check an incoming case for redundancy.
2. Send the incoming case, C , to the Information Retrieval Module to “normalize” its structure. In this process the key words will be extracted from the new case. If, for some reason, the information retrieval module does not extract any key words, then the key words will be extrapolated from the case name and problem description locally. This means that the key words for that case may only be relevant for that case and not the case base in its entirety. This is necessary for the redundancy module to work.
3. Compare the key words discovered in the new case to the key word index. Remember that both the key word and the list of documents in which that word appears are stored in the key word index.

Let C_1 and C_2 be two cases.

Let $k(C_1)$ represent the set of key words in case C_1 .

Let θ represent a threshold that can be defined by either the user or the application. The application defined threshold is 80%.

- (a) For each case C_i in the case base.
 - If $k(C_1) \cap k(C_i) > \theta$, then case C_1 and case C_i are identified as “possibly” redundant.
 - If there were cases identified as possibly redundant, perform a trigram match on the cases in their entirety. If this match produces a sufficiently high result, present the cases to the user.
 - If $trigramMatch(C_1, C_i) > T$, cases are equivalent. T is initially set to 95 by the application.
 - If $trigramMatch(C_1, C_i) > S$ continue to the next step. S is initially set to 80 by the application.
- (b) For each field in C_1 and C_i .
 - Remove matching strings.
 - If one case has a string left containing a key word, it is subsumed by the other.

- If the cases match on any of the fields, they may be candidates for merging.
4. If redundancy or subsumption exists or the application believes the cases are candidates for merging, present both cases and the explanation to the user. If not add the case to the case base.
 5. If the user does not believe that the cases are redundant, s/he can include an explanation that will be stored with the case. That way the two cases will not be flagged each time the case base is tested for redundancy.

5.5.3 Check Entire Case Base for Redundancy

1. Run **Algorithm R_2** to check an entire case base for redundancy.
2. Retrieve the key word index.
3. For each case C_i
 - (a) Send C_i to the R_1 algorithm for one case as described above.

5.6 Design Rationale

The redundancy detection module pares down the choices by making use of the key word index which is stored in a hash table. The first step is to extract the likely cases causing redundancy. This extraction is based on the percentage of shared key words. One scan of the key word index is sufficient to mark an incoming case as a candidate for redundancy or to remove it from further consideration. This avoids extensive testing on unrelated cases.

Let X_1 be the new incoming case.

Let $k(X_1)$ represent the set of key words in X_1 .

If the case base itself is free from redundancy and the key words are evenly distributed, then there can only be $T * ||k(X_1)||$ candidate cases for redundancy where T is a threshold between 0 and 1. T represents the fraction of key words that two cases must share before being identified as possibly redundant. (T is initially set by the application at 0.8, but can be modified by the user.) If the case base is free from redundancy, then no two cases in the

case base share more than $(T * 100)\%$ key words. Therefore, the number of cases that can be marked for further testing must be less than or equal to the number of key words in the incoming case.

Although the worst case time for this algorithm is $O(N)$ as mentioned earlier, it is more than likely that the algorithm will run much faster. Assuming that the number of key words in the incoming case is much less than the size of the case base, then this algorithm will run very quickly relative to the size of the case base.

5.7 Discussion

This chapter discusses the strategies used to identify redundancy within a case base. The algorithms to detect redundancy in an incoming case and to detect redundancy within an entire case base are presented. The redundancy detection module uses a normalized representation of the cases and applies the redundancy detection algorithms. If the case successfully passes through the redundancy detection module, then the application passes the case through to the inconsistency detection module. Inconsistency detection is performed if the case is deemed not redundant by the application.

Chapter 6

The Inconsistency Detection Module

6.1 Addressing the Knowledge Acquisition Problem

I have identified three different ways to code the knowledge about a particular domain. The first method is to compile the rules into the case based engine in a programming language such as C++. The functions in this programming language can then code the if-then rules. This method is very efficient. However, the rules are difficult to update and require recompilation every time a change is needed. These disadvantages are significant; case based reasoning is used because of the relative ease of increasing the power of the case base through incremental updating. Using this system, the user would be required to collate information about the domain prior to the development of the system. A case based reasoning system should be flexible enough to allow updates without recompilation. Also, the users of the system should be able to input new rules themselves.

A second option is to use logic rules, in the same manner as in an expert system. The advantage of this method is that the rules do not have to be recompiled when they are updated. However, like rule based expert systems, they are difficult to obtain and maintain. The users of the system may have little or no experience with logic formulations, increasing the difficulty of rule authoring. Inexperienced users typically find formulating boolean rules confusing [15].

As the indexing language has now been established through the use of information retrieval techniques, I decided to use string based rules to represent the guidelines. Essentially, these rules have the same structure as the cases and therefore, the indexing language can be applied to both the rules and the case base allowing for easier comparison.

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, I call these rules “guidelines”.

6.2 Guideline Representation

Guidelines are simply contradictory combinations of key words or phrases. If the conjunction of the words in a rule appears in a case, then an alert is issued to the user.

Therefore, if K represents the key word set, guidelines can be expressed as $k_1 \wedge k_2 \wedge \dots \wedge k_n$ where $\{k_i \in K | i = 0 \dots n \wedge |K| = n\}$. An example guideline in the printer-repair domain may look as follows:

Guideline : line printer toner cartridge

Line printers do not use toner cartridges, so this combination of words should probably not appear in a case. Range inconsistencies can also be defined. These rules are referred to as adjacency guidelines.

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 further examined to determine the adjacency of the words within the guideline. In the second example, the word “channel” is located and then the words directly following are tested to determine if there is a number greater than “89”. If so, a 100% chance of contradiction is reported to the user. If one of the words in a guideline appears in the case name, one appears in the case description and the remaining two appear in the case solution, it is not likely that the

guideline is violated by that case. However, if the words from the guideline are within some threshold T number of words, then the guideline is violated and the case and the guideline are then presented to the user.

6.3 Inter-case Representation

The strategy to detect inter-case inconsistency is very similar to detecting intra-case inconsistency. Inter-case inconsistency occurs when two or more cases are composed together and a contradiction arises due to this composition. Therefore, inter-case inconsistency is identified when:

- Let C represent the case base and $\{c_i | c_i \in C \wedge i = 1..n \wedge |C| = n\}$.
- Let R represent the set of guidelines and $\{r_i | r_i \in R \wedge i = 1..n \wedge |R| = n\}$.
- Let $k(c_i)$ represent the set of key words in c_i .
- Identify inter-case inconsistency if $k(r_i) \in k(c_1) \cup k(c_2)$. That is, inter-case inconsistency arises violated Guideline i if the union of the key words in case c_1 and c_2 are a superset of the key words contained in guideline r_i .

Simply put, two cases should not be composed together if the union of their key words is a superset of one of the guidelines. If the combination of words within a guideline can be found in the union of two cases, then that guideline can be violated when the two cases are composed together. Again the composition should occur within one field of the case. For example, if the first two words appear in the name field of the first case and the latter words of the guideline appear in the solution field of the second case, it is unlikely that the guideline will be violated when the two cases are combined.

6.4 Inconsistency Detection Algorithms

Detecting intra-case inconsistency requires the use of background knowledge in the form of guidelines. These guidelines consist of known impossibilities or anomalies that rarely occur. As case based reasoning is generally applied to poorly structured and understood domains [54], these guidelines can be entered at anytime by the user. Once a person becomes familiar with the system or receives feedback from other users of the system, adding guidelines should become easier. Soft constraints, as well as hard constraints, can be modelled. The guideline provided earlier, **line printer toner cartridge**, is an example of a hard constraint. No

line printer uses a toner cartridge. If this combination of words exists adjacently in a case, then a warning must be issued. In Case Maintainer, the concept of adjacency is first set by the application, but can be modified by the user.

The rule base is organized into an index similar to the key word index. For each term that appears within the rule base, the set of guidelines in which that term appears is noted. The guideline index, therefore, is a list of terms and the corresponding documents or guidelines in which they appear. This index is stored as a hash table and, as such, inserting, deleting and retrieving terms is very efficient.

6.4.1 The Intra-case Inconsistency Detection Algorithm

1. Send incoming case to information detection module.
2. Match key words of the incoming case to each guideline in the rule base using the guideline index.
If all words in the guideline match, continue.
3. If all words in the guideline are adjacent to each other in the case, signal inconsistency.
Adjacency initially means in the same field.

To analyze the complexity of this algorithm, let us define $M = ||rulebase||$. For each rule in the rule base, the key words in the case must be compared to the rule. In the worst case, the case violates every rule. So the complexity of this algorithm is simply $O(M)$. The size of the rule base should be much less than the size of the case base, particularly for large case bases. Thus, the complexity of the inconsistency detection algorithm should be smaller than the complexity of the redundancy detection module.

6.4.2 An Example

An example of using guidelines to detect intra-case inconsistency follows. The incoming case and the guideline that it violates are illustrated in Table 6.1

The Process

1. Preprocessing step

Let the incoming case be C_1 ; the case is illustrated in Table 6.1.

Case Name: laser printer printed page black
Case Solution: <i>laser printer</i> printed page black unseated <i>ribbon</i> reseat toner cartridge reprint document reseat toner cartridge turn laser printer press button release latch printers require removing paper tray first
Guideline: laser printer ribbon

Table 6.1: Example of a Normalized Case and the Guideline it Violates

Let the key words in C_1 be represented as $k(C_1)$

$k(C_1) = \{\text{toner, cartridge, **ribbon**, tray, press, button, release, latch, toner cartridge, page black, paper tray, **laser printer**\}$.

Retrieve the key word index.

For each guideline R

- (a) If $|k(C_1)| < |k(R)|$, then C_1 can not contain all of the key words in R , then exit. In the example, C_1 contains 12 key words or phrases. The guideline R only contains 3, so the algorithm proceeds.
- (b) If $k(R) \subseteq k(C_1)$ then C_1 is possibly inconsistent. All of the words within the guideline R appear within C_1 , so proceed.

If no guideline is violated, then end the algorithm.

2. Adjacency step

Each term in R is in C_1 , but they may be in different fields or far enough apart to be unrelated. This step determines that the terms are reasonably close to one another.

For each term in R , r_i ,

- (a) Determine the position of each occurrence of r_i within C_1 . For example, **laser printer** appears in the first two positions in both the Case Name and Case Solution fields. The word **ribbon** appears in the seventh position in the Case Solution field.
- (b) If the user has not specified an adjacency threshold, then the position can be represented by the name of the fields in which the term appeared. If the user has specified an adjacency threshold, the position will be represented by the field and the position in which the term appears in that field. As all three terms in

the rule appear within the same field, Case Solution, in C_1 , the case is deemed inconsistent by Case Maintainer.

- (c) If any word does not appear within the same field or within the adjacency threshold if it is specified as the others, finish.

If all of the words in R are within threshold T positions, alert the user of possible inconsistency. Display both C_1 and R to the user.

6.5 Design Rationale

The inconsistency detection module was designed to be similar to the redundancy detection module. Essentially, the algorithms are identical except that the inconsistency detection module relies on the guideline index and the threshold is always equal to 1 whereas the redundancy detection module relies on the key word index and the threshold is initially set to 0.8.

The guideline index is represented as a hash table to facilitate inserting, deleting and retrieving terms from the index. Assuming that the guidelines do not overlap significantly, a case should not violate a large number of guidelines. The worst case scenario with this algorithm is the incoming case violating every guideline in the rule base. A more likely scenario is a case violating zero (0) or one (1) guidelines. In this case, the algorithm will run in linear time with respect to the number of guidelines.

6.6 Discussion

The strategies for inconsistency identification are presented in this chapter. The inconsistency detection module relies heavily on user input. The user is responsible for building the rule (or guideline) base that Case Maintainer uses to test for inconsistency. It is then critical that Case Maintainer automates the rest of the task. This is done by testing a case twice. The first step is to test if the terms within the guideline appear within the case. If they do, the next step is to test if those terms appear adjacently within the case. Adjacency is defined as occurring within the same field within a case.

Speed is not crucial during the Case Authoring part of case based reasoning; quality is of higher importance. The redundancy and inconsistency algorithms are typically applied

to one case at a time. The combination of these facts not only means that the redundancy and inconsistency detection modules will still produce results quickly, but if they do not, the competence of the case based reasoner should not be affected.

The redundancy and inconsistency detection modules work very similarly. I observed the parallels in applying the information retrieval techniques to redundancy and inconsistency. I decided to take this analogy to the next step applying it to the algorithms to detect both inconsistency and redundancy. Both algorithms use indices and thresholds. This will make the maintenance of Case Maintainer simpler.

Chapter 7

Empirical Testing

I have implemented the agent architecture in the framework of the CaseAdvisor™ system¹ developed by the Case Based Reasoning Group at Simon Fraser University. 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. Its advanced functionalities includes case authoring, problem resolution, interactive planning, and case adaptation. CaseAdvisor™ has been applied to many different help-desk applications in industrial settings.

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

- The information retrieval approach for processing unstructured 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.
- 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.
- The inconsistency detection module is capable of detecting intra-case inconsistencies

¹To receive an evaluation copy, contact <http://www.cs.sfu.ca/cbr>.

through the use of string based guidelines. Again, controlled introduction of inconsistent cases will be generated to test the system.

- The preliminary testing indicates that the combination of the modules described above can produce a competent case based reasoner.

7.1 Testing the Information Retrieval Module

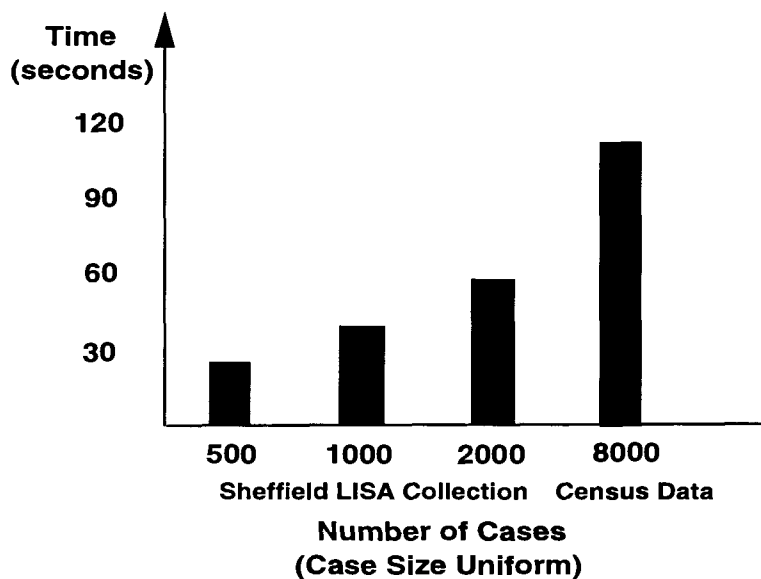


Figure 7.1: CPU Time To Apply Information Retrieval Techniques

Figure 7.1 demonstrates that even the one time cost of normalizing a case base is not that expensive. The time displayed is the 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. An average case is presented in Table 7.1. The empirical testing proves that the information retrieval module can handle cases of that size in a reasonable amount of time. When applied to an actual case base designed to diagnose cable TV failures,

Case Id : 1502
Case Name : Libraries in the Faroe Islands
Abstract : The 1st public library in the Faroe Islands (Faero County Library, now called Foroya Landsbokasavn) was established in 1928. Describes its development despite many acute problems, deterioration after the death of the librarian in 1878, recovery from 1920 (thanks to a new librarian), and movement into a new building in 1980. Outlines the financial situation of the island's public libraries and gives figures for stock, borrowers, and loans. Draws attention to the situation of the Faroese language - it was not taught in primary schools until this century; 80-100 books and some small publications are published in Faroese annually; and most printed matter is in Danish.

Table 7.1: Typical Case Extracted from the Sheffield LISA Collection

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.

Table 7.1 illustrates a case from the Sheffield LISA collection to present an idea of the level of detail contained in a "typical" case.

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 at Roger's Cablevision.

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. Figure 7.2 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. Without tweaking, the key words generated by Case Maintainer matched 87% of the key words generated by those familiar with the cable domain. The only list that Case Maintainer did not match at least 80% of terms with was a whopping 116 words provided by a subject with limited domain expertise and computer experience.

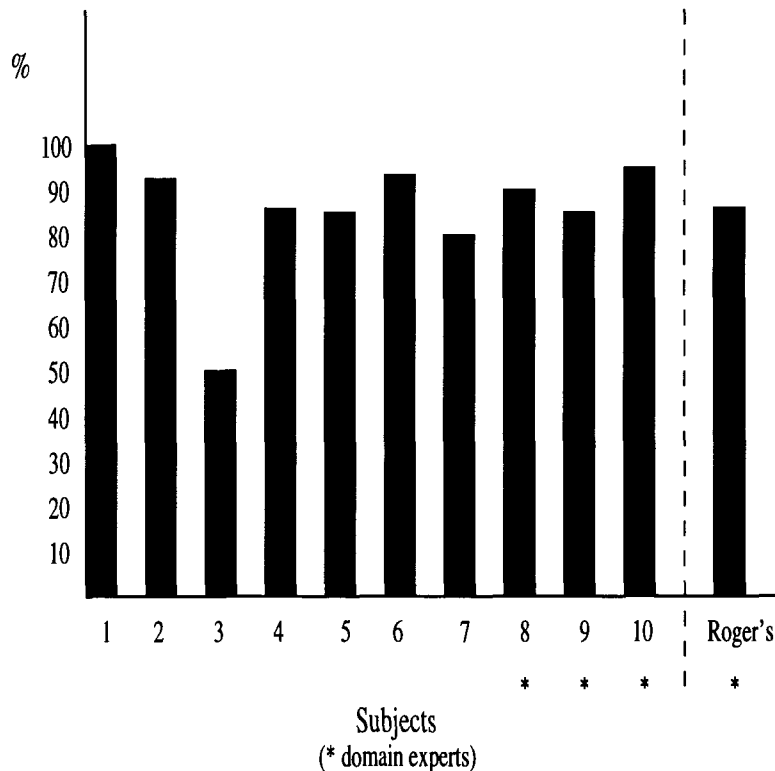


Figure 7.2: Percentage of Key Words Matching Those Found by Case Maintainer

7.2 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.

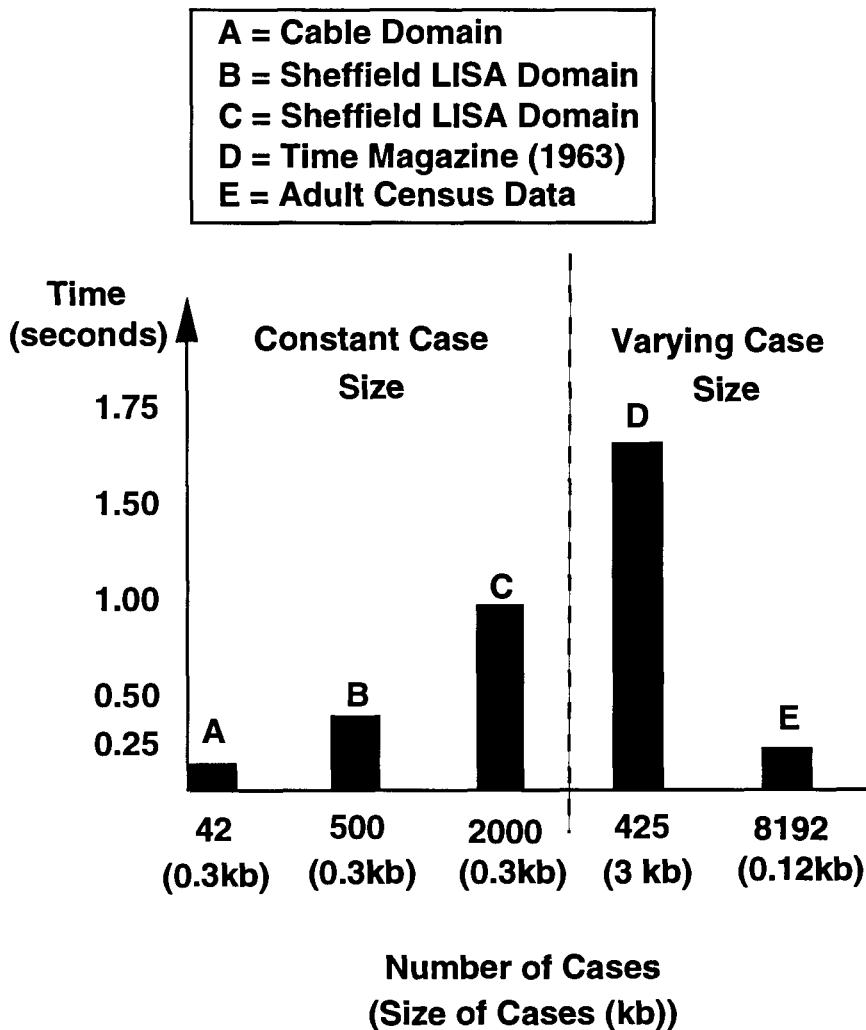


Figure 7.3: CPU Time to Detect Redundancy

Figure 7.3 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 Figure 7.3 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, 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 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% 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. 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. A sample decision tree is available in Appendix A. Figure 7.4 presents the results of this experiment.

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

Figure 7.4: Quality of Redundancy Module

Figure 7.4 demonstrates the efficacy of the 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,

<p>Sample Case (0.3 kilobytes) Case Name: There is no cable and the screen is black. Case Problem Description: The screen is black, there is no sound on any channel and there is snow in your picture. 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 then disconnect the cable. Replug the cable in 30 seconds. If there is still no reception, the problem is most likely in the TV set.</p>
<p>Generated Guidelines Guideline: no cable black screen electrical connections tune TV set channel replug cable no reception problem TV set bubba Guideline: no black cable screen connections electrical tune TV set snow picture problem TV cable system bubba</p>
<p>Violated Guideline: check electrical connections tune TV set channel 3 disconnect cable replug cable 30 seconds no reception problem TV set</p>

Table 7.2: Sample Case and Automatically Generated Guidelines

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 as not redundant. This means that 88% 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 caveat is that all of the cases involved in this experiment were derived from the same source, limiting the generalization of these results.

7.3 Testing the Inconsistency Detection Module

The performance of the inconsistency detection module is dependent on the rule base. The efficacy of this module is dependent on the quality of the rules in the rule base and the efficiency is dependent on the number of rules in the rule base. Therefore, the efficacy is difficult to measure unless done in a real situation, but the efficiency of the module can still be illustrated easily.

Tests were only applied to incoming cases rather than on the case base in its entirety.

I tested the inconsistency detection module with a number of different rules and timed the process. As soon as a rule is violated, processing stops and the case and the rule are presented to the user. Once the user makes the required change, the processing continues. I tested the case base with rules that did not cause any violations and rules that did render a case inconsistent. Remember that as soon as a term within a rule does not match with a case, processing ceases. Therefore, I devised guidelines that would require as much processing as possible to test the system under stringent conditions.

Guidelines were devised that were as long as the largest field in the cases. These guidelines were automatically derived from the incoming case. Substrings of the case were removed and nonsense strings were appended to the guidelines. These substrings consisted of strings matching the ordering of terms in the case, a random collection of terms contained in the case and reversed strings. To illustrate a trivial example of generating guidelines, consider the case "one two three four". Generated guidelines would include strings such as "one two four", "one two three four", "one three four bubba" and "one two three bubba". The first two guidelines should cause violations and the following two should not. The generated guidelines share most terms with the cases to cause the application to have to do more processing. One of the actual sample cases and the system generated guidelines are presented in Table 7.2.

As demonstrated in Figure 7.5, the inconsistency module can complete processing on a three (3) kilobyte case and forty (40) guidelines in under one (1) second. The numbers in Figure 7.5 reflect the average CPU time needed to complete the inconsistency detection process. Three trials were completed for each variation in rule base size. The case being added remained constant over all three trials, but the rules were varied. Although the content of the rules was manipulated, the length of the rules and the number of key words the rules shared with the test case were maintained. Essentially, the three time trials returned similar results.

7.4 Testing the Competence of the Case Based Reasoner

The testing in this section concentrated solely on the cable domain. The case base that was developed by the Case Based Reasoning Group and beta tested at Roger's Cablevision was used as a benchmark. We compared this case base and a case base generated by Case Maintainer in a variety of ways. For the remainder of this section, I will refer to the case base

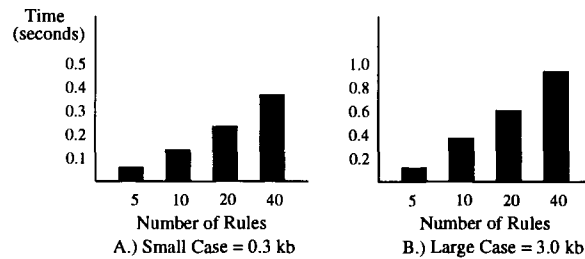


Figure 7.5: CPU Time to Detect Inconsistency

developed down at Roger's as RogerCable and the case base developed by Case Maintainer as MyCable. Retrieval accuracy and efficiency are the two main advantages of using case based reasoning. A case based management system must not degrade these two processes. Retrieval efficiency can simply be measured through time tests of both case based reasoning systems. There were no significant differences in retrieval. Due to the identification of three (3) redundant cases by Case Maintainer, MyCable was slightly faster at retrieval. As the different case bases retrieve distinct sets of cases for each query, the retrieval efficiency was further measured in the next experiment.

Table 7.3 demonstrates that Case Maintainer can build an appropriate case base using statistical information retrieval techniques. The key words in Table 7.3 were generated by combining the results from the experiment in which subjects were required to extract key words and phrases from a text file. These key words appeared in 80% of the respondent's key word and phrase lists.

I then used the key words and phrases as queries to RogerCable and MyCable. The first case retrieved is presented in Table 7.3. In one third of the instances, both case bases return the same case. For the rest of the key words, the case retrieved was also retrieved by the other case base, just in a different order. The obvious exception to this is the word "snow" for which RogerCable could provide no solution. To test the efficacy of the cases retrieved, I examined the frequency with which the retrieved cases are used by the system. Out of the five cases where the first case retrieved differed, MyCable returned the more frequently used case three (3) times. Therefore, aside from the one instance where RogerCable could not support the query, the case bases are operating at approximately the same level of accuracy. Considering that MyCable was obtained automatically, this is a positive result.

The test indicates that MyCable appears to be slightly more efficient and effective than the original case base. However, as statistical measures were used to generate key words,

Query (Key Word)	RogerCable (Case Name of First Case Retrieved)	MyCable (Case Name of First Case Retrieved)
sega	sega channel problems	sega channel not working
black	close caption; black screen	black screen; no cable
converter	converter hook up problems	converter hook up problems
descrambler	PAY TV channels scrambled	descrambler hook up problems
snow	NO CASES RETRIEVED	picture problem; snowy picture
channel	converter hook up problems	all channels except PAY TV channels working
no reception	poor reception with cable direct to TV	poor reception
dpv parental	sega channel problems	dpv parental control code problems
no cable	poor reception with cable direct to TV	no cable; more than one TV in home

Table 7.3: First Case Retrieved Given Key Words For Each Case Base

Subject	Educational Level	Discipline
1	B.A.	Psychology
2	Master's	English
3	Master's	English
4	B.A.	Gerontology
5	Master's	Physics
6	PhD	Geography
7	PhD	Computer Science
8	PhD	Computer Science
9	B.A.	Political Science
10	B.A.	Criminology

Table 7.4: Educational Status of Subjects from Competence Experiment

phrases and questions, I felt that further testing was necessary to determine if the new case base was as intuitive and effective to users.

I used ten (10) subjects for the following experiment, Competence. These subjects had no prior exposure to my thesis. Table 7.4 shows the level of education and major for each subject. The subjects were provided with a list of key words and phrases and asked to query both case bases. The key words and phrases were those generated by over 80% of the subjects. I removed the word “snow” from consideration as it would bias the experiment in favour of MyCable. The subjects were not informed as to which case base was which. Six (6) of the ten subjects had never used a case base reasoner before. This seemed to have no impact on the answers provided in the questionnaire. After using both case bases, the subjects were required to answer a questionnaire. Following this, I asked the subjects a number of questions regarding the usability of the case bases. Note that the user interface for both case bases is exactly the same. The only variable is the case base itself.

The results from the Competence questionnaire are presented in Tables 7.5 and 7.6. In Table 7.5, the scores presented are the average scores across the subjects who answered the questions. Each question is associated with a range of scores from 1 to 5, with 5 representing the most positive response and 1 representing the most negative. For the first two questions, all of the subjects answered the questions. However, the question asking if the subjects noted any inconsistencies only drew two (2) responses. The two subjects who answered this question were familiar with the domain. The other users, who had little

Question	RogerCable	MyCable	Not Answered
Max Score = 5.0	Avg. Score	Avg. Score	# Subjects
Are the cases retrieved relevant to the queries used?	4.2	4.3	0
Did the cases retrieved appear to answer your queries?	4.2	4.4	0
Was there any evidence of redundant cases?	2.8	1.0	3
Was there any evidence of inconsistent cases?	4.0	1.0	8

Table 7.5: Competence of the Two Case Bases

experience with the domain, did not note any inconsistencies in either case base. Seven (7) subjects responded to the question regarding the existence of redundant cases. All of the subjects who answered the question regarding redundant cases identified the redundancy present in RogerCable, but did not see any redundancy in MyCable.

The first two questions relate to the competence of a case base. There were very few differences in the responses for these questions. Most subjects felt that cases retrieved did answer or solve the queries they posed to the case base. The between case base scores did not vary greatly within subjects. Thus, MyCable retains a high degree of coverage and a low degree of reachability for the case base. According to the two domain experts, MyCable also maintains a high degree of intra-case consistency for the entire case base. This satisfies the criteria I have set forth for a competent case base reasoner.

The second phase of the questionnaire asked the respondents to directly compare the two case bases. Most of the respondents deemed that the two case bases were equivalent. MyCable did not fare well in the question about understandability. I will focus on possible reasons for this later. RogerCable was rated slower by almost half of the subjects. I believe this is because Case Maintainer removed three redundant cases from the case base and thus returned fewer cases for some queries. Although, the retrieval time is essentially equivalent,

Question	RogerCable	MyCable	Same
	# Subjects	# Subjects	# Subjects
Which case base was quicker?	1	4	5
Which case base was more accurate?	2	3	5
Which case base answered queries more effectively?	2	3	5
Which case base was more easily understood?	5	1	4
Overall, which case base appeared more effective?	2	2	6

Table 7.6: Comparing the Case Bases

this may have led the subjects to assume that MyCable was answering queries more quickly than RogerCable. In general, both case bases were rated very similarly.

Most of the comments that I received during the interviewing process were favourable. The users enjoyed using the Problem Resolution module and felt that they had learned about the cable domain. However, two points were repeated by a majority of the subjects. Some of the phrases extracted by Case Maintainer have no semantic meaning. This lack of meaning seemed to cause the user to find MyCable less user friendly and more confusing. Five (5) subjects cited this as a problem and indicated that it affected their answer on the understandability question as described in Table 7.6.

There are further problems with the phrase extraction when applied to case retrieval. The phrase extraction algorithm simply measures how often two or more words appear together within the same field. This means that if the phrase “no cable” is identified as a key phrase, the case “**no** sound on sega channel; connect **cable**” is retrieved. This is also a problem with the original case base brought about by the use of trigram matching. However, if a simple parser or a more rigid adjacency algorithm is employed during the phrase extraction process, this problem can be alleviated. Ultimately, the user must be responsible for editing the phrases.

A different issue raised by four (4) of the subjects focused on the case scoring mechanism. Case Maintainer scores cases higher than the original case base if key words or phrases are contained in the query. The users liked this higher scoring. However, if a user types in the query “hook up” and then answers “Yes” to the question “Problems with hook up”, the updated scores for the corresponding cases are too high. This problem is being addressed by a different member of the case based reasoning group who is working on a module to automatically answer the questions related to the query without reflecting this answer in the case scores.

7.5 Discussion

The purpose of this chapter was to demonstrate that Case Maintainer can effectively build and maintain case bases. Experimental results were presented supporting this claim. Each module was tested for the CPU time that it needed to process differing amounts of data. These preliminary results clearly show that Case Maintainer can efficiently and effectively manage a case base.

Chapter 8

System Development

This chapter presents a real life example of the features of Case Maintainer, implemented using Visual C++ Version 4.2. This example is based on the text file included in Appendix B. The text file represents a set of cases designed to diagnose cable failure at Roger's Cablevision. It was designed by the customer sales representatives at Roger's to facilitate training new representatives and contains 42 cases. This chapter is designed to provide the reader with a vision of the end product.

8.1 Information Retrieval Module

Case Maintainer's first step is to convert a text file to a case base. The text file must contain cases that have the structure illustrated in Table 8.1. The case name, case description and the case solution should be delimited in some fashion. The default delimiters are *Case Name:*, *Case Problem Description:* and *Case Solution*, but the user is able to change these delimiters to parse any given text file. The only required field is *Case Name*.

8.1.1 Stop Word Removal

The first step in converting a text file to a case base is to build an inverted index. This index consists of all of the terms within the document. Each line of the index is a term, followed by its weight within a document, followed by the document number in which the term appears. Before the inverted index can be constructed, the stop words are removed from the documents. The user can use the default stop word file or edit the stop word file

<p>Case Name: no cable; black screen; tune local channel</p> <p>Case Problem Description: This may be a problem with your TV or the cable system.</p> <p>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.</p>

Table 8.1: Example Case in the Cable Domain

Example Stop Words
this
a
you

to make it appropriate for the current domain. Typical stop words are those words which either add little to the meaning of a case, may appear in every case or do both. A stop word for the cable domain could possibly be the word *cable* itself. Alternatively, the users of a printer repair domain would presumably choose to retain information about cables, but elect to ignore the word *printer*.

After the stop words have been extracted, the case in Table 8.1 will contain the following information.

Inverted Index

The case represented in Table 8.1 would have an inverted index that begins the same way as Table 8.3. The inverted index is a list of all terms in each document. If a term appears more than once in a document, it still only appears once in the inverted index but its weight

case name no cable black screen tune local channel
case problem tv or cable system
case checked electrical connections tv set channel 3
disconnect cable still no reception problem tv set

Table 8.2: Example of Removal of Stop Words

Term	Weight	Document
case	3	1
name	1	1
no	2	1
cable	2	1
black	1	1
...	.	.
...	.	.
...	.	.

Table 8.3: Example Inverted Index

is increased. The frequency with which the term appears within a document is stored as the weight of the term. The document number is also recorded.

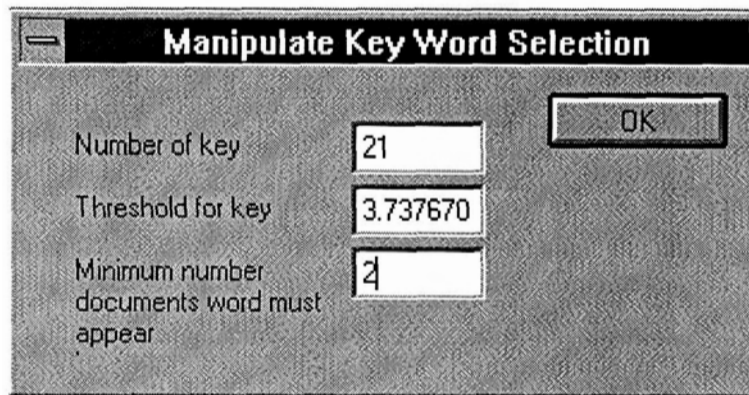
8.1.2 Key Word Extraction

The application then uses the inverted index to extract key words. Key words are those words which appear frequently within a small set of cases and infrequently everywhere else in the case base. The application requires three thresholds to complete the key word extraction. These thresholds are initially set by the application, but can be modified by the user.

1. Number : The number of key words that the user wishes to extract should be identified. This number is originally set to be equal to the number of cases in the case base. The key words are sorted so that the strongest key words are selected. This threshold is based on the number of key words that most case base reasoning systems contain.
2. Threshold : The application uses the inverse document frequency measure to build the key word index. Therefore, the threshold is initially set to be the natural logarithm of the number of cases.
3. Documents : This threshold specifies a range identifying the number of cases in which a word should appear before it is identified as a key word. This is to remove words that appear infrequently within cases, but frequently across cases, from consideration.

A word that appears in every case is not a useful word for distinguishing between cases. This threshold is initially set to be between 2%-6% of the number of cases. Again, the user retains ultimate control and is permitted to change the threshold. A warning is issued if the user changes the threshold to greater than 30% of the cases.

The initial values for these thresholds are provided in Figure 8.1 for a case base that contains 42 cases.



Field	Value
Number of key	21
Threshold for key	3.737670
Minimum number documents word must appear	2

Figure 8.1: Thresholds for Key Word Extraction

The key word extraction functions build an index for the key words that is similar to the inverted index, but for each term includes the entire set of documents in which the term appears.

8.1.3 Editing the Key Word File

Case Maintainer allows the user to add and delete key words. The only restriction is that any key word to be added must appear in at least one case. All cases may not be equally representative. If a particular case encompasses a large set of problems, the words in that case may be of specific importance. However, this information may not be detected by the key word extraction algorithm.

8.1.4 Phrase Extraction

After extracting the key words, Case Maintainer extracts key phrases. Phrases are identified if two words frequently appear together within the same field. The application requires two thresholds to complete the phrase extraction. These thresholds are initially set by Case

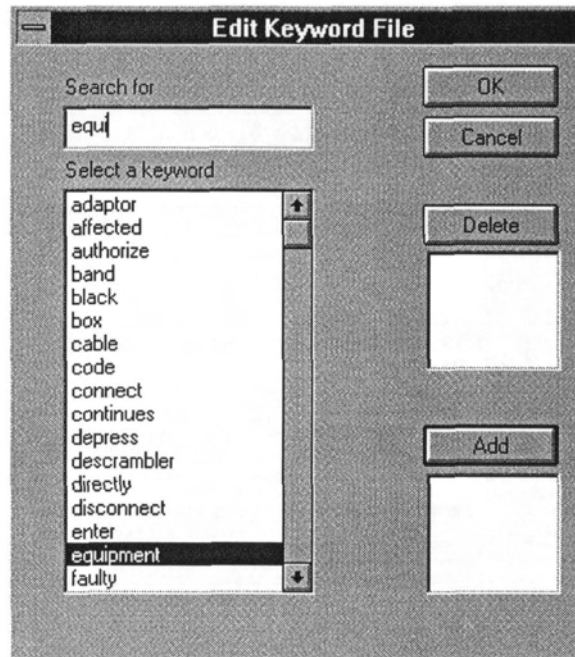


Figure 8.2: Editing the Key Word File

Maintainer, but can be modified by the user. Each phrase must contain at least one key word.

1. Number : The number of phrases that the user wishes to extract should be identified. The number is originally set to the number of cases in the case base.
2. Threshold : The threshold is a measure of term cohesion. It must be a decimal number between 0 and 1. It measures the likelihood, $\text{coh}(A, B)$, where A and B are both terms that appear within the text file, that if term A appears in a particular case, then term B will also appear as well.

The phrase extraction functions build an index for the phrases that include all the documents in which the phrase appears.

8.1.5 Editing the Phrase File

The phrase file can be edited in the same manner as the key word file.

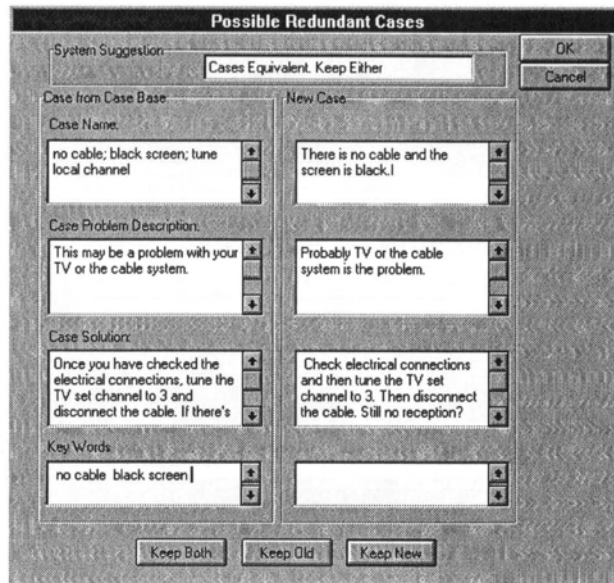


Figure 8.3: Example of Possible Redundancy

8.1.6 Question Extraction

Given the key words and the phrases that appear in each case, a number of questions can be extracted from the case. These questions can be used by Problem Resolution to pinpoint the case in which the user is interested. A default threshold of 3 questions per case is defined by the system. However, in the instance where the case based reasoner contains particularly large cases or small cases, this number may need to be modified.

There are three thresholds for question extraction.

1. Number : The number of questions per case. This number can be modified for the entire case base or on a per case basis. It is originally set by Case Maintainer to be 3.
2. Before + After: After extracting phrases and key words from the given case, it is necessary to form a question. The default is : "Problems with " phrase "?".

8.2 Redundancy Module

After the key words and phrases have been identified, the redundancy module can be applied. Alternatively, the user can choose to apply the redundancy module at a later time. The redundancy module makes use of the key word index to identify cases that share a suspicious

Create a New Case

Case Information:

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:
Dnce you checked the electrical connections, tune the TV set channel 3 and disconnect the cable. If there's still no reception, the problem is most likely in the TV set. screen black no sound on any

Extract

Questions:

Problems with no cable?
Problems with system connections?
Problems with tune local?

Thresholds
Add
Delete

Key words:

cable
tune
picture
set

Add

OK
Cancel
Get Case
Save Case
Clear Fields

Figure 8.4: Adding a sample case to the cable domain

number of key words. Two cases that share over 80% of their key words are marked as “possibly redundant”. As each case is considered, the redundancy module extracts the key words in the current case. Using the key word index, it can quickly identify possible redundancy.

Figure 8.3 illustrates two cases from the text file that were identified as redundant. A dialog box presents the users with both cases along with a system suggestion. The user can modify none, either or both of the cases. The user is given the option of keeping both or just one of the cases. The default choice is made by Case Maintainer. If the application believes that the new case subsumes the case already in the case base, it will suggest that the user retain the new case and discard the old. If Case Maintainer believes that the two cases are essentially equivalent, that information will be offered to the user. Additionally, the algorithm will suggest that the user may wish to merge two cases if they share a common field, such as *Case Solution*.

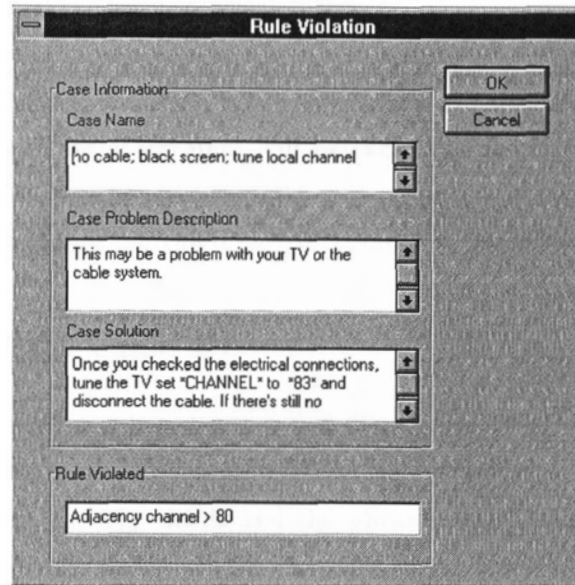


Figure 8.5: Rule Violation in the Cable Domain

8.3 Inconsistency Module

The inconsistency module can be applied iteratively as the case base is being structured or it can be applied after all the processing is completed. To apply the inconsistency module, a rule base is required. The user can add string based guidelines to the rule base. The rule base can be edited at any time, because the initial information known about the domain may not be sufficient to represent the appropriate background knowledge. The rules do not have any required format - the application simply tests if all of the words existing in the rule are also contained in a case. The user can also specify rules testing equality, greater than and less than relationships.

If a case seemingly violates a rule, the rule and the case are presented to the user. At this point, the user may edit the case, delete the case, edit the rule, or do nothing. If the user deems that either the case does not violate the rule or that the rule may be flexible, s/he can simply elect to keep the case. Otherwise, the user may choose to delete the case or to edit the offending sections of the case. A case that violated one of the rules in the cable domain is presented in Figure 8.5. Notice that the rules are in free form text.

8.4 Adding a New Case

If the user chooses to add a new case, the redundancy and inconsistency modules can be turned on or off. First, Case Maintainer will extract the key words and phrases from the new case. Then, it will check if the new case will cause a redundancy relationship in the case base. If this is the case, then the dialog illustrated in Figure 8.3 will appear.

8.5 Merging Two Case Bases

The user has the option of merging together two cases bases. The user can choose to merge two text files or two case bases. Key word and phrase extraction is performed on the newly merged file. In this way, the key words and the phrases will be identified according to all of the information that now exists within the case base. Redundancy and inconsistency identification can be performed as the merge occurs.

8.6 Save the Case Base

Given all of the information extracted above, the application can now save the text file in case base format. If a case does not contain any key words, then the key words from that case are extracted from the case name. The phrases and key words that are in a case represent the key words through which the case can be accessed in the Problem Resolution module of CaseAdvisorTM. The questions extracted from each case are also saved. The question containing the phrase or key word with the highest weight is given the highest weight in the question index also. The user does have the option to change these weights at any time.

The case base is saved in a manner that makes it immediately usable by the Problem Resolution or Case Authoring modules of CaseAdvisorTM.

8.7 Discussion

This chapter illustrates the process Case Maintainer undertakes to maintain case bases. The case base used in this chapter is taken directly from Roger's Cablevision and demonstrates that even small case bases contain redundant and inconsistent information. Out of the 42 cases provided, one was inconsistent and three were redundant. This information was

elicited from one person with experience in the domain. Often, legacy data sources contain the work of several different authors. Presumably, these data sources would contain even more redundant and inconsistent information than the case base presented in this chapter.

This chapter illustrates how redundancy and inconsistency can occur within a case base. It also presents examples of redundancy and inconsistency that can be detected by Case Maintainer. The text file used for this example was not changed before being converted by Case Maintainer.

The final purpose of this chapter is to illustrate the process of indirect management. At each step, the user can override any of Case Maintainer's functionality. Depending on the user preferences, Case Maintainer will automate much of the process or very little. This style of management gives the user ultimate control over the system.

Chapter 9

Conclusions and Future work

9.1 Contributions

This thesis has provided a theoretical and empirical framework for the design of an agent to maintain large and unstructured case bases.

1. A unified view of the previous research in the area of case base management was presented illustrating the lack of a global approach in this area. Research was examined from different areas in artificial intelligence.
2. This thesis provides a new definition for the competence of a case based reasoning system. We add a third criterion, **consistency** to the two criteria, **coverage** and **reachability**, offered by [48].
3. This thesis provided a theoretical framework for the case base management problem. Previous approaches focused on only one or two details within the management problem. This is the first attempt to comprehensively describe the management problems within a case base.
4. Knowledge management is a concern of the Artificial Intelligence community. This problem has grown exponentially over the last few years due to the Internet, cheaper memory, enhanced data bases, and the computerization of industry. The amount of knowledge being collected is staggering. This thesis presented a unified view of the research thus far and potential problems in applying this research to real problems. This thesis also provides a new definition of management.

5. This thesis provided an empirical framework for case base management. The design of an agent to semi-automate the case base management process was discussed and implemented. This agent is available in CaseAdvisor™ - it is called Case Maintainer. Case Maintainer has been tested on a number of different domains supporting the claims of this thesis.

9.2 Future Work

The following is a list of possible avenues of future research.

Information Retrieval Techniques: The information retrieval techniques used in my application were sufficient. However, with the remarkable amount of research being conducted in this area due to the explosion of the Internet, other techniques could be used to possibly increase both the efficiency and the efficacy of the Information Retrieval module.

1. Retaining information in the inverted index to improve the efficiency of the redundancy and inconsistency detection modules. This would also involve a storage space vs. processing time analysis.
2. Adding a simple Natural Language parser to the phrase extraction function to increase the semantic meaning of the phrases.
3. Adding additional power to the stopper function so that not only stop words, but also those words that appear too frequently within the case base to assist in indexing, are removed.

Redundancy Detection:

1. Investigate whether adding a simple Natural Language parser improves redundancy detection without degrading efficiency.

Inconsistency Detection:

1. Different representations of guidelines should be tested.
2. Automatic construction of guidelines generated from the information in the case base should be tested.

Dynamic Organization:

1. Design dynamic organization module according to specifications outlined in this thesis.
2. Test this module to ensure that case retrieval time of frequently used cases is quicker and that the case retrieval time of infrequently used cases is bearable.

Evaluation:

1. Empirical testing is only significant when more subjects are tested. Even though the preliminary results indicate that Case Maintainer is a viable management agent, further testing must be performed to strengthen this claim. Also, testing by actual users of the module is required.
2. Further evaluation of the inconsistency detection module is required by domain experts.
3. Different case bases must be acquired and tested.

Appendix: A.) A Decision Tree

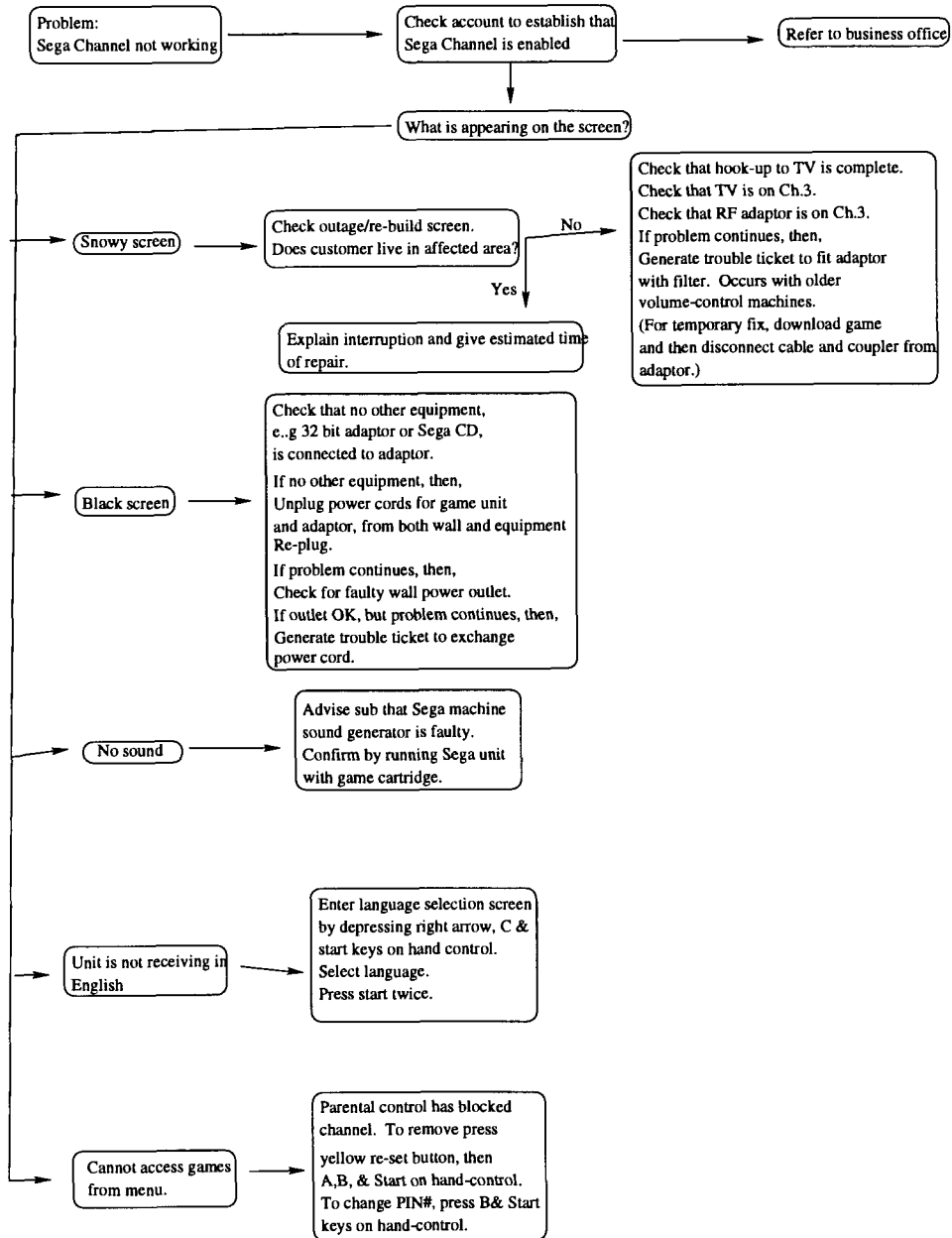


Figure 9.1: Sample Decision Tree for Redundancy Experiment

Appendix: B.) Cable Cases

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 checked the electrical connections, tune the TV set channel to 333 and disconnect the cable. If there's still no reception, the problem is most likely in the TV set.

Case Name: no picture; white screen; faulty TV, descrambler, converter

Case Problem Description: Usually, this is a faulty TV set but the descrambler may be the problem.

Case Solution: Connect cable directly to TV; if there is a picture, the descrambler is the problem. Roll out a truck.

Case Name: picture problem; snowy picture; temporary broadcast problem

Case Problem Description: If it's just one channel, it's a temporary broadcast problem. If not check to make sure all your cable connections are tight.

Case Solution: Connect cable directly to TV. If the problem persists, dispatch truck to home.

Case Name: no picture; blank screen with white band; faulty TV

Case Problem Description: No picture a single white line across the middle of your TV set.

Case Solution: This is normally a faulty TV set. Call your TV repairman.

Case Name: double image picture; check fine tuning

Case Problem Description: Ghosting or double images, or another picture in the background.

Case Solution: Check the fine tuning (be sure automatic fine tuning (AFT) is off fine tuning.

Case Name: speckled bands across picture; interference from other equipment

Case Problem Description: Speckled bands across the picture can be interference from something

Case Solution: Turn off appliances. If the problem persists, dispatch truck.

Case Name: picture flashing on one channel; temporary broadcast problem

Case Problem Description: Picture flashing on one channel may be a temporary breakdown

Case Solution: If the problem persists or all channels are affected, call us.

Case Name: picture problem; too small/large; call TV repairman

Case Problem Description: Picture too large or too small for screen.

Case Solution: Call your TV repairman if the problem persists.

Case Name: forgot DPV parental control code, use the A34 command

Case Problem Description: Often a subscriber will forget the parental control code

Case Solution: They can bring in the converter to be reset physically or we can do it automatically.

Case Name: entering DPV parental control code

Case Problem Description: forget how to set the DPV parental control code

Case Solution: Choose 4 digit parental control code Press 'LEARN'; displays 'LE' Press 'PC/PM' or '*'; displays 'LP' Press 'ENTER'; displays flashing 'LP' Enter code; Press 'ENTER'; displays channel DPV box

Case Name: changing DPV parental control code

Case Problem Description: Select uncontrolled channel

Press 'LEARN' Press 'PC/PM' or '*' Enter old code Press 'ENTER' Enter new code Press 'ENTER' Case Solution: activate DPV deactivate DPV parental control DPV box

Case Name: DPV parental control activation

Case Problem Description: forget how to activate DPV parental control code

Case Solution: Select channel to be controlled Press 'F' Press 'PC/PM' or '*' Press 'PC/PM' again Press 'PC/PM' again Channel is now parental lock capable Press 'F' Press 'PC/PM' Enter code Press 'ENTER' activate DPV parental control DPV box

Case Name: DPV parental control deactivation

Case Problem Description: forget how to deactivate DPV parental control code

Case Solution: Select channel to be cleared Press 'F' Press 'PC/PM' Enter code Press 'ENTER' Selected channel is now unlocked DPV box deactivate DPV parental control

Case Name: VCR Plus will not record

Case Problem Description: VCR Plus will not work with older model VCRs without remote control connected by a wire to the VCR.

Case Solution: Buy new VCR, or return VCR Plus. remote attached by wire VCR pre 1987

Case Name: VCR Plus should record, VCR problem

Case Problem Description: VCR Plus works great with cable TV.

Case Solution: It works best with newer model VCRs with number key pad on remote control

Case Name: hook up dual tuner PIP-TV, VCR, DPV, and closed caption unit

Case Problem Description: Customers will want to step through the hook up procedure

Case Solution: Leave in bypass mode for ANT 1 Signal through DPV for ANT 2 Leave CC unit on 3

Case Name: hook up DPV (without VCR) to stereo to receive surround sound

Case Problem Description: To receive surround sound, they must have a stereo TV.

Case Solution: Use audio outputs to connect to auxiliary on the amplifier stereo TV DPV box

Case Name: hook up dual tuner PIP TV with VCR (no pay TV)

Case Problem Description: Must have a PIP TV and VCR.

Case Solution: Take customer through hook up procedure. PIP TV VCR Premium Channels

Case Name: hook up VCU switcher with VCR and Starbase descrambler

Case Problem Description: give step by step instructions to hook up

Case Solution: Before programming, set TV to channel 3, VCU "WATCH" button to "VCR". If you don't, no instructions will appear on screen for programming.

Case Name: close caption hook up with DPV

Case Problem Description: give instructions to hook up

Case Solution: Must have closed caption box and DPV parental control unit.

Case Name: Sony PIP TV (2 tuners) hook up, converter, descrambler, VCR

Case Problem Description: give step by step instructions to hook up

Case Solution: VCR must be on channel 3 for taping with this set up.

Case Name: converter and Starbase descrambler hook up

Case Problem Description: give instructions to hook up

Case Solution: Note: set TV to channel 3

Case Name: VCR and converter hook up; record premium channels

Case Problem Description: step by step instructions to hook up

Case Solution: Allows recording of any channel include "Premium Channels". This method does not allow viewing one channel while recording another.

Case Name: VCR and converter hook up; record basic channels

Case Problem Description: step by step instructions to hook up

Case Solution: Allows recording of basic channel while viewing any channel. This method does not allow recording of "Premium Channels".

Case Name: DPV parental control of MuchMusic

Case Problem Description: step by step instructions to hook up

Case Solution: Install MuchMusic trap at pole/ipb program DPV to lock out channel

Case Name: no cable; more than one TV in home

Case Problem Description: This may be a cable or problem with one TV.

Case Solution: Disconnect all but one TV and check picture. If no picture send a service technician to check it out.

Case Name: cable problems; additional equipment hook up

Case Problem Description: Disconnect all additional equipment and connect cable directly to TV.

Case Solution: If there's still no picture, send service tech. If there is a picture, the VCR or converter is affecting the cable.

Case Name: no cable; receive channels up to 13

Case Problem Description: Customer possibly has not switched the CATV switch.

Case Solution: Locate CATV function on TV/VCR and switch to "CABLE".

Case Name: close caption hook up with DPV parental control unit

Case Problem Description: step by step instructions to hook up

Case Solution: customer must have both close caption box and DPV parental control unit.

Case Name: Hong Kong TV will not work with VCR or converter box

Case Problem Description: Hong Kong TV will not work with VCR or converter box

Case Solution: TV set requires re-tuning to accommodate NTSC signal. Either take TV to K.S. Audio/Video for re-tuning or generate trouble ticket.

Case Name: sega channel not working

Case Problem Description: error message 0003, 0005, 002F,002C

Case Solution: re-authorize adaptor depress yellow button on top of adaptor

Case Name: sega channel not working

Case Problem Description: error message 0003, 0005, or 002F

Case Solution: re-authorize adaptor depress yellow button on top of adaptor

Case Name: sega channel not working; screen freezes

Case Problem Description: downloading game screen freezes sega channel not working

Case Solution: re-authorize adaptor depress yellow button on top of adaptor

Case Name: sega channel not working snow screen

Case Problem Description: sega channel not working; snowy screen

Case Solution: check outage screen if customer does not live in affected area check hook up complete TV is on channel 3 RF adaptor on channel 3

Case Name: black screen sega channel not working

Case Problem Description: black screen and sega channel not working

Case Solution: disconnect additional equipment unplug power cords for game unit and adaptor from both wall and equipment replug

Case Name: no sound sega channel not working

Case Problem Description: no sound and sega channel not working properly

Case Solution: Sega machine sound generator is faulty

Case Name: sega unit not receiving in English

Case Problem Description: sega unit is using incorrect language selection

Case Solution: enter language selection screen by depressing right arrow, C and start keys on hand control select language press start twice

Case Name: cannot access sega games from menu

Case Problem Description: cannot access sega games from menu, parental control has blocked channel

Case Solution: to remove parental control, press yellow re-set button, then A, B and start on hand-control to change PIN press B and Start keys on hand control

Case Name: segal channel not working

Case Problem Description: error message 002C

Case Solution: re-authorize adaptor, depress yellow button on top of adaptor

Case Name: no reception on channels 5 or 6

Case Problem Description: no reception on channels 5 or 6 dues to incorrect 'CATV' setting

Case Solution: set 'CATV' switch on TV set remote is set to TV setting

Case Name: no reception on channels 2-6

Case Problem Description: no reception on low band

Case Solution: check no splitter on cable fine tune TV channels if problem continues, unplug TV for 30 seconds replug if problem continues, generate trouble ticket

Case Name: no reception of channels 7-13

Case Problem Description: no reception on high band

Case Solution: check no splitter on cable fine tune TV channels if problem continues, unplug TV for 30 seconds replug if problem continues, generate trouble ticket

Bibliography

- [1] A. Aamodt and E. Plaza. Foundational issues, methodological variations and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1993.
- [2] D. Aha. Case-based learning algorithms. *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*, 1:147–158, 1991.
- [3] R. Alterman and D.Griffin. Improving case retrieval by remembering questions. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1:678–683, 1996.
- [4] K.D. Ashley. Reasoning with reasons in case-based comparisons. *Proceedings of the First International Conference on Case-Based Reasoning (ICCBR-95)*, pages 133–144, 1995.
- [5] K. Bellman. The modelling issues inherent in testing and evaluating knowledge-based systems. *Expert Systems with Applications*, 1:199–215, 1990.
- [6] L.A. Breslow and D.W. Aha. Simplifying decision trees. *Knowledge Engineering Review*, In press, 1997.
- [7] W. Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.
- [8] D.W. Cheung, A. W. Fu, and J. Han. Knowledge discovery in databases: A rule-based attribute oriented approach. *Proceedings of the 1994 International Symposium on Methodologies for Intelligent Systems*, 1:164–173, 1994.
- [9] M. Cox and A. Ram. An explicit representation of forgetting. *Proceedings of the Sixth International Conference on Systems Research*, 6:115–120, 1992.

- [10] J.J. Daniels and E.L. Rissland. A case-based approach to intelligent information retrieval. *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1:238–245, 1995.
- [11] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. *From Data Mining to Knowledge Discovery*, pages 1–34. In Fayyad et al. [12], 1996.
- [12] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurasamy, editors. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, Menlo Park, California, 1996.
- [13] William B. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms*. Prentice-HALL, North Virginia, 1992.
- [14] S. Franklin and A. Graesser. Is it an agent or just a program?: A taxonomy for autonomous agents. *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*, 1:21–35, 1996.
- [15] D. Gelbart and J.C. Smith. Towards combining automated text retrieval and case-based expert legal advice. *Law Technology Journal*, 1:19–24, 1992.
- [16] I. Guyon, N. Matic, and V. Vapnik. *Information Patterns and Data Cleaning*, pages 181–203. In Fayyad et al. [12], 1996.
- [17] J. Han and Y. Fu. *Exploration of the Power of Attribute-Oriented Induction in Data Mining*, pages 399–424. In Fayyad et al. [12], 1996.
- [18] D. Hennessy and D. Hinkle. Applying case-based reasoning to autoclave loading. *IEEE Expert*, 7(5):21–27, 1992.
- [19] D. Hinkle and C. Toomey. Applying case-based reasoning to manufacturing. *AI Magazine*, Spring:65–73, 1995.
- [20] H. Kaindl. Verification and validation of knowledge-based systems using semiformal representation. *Proceedings of the AAAI 96 Workshop on Verification and Validation of Knowledge-Based Systems*, pages 7–16, 1996.
- [21] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufman, California, 1993.

- [22] J. Kolodner and W. Mark. Case-based reasoning. *IEEE Expert*, 7(2):5–6, 1992.
- [23] P. Koton. Using experience in learning and problem solving. *MIT, Laboratory of Computing Science*, PhD. Thesis Dissertation, 1988.
- [24] D. Leake. Constructive similarity assessment: Using stored cases to define new solutions. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 313–318, 1992.
- [25] B.P. Lientz and B. E. Swanson. Problems in application software maintenance. *Communications of ACM*, 24(11):763–769, 1981.
- [26] B.P. Lientz and G.E Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978.
- [27] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–41, 1994.
- [28] S. Markovich and P. Scott. The role of forgetting in learning. *Proceedings of the Fifth International Conference on Machine Learning*, 1:459–465, 1988.
- [29] 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.
- [30] S. Minton. Qualitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42:363–391, 1990.
- [31] N. Negroponte. *The Architecture Machine: Towards a More Human Environment*. MIT Press, Cambridge, Mass., 1979.
- [32] T. Nguyen, M. Czerwinski, and D. Lee. Compaq quicksource—providing the consumer with the power of ai. *AI Magazine*, Fall:50–60, 1993.
- [33] M. Pearce, A.K. Goel, J. Kolodner, C. Zimring, L.Sentosa, and R. Billington. Case-based design support : A case study of architectural design. *IEEE Expert*, 7(5):14–20, 1992.
- [34] R. Perkins. Diagnostic system designer. *Manufacturing Intelligence*, 10:16–19, 1992.

- [35] A.D. Preece. Towards a methodology for evaluating expert systems. *Expert Systems*, 7(4):215–233, 1990.
- [36] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
- [37] K. Racine and Q. Yang. On the consistency management of large case bases: the case for validation. *Proceedings of the AAAI 96 Workshop on Verification and Validation of Knowledge-Based Systems*, pages 84–90, 1996.
- [38] K. Racine and Q. Yang. Maintaining unstructured case bases. *Proceedings of the 2nd International Conference on Case Based Reasoning*, To appear, 1997.
- [39] A. Ram and J.C. Santamaria. Continuous case-based reasoning. *Proceedings of the AAAI-93 Workshop on Case-Based REasoning*, pages 86–93, 1993.
- [40] E. L. Rissland and J.J. Daniels. A hybrid cbr-ir approach to legal information retrieval. *Proceedings of the Fifth International Conference on Artificial Intelligence and Law*, 1:52–61, 1995.
- [41] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. Computer Science Series McGraw Hill Publishing Company, New York, 1983.
- [42] B. Schneiderman. Direction manipulation: A step beyond programming languages. *IEEE Computing*, 16(8):57–69, 1988.
- [43] R. Shank, A. Kass, and C. Riesbeck. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, New Jersey, 1994.
- [44] J. Shavlik. Finding genes by case-based reasoning in the presence of noisy case boundaries. *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, 1:291–303, 1991.
- [45] H. Shimazu and Y. Takashima. Detecting discontinuities in case-bases. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1:690–695, 1996.
- [46] E. Simoudis. Using case-based retrieval for customer technical support. *IEEE Expert*, 7(5):7–13, 1992.

- [47] B. Smyth and P. Cunningham. A comparison of incremental case-based reasoning and inductive learning. *Proceedings of the 2nd European Workshop on Case-Based Reasoning*, 1:32–39, 1995.
- [48] 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.
- [49] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland, 1988.
- [50] J.E. Vargas and S. Raj. Developing maintainable expert systems using case-based reasoning. *Expert Systems*, 10(4):219–225, 1993.
- [51] M. M. Veloso and J. Carbonell. Derivational analogy in prodigy. *Machine Learning*, 10(3):249–278, 1993.
- [52] A. Vermesan and T. Bench-Capon. Techniques for the verification and validation of knowledge-based systems: A survey based on the symbol/knowledge level distinction. *Software Testing, Verification and Reliability*, 5:233–271, 1995.
- [53] A. Vermesan. A definition of subsumption anomalies in conceptual models of object-oriented kbss. *Proceedings of the AAAI 96 Workshop on Verification and Validation of Knowledge-Based Systems*, pages 17–24, 1996.
- [54] I. Watson. Case based reasoning tools: An overview. *Proceedings of the Second UK Workshop on Case Based Reasoning*, 1:71–88, 1996.
- [55] S. Wess, K. Althoff, and M. Richter, editors. *Topics in Case-Based Reasoning*. EWCBR-93, Germany, 1993.
- [56] D. Wettschereck, D. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, In press, 1997.
- [57] P. Winston. *Artificial Intelligence*. Addison-Wesley Publishing Company, Massachusetts, 1992.

- [58] G. Wolff. Learning and reasoning as information compression by multiple alignment, unification and search. *Computational Learning and Probabilistic Reasoning*, 1:67–85, 1996.
- [59] Q. Yang, E. Kim, and K. Racine. Caseadvisor: Supporting interactive problem solving and case base maintenance for help desk applications. *IJCAI 97 Workshop on Practical Uses of CBR*, To appear, 1997.