Static and Dynamic Feature Weighting in Case-Based Reasoning(CBR)

by

Zhong Zhang

B.Sc. (Comp. Sci.) Nanjing University 1990

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

SIMON FRASER UNIVERSITY
December 1997

APPROVAL

Name: Zhong Zhang

Degree: Master of Science

Title of Thesis: Static and Dynamic Feature Weighting in Case-Based Reasoning(CBR)

Examining Committee: Dr. Lou Hafer
Chairperson
Chair

Dr. Qiang Yang
Senior Supervisor

Dr. Jiawei Han
Supervisor

Dr. Mehrdad Said
External Examiner

Date Approved: Dec 9 97

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

Static and Dynamic Feature Weighting in Case-Based Reasoning (CBR)

Author:

(signature)

Zhong Zhang

(name)

Dec. 10, 1997

(date)

Abstract

Case-based reasoning(CBR) is a recent approach to problem solving, in which domain knowledge is represented as cases. The case retrieval process, which retrieves the cases most similar to the new problem, depends on the feature-value pairs attached to cases. Different feature-value pairs may have different importance in this process, which is usually measured by what we call the *feature weight*.

Three serious problems arise in the practical applications of CBR regarding the feature weights. First, the feature weights are assigned manually by humans, not only making them highly informal and inaccurate, but also involving intensive labor. Second, a CBR system with a static set of feature weights cannot cater to a specific user. It would be desirable to enable the system to acquire the user preferences automatically. Finally, a CBR system often functions in a changing environment, either due to the nature of the problems it is trying to solve, or due to the shifting needs of its user. We wish to have a CBR system that always adapts to the user's changing preferences in time. These three problems comprise one of the core tasks of case base maintenance problem.

Our approach to these problems is to maintain feature weighting in both the static and dynamic contexts. The static feature weighting method grasps the irregular distribution information of feature-value pairs within a case base. Our intuition is that the more cases a feature-value pair is associate with, the less information it conveys. The dynamic feature weighting method examines the feature weights in a changing environment. We integrate a neural network into CBR, in which, while the reasoning part is still case based, the learning part is shouldered by a neural network. We hope that this integrated framework would be a living system, which learns a user's preferences over time, and simulates these preferences with its own behavior.

We propose and implement the underlying algorithms for these two feature-weighting methods. Our empirical tests produce the optimal results that we desire, and confirm our hypotheses and claims. Our work contributes much to the research in the maintenance of knowledge bases.

Acknowledgements

I am greatly indebted to my senior supervisor, Dr. Qiang Yang, not only for his technical contributions to this thesis, but also for his insights to many facets of case-based reasoning. He has been a constant source of inspiration, without which this thesis would not have been possible. His patience and encouragement during the struggling periods of this thesis are very much appreciated.

I thank my supervisor Dr. Jiawei Han. I benefited much from the course I took under his instruction. Thanks are also given to Dr. Mehrdad Saif. He made valuable comments about my thesis. Furthermore, I am grateful to my defense committee to give me this valuable chance to present my thesis.

Many thanks to my fellow colleagues in the CBR group at Simon Fraser University. The pleasant working environment and research atmosphere there have contributed much to my thesis.

I would like to pay tribute to my fellow graduate students for their comments on the system in which the algorithms in thesis are implemented.

Finally, I acknowledge the constant support from my family.

Contents

Abstract iii
Acknowledgements iv
List of Tables ix
List of Figures xi
1 Background 1
1.1 Introduction to CBR 1
1.2 Applicability of CBR 4
1.2.1 Advantages 4
1.2.2 Disadvtanges 5
1.3 Successful Examples for CBR 5
1.3.1 CASEY – A Research Project 5
1.3.2 CLAVIER – An Industrial Application 6
1.3.3 Other CBR Applications 7
1.3.4 Some CBR Tools 8
1.4 Research Areas in CBR 8
1.4.1 Case Representation 9
1.4.2 Case Retrieval 10
1.4.3 Case Reuse 12
1.4.4 Case Revision 13
1.4.5 Case Retainment 13
1.4.6 Integrated Approaches 14
1.5 Relationship of CBR to Other Methods 15
1.6 Case-Base Maintenance Problem 16
1.7 Summary 17

2 Feature Indexing and Weighting 18
2.1 Introduction 18
2.1.1 A Small Example of Case Base 18
2.1.2 k-NN Algorithm 19
2.1.3 Features in CBR 20
2.1.4 Weights in CBR 21
2.2 Related Work 26
2.2.1 Learning Agents 26
2.2.2 Case-Based Learning 28
2.2.3 Feature Weighting 30
2.2.4 Introspective Learning 31
2.2.5 Other Learning Methods 32
2.3 My Method 35
2.4 Outline of Thesis 36
3 Static Weight Adjusting Method 37
3.1 Introduction 37
3.2 Formal Description of Static Adjusting Model 38
3.3 Algorithm for Static Adjusting Method 41
3.3.1 An Example 41
3.3.2 Implemental Considerations 42
3.3.3 General Algorithm 44
3.3.4 Computational Analysis of Algorithm 46
3.4 Summary 46
4 Dynamic Weight Learning Method 47
4.1 Introduction 47
4.2 Machine Learning in Knowledge-Based Systems 49
4.3 Neural Networks 50
4.3.1 Components in A Neural Network 50
4.3.2 Delta Rule 52
4.3.3 Backpropagation Neural Networks 53
4.3.4 Mathematical Description 54
4.3.5 Bayesian Neural Networks 56
4.4 Motivations for Integration 56

4.5 Learning Model and Mathematical Description 58
4.5.1 User's Learning Model 58
4.5.2 Mathematical Description 60
4.5.3 Learning Policy and Updating Policy 65
4.5.4 Learning Process 65
4.5.5 Computational Analysis of Algorithm 67
4.5.6 Real-Time Response 68
4.6 Summary 68
5 System Design and Development 70
5.1 Introduction 70
5.1.1 Case-Authoring Module 70
5.1.2 Problem-Resolution Module 71
5.2 Design for Static Weight Adjusting Component 73
5.2.1 Adjusting Weights Statically 73
5.2.2 Using Statically Adjusted Weights 76
5.3 Design for Dynamic Weight Learning Component 77
5.3.1 Constructing and Using Solution Base 77
5.3.2 Using and Learning Weights Dynamically 79
5.4 Summary 83
6 Empirical Tests 84
6.1 Introduction 84
6.2 Test for Static Weight Adjusting Method 85
6.2.1 Experiment of An Artificial Case Base 85
6.2.2 Experiment for Roger's Cable Case Base 91
6.2.3 Repetitive Adjusting 95
6.2.4 Running Time for Static Weight Adjusting Method 95
6.3 Test for Dynamic Weight Learning Method 95
6.3.1 A User's Preference 96
6.3.2 Experiment for Movie Case Base 97
6.3.3 Experiment for Roger's Cable Case Base 107
6.3.4 Some Analysis 111
6.4 Summary 113
7 Conclusion and Future Work 115

7.1 Summary 115
7.2 Limitations 117
7.2.1 Limitations of Static Weight Adjusting Method 117
7.2.2 Limitations of Dynamic Weight Learning Method 117
7.3 Future Work 119
Bibliography 122

List of Tables

1.1 What Types of CBR Are Companies Using? 7
1.2 CBR Products and Their Developers 8
1.3 First Example of Case Representation 10
1.4 Second Example of Case Representation 10

2.1 A Small Case Base of Four Loan Cases 19

4.1 Scenarios for Knowledge-Based Systems(K) and Learning Systems(L) 49

6.1 Distribution of Cases among Question-Answer Pairs 86
6.2 Percentage Ranges and Minimum Weights for Two Tests 86
6.3 First Adjusting Result for Each Question-Answer Pair 87
6.4 Typical Cases and Their Associated Q-A Pairs 87
6.5 Second Adjusting Result for Each Question-Answer Pair 89
6.6 Distribution Information in Roger's Cable Case Base 91
6.7 Three Sets of Adjustment Parameters for Roger's Cable Case Base 92
6.8 Adjustment Results for Three Sets of Adjustment Parameters 93
6.9 A User's Preference(Part p) 99
6.10 Initial Problem Retrieval Result 99
6.11 Problem Retrieval Result after First Round 100
6.12 A User's Preference(Part s) 100
6.13 Training Process of Solutions 101
6.14 A User's Preference(Part p) 102
6.15 Problem Retrieval Result after Five Rounds 103
6.16 A User's Preference(Part s) 105
6.17 Solution Retrieval Result after Three Rounds 106

6.18 A User's Query 107
6.19 Case Retrieval Result 108

List of Figures

1.1 The Cycle of Case-Based Reasoning(CBR) 2

3.1 When is A Feature-Value Pair Good or Bad? 39
3.2 Distribution of Feature-Value Pairs 41
3.3 Weights of Feature-Value Pairs 42
3.4 Mapping from Feature-Value Pairs to Groups 44

4.1 An Artificial Neuron 51
4.2 A Possible Architecture of Neural Network 52
4.3 A BackPropagation Neural Network 53
4.4 New Structure of A Case Base 58
4.5 User's Learning Model 59

5.1 Case-Advisor System 71
5.2 Static Information in A Case Base 74
5.3 Step 1 in Static Weight Adjusting Process 75
5.4 Step 2 in Static Weight Adjusting Process 76
5.5 Step 3 in Static Weight Adjusting Process 77
5.6 Create A New Solution 78
5.7 Associate A Solution to A Case 79
5.8 Back-propagation Neural Network Learning in Problem-Resolution Module . 80
5.9 A User's Own Weights 81
5.10 Problem's Confirmation or Disapproval 82
5.11 Solution's Confirmation or Disapproval 83

6.1 Cases-Scoring Process of First Test 88

6.2 Case-Scoring Process of Second Test 90
6.3 Adjusted Weights vs. Initial Weights for Roger's Cable Case Base 94
6.4 Training Process for Problem $P3$ 104
6.5 Error Convergence Chart for Seven Highest Cases in Seven Queries 110
6.6 Error Convergence Chart for Seven Second-Highest Cases in Seven Queries . 111

7.1 Normal Distribution of Weights 118

Chapter 1

Background

Case-based reasoning(CBR) is a recent approach to problem solving and knowledge reuse. In the field of artificial intelligence(AI), CBR differs from many other learning and problem-solving techniques in its representation of knowledge and learning process. Basically, a case-based reasoner will retrieve a case it deems most similar to a new problem currently being addressed, then it will adapt that case to fit the new problem, after that, the adapted case will be applied to solve the problem, which makes further adaptations possible, and finally, the new adapted case will be saved for future reuse. In CBR, knowledge is represented as cases which are stored for future reuse and analysis while the learning process is through the accumulation of new cases as well as through the refinement of the case retrieval quality.

In recent years, CBR gained wide acceptance both in the industrial world and in the academic field. These can be witnessed by many successful industrial applications[49] and an increased rate of research papers in CBR at international AI conferences[3, 26].

1.1 Introduction to CBR

A case base is composed of cases. As defined in [24], *A case is a conceptualized piece of knowledge representing an experience that teaches a lesson fundamental to achieving the goals of the reasoner.* Essentially, a case has two parts. The first part is a specific piece of knowledge the case grasps. The second part is the context with which that piece of knowledge is associated. This context defines a specific circumstance under which the case will be retrieved to form a solution to a new problem.

People use cases to help understand and access old situations and to help solve new

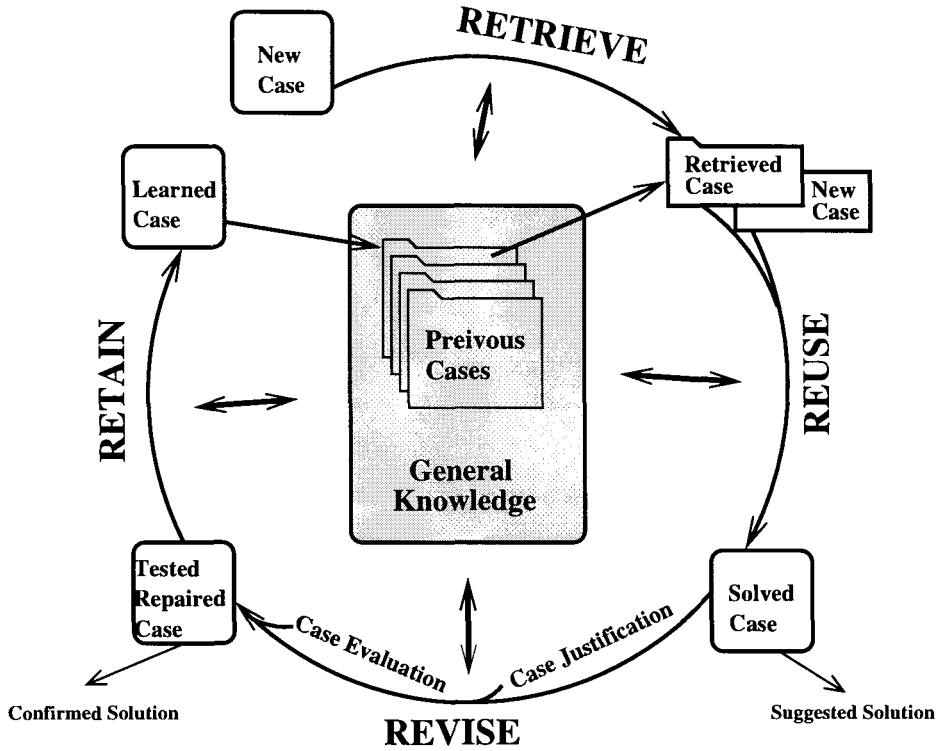

Figure 1.1: The Cycle of Case-Based Reasoning(CBR)

problems. In CBR, reasoning is viewed as a process of remembering one or a small set of concrete previous cases, and drawing decisions on the comparisons between them and the new situations. The intuition behind CBR is that situations do recur with regularity. What has been done before in a situation is likely to be a possible solution to a similar new situation, forming a good starting point in the solution-searching process.

Basically, we can view the process of CBR as a cycle[24, 41, 3]. Figure 1.1 is adapted from [3] and illustrates it. The cycle is composed of the following four general steps.

1. **Case Retrieval**: Retrieve the most similar case(s) to a new problem from a case base.

2. **Case Reuse**: Reuse the retrieved case to solve the problem.

3. **Case Revise**: Revise the retrieved case with the hope it could fit the new problem.

4. **Case Retain**: Retain the revised case as a new case in the case base.

The first and foremost step in CBR is to **retrieve** the most similar cases in a case base according to the new problem posed by a user. In many practical implementations, this new problem is often represented as a query, whose format is system-dependent. For instance, it can be of a free-text description, or a set of question-answer pairs. Unlike the query in a database application, a case-based reasoner may return a list of cases which are deemed most similar to the query. There is no necessary requirement that these cases match the query *exactly*. After the first step, a process called **case reuse** will be triggered so that the relevant portions of the retrieved cases will be extracted to form a solution to the new problem. Because the old problems rarely match the new one exactly, the formed solution must be adapted to fit the new problem. This process is called **case revise**. Usually the final step is **case retain**, in which the adapted case will be saved appropriately in the case base for future use.

There are two important sub-steps from **case revise** to **case retain** we must address[24]. After a proposed solution has been extracted, it often needs to be justified and evaluated before it can be tried to solve the problem in a new situation. When all knowledge necessary for this evaluation is available, this step can be thought as a validation step. However, in many real-world applications, there are too many unknown factors to make such a validation possible. Often, to justify whether or not the proposed solution is the best candidate, there is a need to recursively call memory processes to retrieve cases with similar solutions, and compare and contrast the proposed solution with them. As an example, if an already-known failing case with the similar solution exists, then the reasoner must consider whether or not the proposed solution might be of the same problem. In the next step, the proposed solution will be actually tried out in the real world. Feedbacks from its application to a new problem will be obtained and analyzed. If the result is met as expected, then there is no need for further analysis. Otherwise, the anomalies should be analyzed, and what resulted in them as well as what can be done to prevent them should be figured out. The purpose of this process is to give the reasoner a chance to evaluate how good its proposed solution is. There are several possible ways to do such an evaluation. Sometimes it is done using the comparisons with previous cases; sometimes it is done according to the feedbacks from the real world; and sometimes it is done using an artificial simulation. The evaluation will make possible the need for additional adaptations of the proposed solution.

1.2 Applicability of CBR

There is much evidence that people do use CBR in their daily reasoning process. Some of the evidence, which we can observe, just happens around ourselves. For instance, in order to prepare a meal for a dozen of guests, we often recall previous similar menus for possible suggestions. Some other evidence is the result from the research experiments. As cited in [24], Ross has shown that people learning a new skill often refer back to previous problems to refresh their memories on how to deal with the new tasks. Many research results, which were conducted in the car mechanics and the physicians' daily practices[24], further support this evidence. In addition, psychologists have also found that people always use cases to make decisions when they solve new problems[24].

1.2.1 Advantages

As a reasoner, CBR has many advantages as compared to other techniques.

- It provides proposed solutions quickly, avoiding the time necessary to obtain them from scratch. These proposed solutions form a good start point in solving new problems, although there is a need to evaluate them.
- It does not require that the domain be completely understood before a reasoner can draw proposed solutions to a new problem. It provides a chance to make assumptions and predictions according to what worked in the past without a complete understanding of the domain.
- It provides a reasoner a way for evaluating solutions when there is no algorithmic method available for such an evaluation. Again according to what worked in the past, it provides a possible method for dealing with a situation where there are too many unknown or undetermined factors for evaluating solutions.
- It provides a way for warning the potential problems that arose in the past for the new situations.
- It helps a reasoner focus on the important parts of a problem by figuring out what parts in a problem are important in the reasoning process. When a reasoner attempts to adapt a solution, it hopes that such an adaptation would include more of what resulted in previous successes and less of what caused previous failures.

1.2.2 Disadvtanges

There exists no reasoning process which is perfect without any weaknesses. Of course, CBR is not an exception to this fact.

- A case-based reasoner might rely totally on its previous experiences without any validation process when facing a new situation, which might result in incorrect solutions.
- A case-based reasoner might allow cases to bias too much in solving a new problem, which might result in inappropriate cases, costing valuable problem-solving time or leading to expensive errors.

All these disadvantages are related to using cases poorly or inappropriately to reason. However, CBR is a natural way of the reasoning of people, and the efforts of explaining the reasoning process in it might give us an opportunity to learn how to reason better using cases. In addition, the case memory technology will allow us to build decision-support systems that enhance the human memory by providing the appropriate cases while still allowing the human users to reason in a natural way. These will, undoubtedly, enable CBR to avoid the negative weaknesses, making it a promising solution as one of the reasoning and learning attempts in AI.

1.3 Successful Examples for CBR

1.3.1 CASEY – A Research Project

In [24], several automated case-based reasoners are described in detail. CASEY is a case-based diagnostician. Its input is the description of a new patient, which is composed of normal signs, present signs and symptoms. Its output is a causal explanation of the disorders the patient might have. It has a case base of approximately twenty-five cases, all of which were diagnosed by the Heart Failure Program(to be discussed later). When presented with a new patient, CASEY searches its case base to see if there exists a similar case. If such a case is found, it will be used to diagnose the patient. Otherwise, it passes the control on to the Heart Failure Program, which will diagnose the patient and then return its result to CASEY for future use. CASEY is constructed atop the Heart Failure Program, a model-based diagnostic program that diagnoses heart failures with unusual accuracy. However, CASEY does more than the Heart Failure Program does. CASEY includes two steps to do

a case-based diagnose for a new patient. First it searches the case base for similar cases and uses the model-based *evidence rules* to determine which of the retrieved cases is sufficiently similar to suggest an accurate and possible diagnosis solution. Then it applies the model-based *repair rules* to adapt the old solution to accommodate the new situation. For more detailed discussions on *evidence rules* and *repair rules* in CASEY, see [24]. While CASEY maintains the same accuracy as the Heart Failure Program, it performs much better if the efficiency is measured. It is shown that a speedup of two to three orders of magnitude is achieved if the CASEY holds a relevant case in its case base. The relevant cases provide a good starting point for constructing a causal explanation, and the time for constructing such a solution from scratch is saved. This system shows a potential to use CBR to speed up the model-based programs without any loss of accuracy.

1.3.2 CLAVIER – An Industrial Application

CLAVIER[24, 49] is one of the first commercial applications of CBR technology. It is a system for configuring the layout of composite airplane parts for curing in an autoclave. These parts are made up of composite materials. Each part has its own heating characteristics and must be cured correctly. Otherwise it will have to be discarded. One problem is that an autoclave's characteristics are not well understood. To make the problem worse, many parts will be fired in a single large autoclave, and will interact with each other to alter the heating and cooling characteristics of the autoclave. Fortunately, the experts in Lockheed kept a file of the previous successful layout configurations for deciding how to layout a new autoclave. However, this was still complicated by the fact that layouts were never identical because the different parts were required at different times and because the design of the parts kept changing. So there was a need to select a previous successful layout deemed to be similar to a new autoclave, and then adapt the old layout to fit the new one. This situation resembled the CBR paradigm closely. When Lockheed decided to implement a knowledge-based system to assist the operators, they chose CBR technology. In CLAVIER, each case is a layout of an autoclave, which is composed of the information describing parts, their relative positions in the autoclave, and the production statistics such as start and finish time, pressure, and temperature. The development of CLAVIER started in 1987, and it has been used since the fall of 1990. CLAVIER started with approximately twenty cases and has collected over one hundred since it has been used. As its experiences grow, it is becoming more accurate in its case retrievals, requiring considerably less adaptation now

Application Types	Companies
Customer Service	Black&Decker London Electric National Westminster Bank NCR
Fault Diagnosis	Apple Computers Caledonian Paper General Dynamics Matra Space Corporation
Help Desk	AT&T Bell Compaq Lucas Arts Nokia
Maintenance	British Airways Cfm International General Electric General Motors
Quality Control	Daimler-Benz ITT NEC Volkswagen

Table 1.1: What Types of CBR Are Companies Using?

than before.

1.3.3 Other CBR Applications

Shown in Table 1.1, which is adapted from [49], is a partial list of the types of CBR applications, as well as which companies are using them, and for what purposes they are being used.

CBR Products	Developers
ART*Enterprise CBR2	Inference Corp.
CASE-1	Astea International
CaseAdvisor	Simon Fraser Univeristy
CasePower	Inductive Solutions Inc.
Eclipse	The Haley Enterprise
ESTEEM	Esteem Software Inc.
KATE	AcknoSoft
ReMind	Cognitive Systems Inc.

Table 1.2: CBR Products and Their Developers

1.3.4 Some CBR Tools

Along another aspect, we can also see that more and more CBR tools have been developed and commercialized into the market. Table 1.2, which is adapted from a website[1], shows a partial list of CBR products and their developers.

1.4 Research Areas in CBR

In recent years, while CBR enjoys its success in the industrial world, it also attracts more and more attention from the academic field. On one hand, most of the research problems are directly from the CBR applications, on the other hand, the solutions to these problems in turn further promote its successful applications.

In general, there is no perfect method in AI which could be a universal approach to all the application domains. CBR could not be an exception to this fact[3]. Based on Figure 1.1, we can see that every step in the CBR cycle is crucial in the construction and use of a CBR system. In order to deal with different situations in different domains, different methods will be desired for each step. To find out which one is better for which application, it is always required that the theoretical proposals and empirical experiments be complemented.

[1] http://www.salford.ac.uk/survey/staff/IWatson/AI-CBR/commerce.html

1.4.1 Case Representation

The first step in developing a case-based reasoner is to build a case base. How to represent knowledge using cases is a nontrivial problem.

There are usually three major components to the content of a case[24].

1. *Problem Description:* the status of the situation where this case occurred, and if appropriate, what problem is needed solving at that time;

2. *Solution:* the explicitly stated or implicitly derived solution to the problem described in problem description;

3. *Outcome:* the resulting status of the situation when the solution was executed.

At present, there is no widely acceptable standard as to what information should be contained in a case. Of the three components in a case as above, the first two represent a shortcut for a case-based reasoner. However, as indicated in [24], in situations with many unknown or undetermined factors, severe inaccuracies might happen when the cases that only contain the problem descriptions and solutions are used to reason. A system, which mindlessly uses the knowledge it stores to solve problems and stores every new problem and solution, will become more and more efficient, but it will also suffer repeating its mistakes as often as its good solutions, making itself untrustable. To this end, the third part, outcome, is used to allow a reasoner to record and analyze the feedbacks from the outside environment. It will record what happens as a result of the execution of the solution, whether the result is a success or failure, in what way it succeeds or fails, and when available, an explanation of why for such a success or failure.

No matter what will be contained in a case, there are two basic pragmatic criterion we need to consider when deciding the information in a case: the functionality and the ease of the acquisition of the information that will be in a case[24, 49].

Once the information contained in a case for a particular domain or application has been decided, the general structure for the representation of this information is quite formal. Tables 1.3 and 1.4 are examples of two types of the representation of a case. It is easy to see that the case in Table 1.3 is detailed to some very small feature-value pairs, while the case in Table 1.4 only has two major parts: problem description and problem solution.

Feature	Make	Name	Type	Engine Size	Price	Doors
Value	Toyota	Camry	Luxury	8	$30,000	4

Table 1.3: First Example of Case Representation

Problem Description	Problem Solution
No reception on low band	1. Check no splitter on cable, fine tune TV channels.
	2. If problem continues, unplug TV for 30 seconds, replug.
	3. If problem continues, generate trouble ticket.

Table 1.4: Second Example of Case Representation

1.4.2 Case Retrieval

The retrieval step starts with a new problem description, and ends when a best matching previous case has been found.

Feature Indexing

Broadly speaking, **the indexing problem** in CBR is the problem of deciding what cases are to be retrieved at appropriate times while guaranteeing that those retrieved cases are applicable to a new problem. Feature indexing involves assigning indices to cases to facilitate their retrieval. The indices associated with a case are combination of its important descriptors or features, which distinguish it from others. When assigning indices to a case, we need to observe the following guidelines[24, 49].

- Indexing should represent the concept that is normally used by the reasoner to describe the content in a case.
- Indexing should address the purpose for which a case will be used.
- Indexing should be predictive on the usefulness of a case
- Indexing should anticipate the increase of a case base.
- Indexing should be concrete enough to be recognized in the future.

After the indices for all the cases in a case base have been decided, strategies such as attaching importance values to indices have been used in order to assess the similarity between an input problem and the previous cases. Usually, this importance value is measured by what we call the *feature weight.* For example, in PROTOS[39], each feature in a stored case has been assigned a degree of importance[3]. In CREEK[2], a similar mechanism is adopted, which stores both the predictive strength(discriminatory value) of a feature with respect to a set of cases, and the feature's criticality, i.e. how the lack of the feature will affect the case solution.

The indexing problem is a central and very attractive problem in CBR. It is also the central topic in this thesis. It will be further discussed in the following chapters.

Retrieval

The case retrieval algorithm is at the heart of CBR. It is, according to the description of a new problem, responsible for retrieving the most similar cases from a case base using the indices of the cases. The indices and the organizational structure of the case base will guide a retrieval algorithm to search for the potential useful cases.

A number of strategies have been proposed in the CBR literature for the case retrieval algorithms. Some of them are serial, while others are parallel; Some use flat-indexed structures, but others use hierarchical-indexed structures. Some use indices to construct structures that distinguish cases from each other at a very small granularity while others discriminate cases at a more coarse level. Each method has its advantages and disadvantages. For example, for a flat case base the best case will be surely retrieved, but if the case base is very large, the search would be very time-consuming. On the other hand, if a case base is organized into a tree structure, the search cost is relatively small, but there is no guarantee that the most similar cases will be retrieved. There is always a trade-off between these two extremes.

Among those well-known and most discussed retrieval methods are nearest neighbor algorithm, and induction tree.

- *Nearest Neighbor Algorithm* This method involves the computation of the similarity between an input case and the previous cases based on a weighted sum of the features' similarity. The problems in this approach are how to determine the underlying similarity function, and how to determine the feature weights. We will discuss this

approach in more detail in Chapter 2. The search time for this method will be increasing linearly with the size of a case base. Therefore it will be more effective when a case base is relatively small. CASEY, a research project mentioned above, employs this method to index the cases by their surface features as well as the internal states, which are part of their diagnoses.

- *Induction Tree* Quinlan's ID3 and its successor C4.5[36] use information gain algorithm to determine which features are more important in discriminating cases, thus forming a decision tree with each branch representing a feature-value pair and each level representing a feature. The more important the feature is, the higher level it will be. The biggest problem in this method is how to deal with the missing or undetermined feature values. Typically this method is most efficient when there are relatively small number of features dominating a case base. ReMind and CBR Express in CBR2 in Table 1.2 use inductive decision trees in their retrieval process.

There are also some other retrieval methods, such as knowledge guided induction, template retrieval, etc.[3].

1.4.3 Case Reuse

There are two main approaches to reusing the previous cases[3, 24, 20, 12].

- Reuse the previous problem solution(which is called the *transformational reuse*).
- Reuse the previous method that derived the solution(which is called the *derivational reuse*).

In the first technique, the retrieved solution will not be applied to a new case. Rather, some transformation operator will be applied to it first, transforming it into a solution to the new case. The transformational reuse does not look at how a problem was solved but focuses on the equivalence of the solutions. Usually the transformational operators are highly domain-dependent, and a control facility to manipulate them is required. CASEY is such an application of the transformational reuse, in which a new explanation is built from an old explanation by rules. In each rule, the condition part indexes the differences between the new and old explanations, while the action part represents a transformation operator.

In contrast to the transformational reuse, the derivational reuse looks at how a previous problem was solved. The retrieved case maintains the information about the method used for solving the retrieved problem, including a justification of the operators used, the subgoals considered, the alternatives generated, the failed paths, and so on. The derivational reuse will apply this method to a new problem in a new context. Generally, the problem-solving systems using this reuse technique are for planning problems. An example of the derivational reuse is the Analog/Prodigy System[20], which reuses previous plans designated by the commonalities of the goals and the initial situations, and resumes a means-ends planning if the retrieved plan fails or is not found.

1.4.4 Case Revision

The case revision consists of two tasks. One is to evaluate the problem solution generated by the case-reuse process. The other is to repair the problem solution using domain-dependent knowledge. To evaluate a problem solution, one needs to apply it to a problem in the real world. The result from this application will be fed back for case revision. CHEF[24] is such a CBR system, in which a case(cooking recipe) is applied to an internal model which is assumed to produce feedbacks for its revision. Case revision involves detecting differences between the current case and retrieved explanations for it. For example, in CHEF, some causal knowledge is used to generate an explanation of why certain goals of a retrieved recipe were not attained. To revise a case, the failure explanations will be used to modify the case in such a way that the failure does not occur. Again in CHEF, the failed plan is modified by a revision module which will add steps to the plan. This revision will guarantee that the causes of the errors will not occur.

1.4.5 Case Retainment

This is the process that incorporates the useful knowledge from a new problem solving process into an existing case base. The reasoner will learn from the success or failure of the retrieved solution, which is triggered by the outcome of the evaluation and possible revisions. If a new problem is solved by the use of a previous case, a new case might be built or the old case might be generalized to subsume the new problem. If the problem is solved by other methods, including getting feedbacks from a user, then an entirely new case will have to be created. In any circumstance, a decision needs to be made about what to use

for the learning process, such as relevant problem description, problem solutions, and even an explanation or justification as to why a solution is appropriate to the current problem. Another source of learning is the problem-solving method or the reasoning path. Failures in the reasoning may also be a source of learning. When a failure is encountered, the system can then get a previous similar failed case, and use it to improve its understanding of the present failure.

1.4.6 Integrated Approaches

In most CBR systems, general domain knowledge is represented by cases. However, the representation and the use of that domain knowledge might need more than CBR itself. It involves integration of CBR and other problem solving methods. Thus, the overall architecture of a CBR system is to determine the interactions and control strategies between CBR and other components[3]. For instance, CASEY integrates a model-based causal reasoning program with CBR to diagnose heart diseases. The whole process is as follows. When the CBR module fails to provide a desired solution, CASEY will start a model-based module to solve the new problem and stores the solution as a new case for future reuse. Since the model-based model is complex and time-consuming, CBR in this integrated framework is essentially a possible way to achieve speed-up learning.

Another such system is the CaseAdvisor™ system developed by the CBR group at Simon Fraser University(SFU). The core of the system is the paradigm of CBR. However, many other features have been integrated into it to make it more attractive. Decision tree is a tree-like structure which can be shared by several cases. It will be used as a set of step-by-step instructions on how to solve a new problem. This feature is especially useful in the diagnosis domains. Rule base is another feature which is based on the observation that the description of a new problem sometimes implies some of its feature-value pairs, which are desirable to be selected automatically in the case retrieval process. In order to retrieve cases more accurately, some information retrieval algorithms are also employed in the system, such as *trigram* match[25]. Constraint-satisfaction algorithm is another attempt integrated into the system in order to solve the constraint problem among different feature-value pairs.

In this thesis, a machine learning method will be incorporated into this system with the hope that it would learn from a user's behavior, and simulate it in its own behavior.

The integration of CBR with other reasoning and learning methods is closely related to the general issue of the architecture for a unified problem-solving and learning approach,

which represents the current trend in the machine learning literature[3]. CBR itself is a combination of problem solving (during case retrieving and reusing) and learning (during case retaining). However, we can still introduce other learning methods into CBR in order to improve its performance? Definitely, this is an interesting, and also important dimension into CBR research.

1.5 Relationship of CBR to Other Methods

In the family of reasoning methods, the relationship between CBR and others will help us better understand CBR.

- *Memory-based reasoning*(MBR) is often regarded as subsumed by CBR. MBR solves a new problem by retrieving the previously solved problems as a start point. However, its primary goal of reasoning focuses on the retrieval process, in particular, on the use of parallel retrieval schemes to enable retrieval without conventional index selection.

- *Analogical reasoning* uses the same cognitive models as CBR. However, analogical reasoning originally concerns more about the abstract knowledge and structural similarity among cases, while CBR is more concerned about how to form the reasonable correspondence between a new case and the old cases based on the pragmatic considerations about the usefulness of the result. CBR does more than analogical reasoning does. Besides the research in the mapping in analogical reasoning, CBR also studies the related processes that are both before and after the mapping.

- *Database systems* are designed to make exact matching between the queries and the stored information, which is not always the case in CBR. Rather, CBR just retrieves the most similar cases, which might include conflicts with some of the features specified in the retrieval query. The possibility as to whether a case is retrieved not only depends on itself, but also on its competitors. Despite these differences, there are some aspects of the CBR and database technologies that can benefit from each other. For instance, as indicated in [26], to manage large case bases in CBR applications, it is advocated to use relational database management systems, combined with other mechanisms to allow flexible query specification and partial matching during the case retrieval.

CBR involves not only the reasoning process, but also the learning process. In contrast to other learning methods, we can also see that CBR is of some advantage.

- *Inductive learning:* Unlike traditional symbolic and other approaches to inductive learning, which define concepts by generalizations on the exemplars, CBR systems define concepts entirely as specific cases. This brings many advantages[26]. It enables system to use a concrete case to support its decisions. The system gives user a chance to verify a case's applicability. It is also useful to solve conflicts. Finally it enables incremental learning because a CBR system can be seeded with a set of seed cases, the coverage of a case base can be enlarged by adding new cases incrementally if such an addition is proved to be needed.

- *Explanation-based generalization:* In explanation-based generalization, rules are used to explain why a training example has some particular properties, and the explanations are used to guide the generalizations. The generalizations are stored for future use. In contrast, CBR generalizes cases at storage time. However, it adapts them when such an adaptation is needed to solve a new problem. Another difference is that, in addition to the generalization, more types of operations will be executed in the adaptation process in CBR, such as specialization, substitution, and possible modification.

1.6 Case-Base Maintenance Problem

The problem of maintaining a case base over time is called *case-base maintenance problem.* When constructing a case base, there are two perspectives we need to consider. The first one is that the case base itself is changing, with the addition of new cases, the deletion of old cases, and the modification of existing cases. While these actions enhance the power of the case base gradually, they also bring some information which is undesirable to the already existing cases. Inconsistent and redundant cases pose a problem to the growth and expansion of a case base. A strategy is proposed in [38] to deal with this kind of case-base maintenance problem.

In this thesis, we examine the case-base maintenance problem along another direction. In our experience with practical applications of CBR technology, three serious problems arise with regard to the feature weights. First, the feature weights are assigned manually by humans, not only making them highly informal and inaccurate, but also involving intensive

labor. Second, a CBR system with a static set of feature weights cannot cater to a specific user. It would be desirable to enable the system to acquire a user's preference automatically. Finally, a CBR system often functions in a changing environment, either due to the nature of the problems it is trying to solve, or due to the shifting needs of its user. We wish to have a CBR system that always adapts to a user's changing preference in time. These three problems comprise one of the core tasks of case-base maintenance problem. Maintaining an updated set of feature weights over time is crucial in the use of a CBR system. This thesis will present an integrated CBR framework which will attempt to solve these problems

1.7 Summary

We hope that this chapter would introduce the background knowledge in CBR to the readers. The possible and potential research areas in CBR are also discussed.

Currently more and more CBR systems come out from the laboratory and are commercialized into the industry world. Because of its merits, many industrial practitioners are turning to CBR to automate various aspects of their tasks, such as help desk, customer service, quality control, equipment maintenance, and so on.

Chapter 2

Feature Indexing and Weighting

2.1 Introduction

We have shown in Chapter 1 that there are mainly three pieces of information in a case. They are *problem description*, *problem solution*, and *outcome*. If we consider them as the lesson a case teaches, then we need to decide within what context such a lesson will be useful. We call this context a case's indices. We have mentioned the indexing problem in Chapter 1. We will discuss it in more detail in this chapter.

2.1.1 A Small Example of Case Base

Throughout the rest of this thesis, we will use a small case base to study some simple examples to get a better understanding of some key concepts in our explanations. However, when necessary, some other examples will also be shown complementarily to better address the situation in question.

The small case base is adapted from [50] and is shown in Table 2.1. Each row of the table represents the loan information about a person. In the table, the column *Loan status* represents the situation whether a bank makes a profit or loses money, which is divided into four groups. They are *very good*(the bank makes a big profit), *good*(the bank makes a profit), *bad*(the bank loses some money) and *very bad*(the bank loses a lot of money). Column 2 represents a person's monthly income. The column *Job status* stands for whether the income of a person is paid at an hourly rate(i.e. waged) or paid annually(i.e.salaried). The fourth column represents a person's monthly repayment for her/his loan.

	Loan status	Monthly income	Job status	Monthly repayment
Case 1	good	42 units	salaried	2 units
Case 2	very bad	40 units	salaried	6 units
Case 3	very good	30 units	waged	3 units
Case 4	bad	18 units	salaried	4 units

Table 2.1: A Small Case Base of Four Loan Cases(1 unit = \$100)

We will attach concrete meanings to these four columns when we use them in explaining different concepts.

2.1.2 k-NN Algorithm

Lying at the core of CBR is the case retrieval process. k-NN is one of the frequently used algorithms to retrieve most similar cases in a case base(Also see discussions in Chapter 1 for other retrieval algorithms).

Basically, k-NN assumes that each case in a case base is defined or indexed by a set of n(numeric or symbolic) features. Given an input query Q, k-NN retrieves a set of k cases most similar to Q(i.e. these k cases have the least distance from Q) from the case base. For a case C and a query Q, the following formula is used to compute the distance between them[51].

$$distance(C,Q) = (\sum_{i=1}^{n}(W_i * difference(f_i^C, f_i^Q)^2))^{1/2} \tag{2.1}$$

where W_i is the weight assigned for feature f_i, $difference$ is the similarity function for different values of a feature, and f_i^C and f_i^Q are the values of feature f_i in case C and query Q, respectively.

The larger the distance between two cases is, the smaller the similarity between them is. On the contrary, the smaller the distance is, the larger the similarity is.

There are different representations of difference function $difference$ in Formula 2.1. Formula 2.2 is one of them[51].

$$difference(x,y) = \begin{cases} |x-y| & \text{if feature } f \text{ is numeric} \\ 1 & \text{if feature } f \text{ is symbolic and } x \neq y \\ 0 & \text{if feature } f \text{ is symbolic and } x = y \\ 0.5 & \text{if } x \text{ or } y \text{ is unknown} \end{cases} \tag{2.2}$$

where x, y are two values of feature f.

In some implementation of Formula 2.1, all features use the same weights. This will allow redundant, irrelevant, and other imperfect features to skew the similarity computation. Hence, many variants have been proposed that the more relevant features in a case base be assigned higher weights in order to show their higher relevancy.

After the similarity to the input query is computed for each case, all the cases in a case base are scored. Cases with higher scores are ranked higher than those with lower scores. It can be easily seen from Formula 2.1 that a case whose description is more similar to a query's is scored higher than that whose description is less similar.

In the following discussions, we will use the formulas shown here to do some simple calculations on Table 2.1.

2.1.3 Features in CBR

People use CBR with the hope that their previous experiences would be *efficiently* and *accurately* recalled. Although other steps in CBR cycle are also very important(see Figure 1.1), the case retrieval step bridges the gap between the past and the present. Its efficiency and accuracy have a direct influence on the overall performance of a CBR system. As the indexing problem aims to solve the efficiency and accuracy problem in CBR, it is not a surprise to see that it is regarded as the central and very active research area in the CBR literature[3].

In practical implementation, the indexing problem has been addressed as a problem of how to assign indices to the cases in a case base. In a case the indices might be the combinations of features that are responsible for its failure or success, which can be used to help a reasoner avoid the errors that happened in the past or help suggest a solution to a new problem.

For example, in the case base shown in Table 2.1, the four columns of the table can be used to represent four different features. Based on the experiences with the loan transactions, the domain experts would like to choose two features monthly income and monthly

repayment to form the indices to the case base, since they satisfy the above discussions as well as the guidelines discussed in Chapter 1 for choosing a set of good indices.

For a particular case, the set of its features interprets a situation, in which someone might remember that case. Some of the features are just directly derived from a case's problem description and solution, while some are inferred from the information that is more abstract. For example, if we just extract features from problem description and solution, there are few common points between basketball and chess. However, if we go to a higher level, we can find these two games still share some abstract features, such as that both games are competitive, each side attempts to set up a situation which is of disadvantage to the other side, and so on. If a chess expert just notices the surface features about basketball, such as two teams, a ball, a field of certain size, and etc., it is almost impossible for her or him to see the similarity between the two games. Thus, the retrieval and matching processes in CBR need do their work based on more than just superficial features.

A case's indices will tell us under what appropriate circumstance to retrieve it. The stored previous cases in a case base are indexed by these indices. A CBR system will then use a new problem as the target indices, and search for the most similar cases, whose indices match those target ones most.

How to choose a set of good indices for a case is very important in CBR. Researches on indexing over past 10 years have shown[24] that the good indices in a case base are features or combinations of features which will distinguish a case from others. The predictiveness of indices is the most concerned aspect for choosing them(Also see discussions in Chapter 1).

2.1.4 Weights in CBR

The indexing problem is scattered into the whole case retrieval algorithm, including the processes of searching, matching, and ranking.

Of these processes, we are especially interested in matching and ranking, which are related to this thesis. They represent a best-case-selection process. The searching process searches in a case base for the most similar cases to a new problem. It requests the matching process to compute the similarity between the problem and a case using certain features represented by the case's indices. At this point, a more comprehensive evaluation of similarity will be executed. This evaluation will take into account the weights of the matching features. According to such a series of feature matchings, searching process collects a set of cases that match the new problem most. These cases will be scored according to their

similarity computed.

We will use Formulas 2.1 and 2.2 to do some simple calculations on the unweighted and weighted similarity between an incoming case and an old case. We still use the small case base as shown in Table 2.1. Given a target T of a person's loan with the monthly income of 30 units(1 unit = \$100), and monthly repayment of 2 units, if we do not use the weight in the computations of the distance between cases, the distance score of Case 1 with regard to this target will be $((42-30)^2+(2-2)^2)^{1/2} = 12$, and the distance score of Case 2 will be $((40-30)^2+(6-2)^2)^{1/2} \simeq 10.8$. From this computation, we can see that Case 2 is more similar to the target T. However, if the domain experts do not think the features monthly income and monthly repayment are equally important in the similarity computation, and may assign a weight of 1 to feature monthly income, and a weight of 4 to the feature monthly repayment. Then the distance score of Case 1 will be $((42-30)^2\times 1+(2-2)^2\times 4)^{1/2} = 12$, but now the distance score of Case 2 will be $((40-30)^2\times 1+(6-2)^2\times 4)^{1/2} \simeq 12.8$. This time the result is totally different; Case 1 is closer to the target T. Thus from this example, we can see how important the role of the feature weights is in the case retrieval process. With different weight assigned to the cases in a case base, the case retrieval result might be different.

The weight of each feature designates the purpose for which a retrieved case will be used. The ranking process will choose those cases that best address a reasoner's purpose. How to assign an appropriate weight to a feature is the *feature-weighting problem.* A function used to compute the similarity between cases can only be as good as the knowledge it has of the weights of the features of a case[24]. The weight associated with each feature tells us how much attention needs to be paid to the matching or unmatching of that feature.

Global vs. Local Weight Assignment

As one dimension to consider the assignment of feature weights, they can be assigned globally, over a large set of cases or over the whole case base, or they can be assigned locally, over a small set of cases or individual cases.

For the global assignment, in order to determine which features of a case tend to be important for a match, a thorough analysis of all the cases in a case base is required. The domain experts will employ their knowledge and experiences to decide which features are more important, and which are less. Usually the assignment work is done during the construction of a case base. Another way to assign global weights is to do a statistical

evaluation of a known number of cases to determine which features predict different outcomes and/or solutions best. Those that are good predictors are then assigned relatively higher weights. A single static value of weight for each feature works well enough for a case base when there is little or no variation in the importance of a feature across different problems.

However, this situation does not hold for every domain. For tasks in some domains, the feature weights vary locally with the context of the matching process. Consider a baseball club boss uses a case base of baseball players' information to determine a fair salary for a new player. When the boss considers the similarity between a new player and an old one, the feature weights of individual players are different depending on what position the new player will play on the field. For instance, when the boss tries to determine the salary for a pitcher, the feature 'batting average' will be having lower weight than that when determining the salary for a fielder, which means that 'batting average' is a more important feature for fielder than for pitcher. This suggests that while the global weight assignment works for the domains with little or no variations in the importance of features, the local weight assignment puts the feature weights into a changing context, thus having more convincing and practical significance.

Static vs. Dynamic Weight Assignment

Along another dimension of the weight assignment, they can be assigned statically, or they can be computed dynamically. In the dynamic assignment, it is required to take into consideration the influence from the context in which the matching is happening. When a case base is used only for one purpose and the relative weights of different features are consistent across the cases, assigning one set of weights to all the features will be possible. However, when the cases in a case base are to be used for several different purposes, each of which requires different focus on the different characteristics of the cases, several different sets of weights might be required. Actually, when a case base serves for different purposes, different matching criteria may be necessary. Each of those criteria is associated with the conditions stating a circumstance it will be used. Which weights should be used or not used in the computation is dependent on the current context.

Our Views on Weight Assignment

Our view of the dynamic nature of the feature weights is based on a broader sense than above, much more motivated from the practical application needs. We have considered the situation mentioned above. However, we go further, making a dynamic context engulf more meanings.

- First, we think that a CBR system is dynamic, in which different users with different purposes may have different views on the cases in a case base. Therefore, they would like to have different feature weights for a case. In addition to the example of a case base of football players which shows that a user may use a case base under different contexts. We consider another example. For a case base of human resource data, different information is stored about individuals, such as her/his height, weight, sighting ability, running speed, health situation, what skills s/he has, what education degree s/he is holding, and so on. A basketball player recruiter may weigh the feature *height* more importantly than others when choosing a player, while a pilot recruiter may weigh the feature *sighting ability* more importantly. Our example shows that the dynamic context also includes the situation that the users may be different for the same case base. Different users, even in the same community, may have different views on the individual cases, which need be reflected in the feature weights in the similarity computation.

- Second, we think that a CBR system is not only a dynamic system, but also a behavior-learning system. If we consider the most similar cases produced by the retrieval process in a CBR system are the result of a CBR system's behavior, then we greatly hope this behavior would simulate a user's behavior to the greatest extent, thus making it possible to produce the most desired cases. From this viewpoint, we imagine that a CBR system with the learning ability could learn from a user's behavior, and adjusting its own behavior accordingly. The result of the learning is that a CBR system's behavior gradually approximate a user's behavior. As identified in [24], there is a tension between using indices to designate usefulness and direct a search and allowing them not to overly restrict what can be retrieved. Behind this tension is the problem of how to accurately decide the feature weights in the case retrieval process. Our view that a CBR is a behavior-learning system helps suggest a solution to it. Instead of the current weight assignment practice which is relatively static, we regard the process of

weight assignment as a dynamic process that never stops. It keeps alive with the work sessions of a CBR system. Our desired accuracy and non-overrestriction of indices will be achieved iteratively through its learning.

When we human beings reason, we always attach different importance to different factors[9] in the searching of final goals, reflecting our own behaviors. When we use CBR, we also want to incorporate this phenomenon into it. A CBR system's behavior is encoded into the weights assigned to the features. Weights, in turn, when used in the similarity computation for the cases in a case base, reflect the system's behavior. The comparisons between a CBR system's behavior and a user's desired behavior will present the differences between them, guiding the behavior-learning process as what to be done in the feature weights.

- Third, we think a CBR system is not only a dynamic system, a behavior-learning system, but further an evolutionary system. We now put a CBR system into a more dynamic context. Within a short-time period, there may be no apparent changes in a user's behavior. However, a user's behavior might be changing with time. For the same problem, s/he might have different views at different times. Thus in the long run, the system needs to have an ability to capture regularities hidden in a user's behavior to accommodate the possible changes. Such a system is a responsive system which keeps its pace with the changes in its environment and reflects them in its own behavior. For example, in a printer diagnosis application using CBR, at first we may consider that the paper jamming is the biggest factor when the problem that paper can not get out occurs, and assign a relatively higher weight to its corresponding feature. However, with the improvement of paper quality, we might change our view. We lower down its weight. Furthermore, with the improvement of mechanical technology, the paper jamming will become less likely a dominating factor in the problem. Thus we want to reduce this weight to a relatively small value. These changes need be reflected in the system, along with the time passing, to make their corresponding effects gradually appear in the following work sessions.

The above three views are in accordance with the tasks with regard to the case-base maintenance problem discussed in Chapter 1. For a general picture, a case-based reasoner is a living system, which can fine-tune its weights according to its user's behavior, respond to its changing environment, and provide its different users with different set of weights.

Before we go on to the specialized discussions in this thesis, we first examine what have done in the same direction for these tasks. This will help understand what we will do.

2.2 Related Work

A CBR system in our imagination is a self-adjusting system. It will monitor a user's behavior, take the user's feedbacks, and adjust itself to reflect them in its own behavior. It can be easily seen that this process, in fact, forms a self loop. With more and more folds of the loop, the system's behavior tends to approximate its user's.

We project our system into a larger background. In general, as more and more AI systems are being used to address the real world applications, how to acquire knowledge and how to take advantage of this pool of knowledge become a very difficult task. This is due to the high degree of variety and complexity of the real world situations. In addition, the fast-changing outside world makes this task even more difficult. Thus, the knowledge acquisition has long been identified as a main problem in constructing a knowledge base. Obviously, CBR provides a relatively flexible approach to deal with this problem(see [24, 3, 26], and our discussions in Chapter 1). However, even with a well-constructed knowledge base, the reasoner can still encounter reasoning failures. The reason for this is due to the inability for the reasoner to properly access[1] and apply its knowledge, and also due to the fact that the once-updated knowledge may become outdated. Recently more and more researchers in AI[1] are becoming interested in how to monitor the performance of the reasoner, compare its current behavior with a user's desired one, and seek the opportunity to learn from this information in order to improve the quality when the knowledge is used next time.

2.2.1 Learning Agents

In [42], it is proposed that *the problem of AI is to describe and build components that reduce the stupidity of the system in which they function.* To interpret this, a goal of post-modern AI should be the improvement of how a host system functions through the development of intelligent parts to it. It is suggested the intelligent parts in a big system be called *intelligent components* rather than *intelligent agents.* Although whether or not this will be widely accepted in AI is not sure, the paper does demonstrate an important direction in AI - the integration of intelligent parts with the host system. Recognizing the limitations and taking advantages of the capabilities of the host system is the secret of the building such an

intelligent component that is robust and successful. The task of the intelligent component is to help the host system choose a response good enough to meet the system's needs in a timely fashion. We can see that, although this paper[42] mainly discusses the integration of the intelligent components with the host system, it does give us some suggestion. Can we do something similar in CBR?

Maes[29] argues more about the feasibility and application needs from the practical fields for such integrations. It is pointed out that the technological developments are not in line with a change in the way people interact with computers. Thus there is a need to change the current dominant metaphor or direct manipulation, which requires the user to initiate all tasks explicitly and to monitor all events. In order for an untrained user to make effective use of the computers of tomorrow, it is suggested that techniques from AI, in particular so-called 'autonomous agents', can be used to implement a complementary style of interaction, which has been referred to as indirect manipulation. A user is engaged in a cooperative process in which s/he and the computer agents both initiate communications, monitor events, and perform tasks. Besides these, the paper argues that two basic problems need to be solved when constructing such an agent. They are *competence* and *trust*. The paper disqualifies two previous approaches to constructing an intelligent agent in the light of these two basic problems. Instead, it proposes an alternative approach that relies on machine learning techniques. It presents a hypothesis which is that under certain conditions, an intelligent agent can 'program itself'(i.e., it can acquire knowledge it needs to assist its user). The agent is given minimum amount of background knowledge, and it learns appropriate 'behavior' from its users. It is shown that in order to fulfill this task, there are several preconditions. One is that the use of the application has to involve substantial amount of repetitive behavior(within the actions of one user or among users). Another one is that the repetitive behavior is potentially different for different users. It is believed that such an intelligent agent has several advantages. It requires less work from the ordinary users, and over time, it can easily become customized to individual and organizational preferences and habits.

The paper demonstrates several examples of such intelligent agents. One is Maxim[53] which assists a user with the daily incoming emails. It can learn to prioritize, delete, forward, sort and archive mails on behalf of its user. Interestingly, this agent is very generic, and when attached to a meeting scheduling software package, it becomes an assistant to a user for scheduling of meetings(such as acceptance/rejection, scheduling, rescheduling, negotiating

meeting times, etc.). NewT[45] is another agent which helps a user filter Usernet News. Once trained, it can recommend articles to its user. The user can give positive or negative feedbacks for articles or portions of articles recommended. The last example of application is the kind claimed by the user as the next 'killer application'. They are entertainment selection agents. They will help a user select movies, books, television and radio programs based on her/his personal tastes. One such system is described in [16], which can be used for recommending science books.

Although this paper is not highly related to CBR, it shows us a potential that introducing a learning component into CBR is not only worth research efforts but also has its practical significance. The motivations shown in [29] also hold in the practical use of a CBR system. Furthermore, the preconditions for building such intelligent agents are also true in CBR. Then why do we not borrow the general idea there and apply it to our work?

While [29] suggests a whole picture of how and under what conditions an intelligent agent acts in a modern autonomous system, it does not go further to discuss about how such an agent can be implemented, such as what the learning policies are, how to determine the learning rate, and etc.. To this end, we go back to CBR and check what have been done along the same direction.

2.2.2 Case-Based Learning

Aha in [7] introduces a subset of CBR, which is called case-based learning(CBL). By the definition in that paper, CBL algorithms focus on the topic of learning. They will take a sequence of training cases and output a *concept description*, which can be used to make predictions of goal feature values for subsequently presented cases. In fact, CBL is an application of CBR. The main component of the concept description is a case base. But almost all CBLs maintain additional related information in order to make accurate predictions. CBL is different from ordinary CBR. It is limited to the feature-value case representation, but does not necessarily employ smart indexing schemes. It focuses on the learning process in CBR. Current CBL algorithms assume that cases are represented using feature-value pairs, where features are either *predictors* or *goal features*. Predictors are those which can be used to predict values for the goal features.

The paper discusses a family of CBL algorithms. The simplest one, CBL1, computes the similarity between two cases as the inverse of Euclidean distance for numeric features and uses a simple matching test for symbolic feature values. CBL1's prediction function is

k-NN algorithm, which predicts that the value for a given case's goal feature is the most frequent goal value among its k most similar stored cases in the concept description.

CBL2 is identical to CBL1 except that it retains only incorrectly classified cases in its concept description. The goal of such retainment is to avoid computing a needlessly large number of similarity assessments during the prediction attempts. CBL3 is further improved to deal with the noisy cases on the basis of CBL2. It keeps a track of the frequency for which a stored case, when selected as one of the current case's most similar stored cases, matches the current case's goal feature value. After that it uses a statistical test of significance to ensure that only the stored cases with significantly high frequency will participate in the final prediction of the goal feature values.

CBL4 is another CBL algorithm introduced in the paper. It mainly aims at the situation where the feature weights vary greatly among the predictor features, for instance, the irrelevant features exist. In CBL4 the feature weights are learned rather than told. It will be adjusted after each prediction attempt during the training process by comparing the current training case with its most similar stored cases. CBL4 initially assigns equal weight to each feature. It increases the weights for features whose values are similar when the correct predictions are made. Otherwise it decreases the feature weights. The adjustment delta is determined by the difference between a feature's values for the two cases' whose similarity is being assessed. This is a learning process. The performance of the prediction is evaluated, and based on the result of the evaluation, the weights of the relevant features are adjusted for the next use. Actually, CBL4 can tolerate irrelevant features quite well as shown in the paper. However, it is also pointed out there is a need for further research work on CBL4 as it does not perform well when the weights of features are context sensitive in the sense expressed in [10]. Here the experts recognize that the feature weights depend on the combinations of the other feature-value pairs currently describing a case. The paper suggests a new algorithm GCM-ISW, although not implemented, to deal with the situation. This new algorithm employs a separate set of feature weights for each (*case*, *goal feature*) pair. These newly introduced weights allow the algorithm to encode different weights for a feature depending on both the case and the feature whose value is to be predicted. The reader is referred to [7] for detailed discussions on this algorithm.

2.2.3 Feature Weighting

Aha in [9] argues that people represent categories not only with the exemplars from those categories, but also with a set of specific weights associated with each feature of the exemplars. The same result is claimed in [24] that the concepts are not represented simply by a single set of feature weights. Rather, a feature weight in similarity calculation depends on its context, i.e., the other features that are present for a particular exemplar. In addition, people can learn the feature weights in the concept learning process even when they have little guidance for doing so. Aha shows an experiment in order to confirm the above hypothesis as whether the human subjects can learn categories that require features to be weighted differently for different category exemplars. As expected, the hypothesis is true. The GCM-ISW, the last one suggested as above, but not yet implemented, now has been implemented in the experiment with the predicted result, which is that the selective attention process of a psychologically plausible learning algorithm must be a context-dependent function.

Wettschereck et al. in [51] survey a family of so-called feature weighting techniques along five dimensions. The goal of these techniques is to automatically assign weights to features using little or no domain-specific knowledge. We are especially interested in the first dimension called *feedback*. This dimension is related to whether or not the feature weighting method receives feedbacks from the k-NN variant algorithm being used. The survey discusses two kinds of such feedback methods. One is called *incremental hill-climber*, whose purpose is to modify the weights so that the modifications will increase the similarity between the cases in the same class but decrease the similarity between the cases in the different classes. There are some examples of this kind of feature-weighting method. They are EACH[44], IB4[6, 52], and RELIEF[23]. The adjustment policies in these examples are further discussed in the survey. As indicated, a drawback of these methods is that they process each training case only once, and are thus sensitive to the order of the presentation of the cases.

The second suite of feedback methods in the survey is called *continuous optimizers*. These feedback methods repeatedly update feature weights using randomly selected training cases. The update algorithm employed plays a crucial role in these methods. As shown in the survey, GA-WKNN[23] uses a genetic algorithm to update features' weights. Lowe[27] uses the function's gradient to increase the learning speed. Its purpose is to optimize the features'

weights using the conjugate gradient algorithm to minimize LOOCV(Leave-One-Out-Cross-Validation[51]) error on the training set. The derivative of this error for each feature weight is used to guide the search. The survey compares the feedback methods with the non-feedback methods. It claims four hypotheses, of which three are related to the feedback methods. These three hypotheses are: when the data are not carefully pre-processed, the non-feedback methods can perform poorly substantially, the feedback methods achieve higher accuracy than the non-feedback methods in the domains with a few interacting features, and the feedback methods enjoy faster learning rates.

For other comprehensive discussions on feature weighting methods, see [13].

2.2.4 Introspective Learning

Introspective learning is an approach to learning problem-solving knowledge by monitoring the run-time process of a reasoner[17, 14, 37, 1]. As indicated in [1], the need for introspective learning, and its increasing popularity have been across a range of AI problem-solving paradigms, from planning to case-based reasoning.

Actually, it can be found that the introspective learning method discussed here can be used to solve the problems in the feature weighting mentioned above.

In [17], Fox et al. describes their experiment with introspective learning in CBR. The ROBBIE system described is an application of an introspective self-model to the task of refining the indices used to retrieve cases. Its goal is to improve reasoning process when encountering failures in its reasoning. The introspective learning component in the system monitors its reasoning process by comparing it to a declarative model which is used to describe the system's ideal reasoning process. Once a failure is found, the model is used to create an explanation of the failure in terms of other failed assertion, and to suggest a repair. They report some experimental results on the system, and based on the results they claim that even under knowledge-poor initial conditions, the introspective learning process of new feature indices improves the success rate of the system over case learning alone. But they still indicate that there exists a problem with the ordering of the presentation of the problems. They also say that they are encouraged by the success of the ROBBIE system to expect the introspective learning to provide more advantages for CBR and elsewhere in the future.

Bonazno et al. in [1] demonstrates another system which combines introspective learning with CBR. They first pose the problem with their experiences in constructing a CBR system

for Air Traffic Control. The problem is that it is difficult to determine the important features and adjust their relative importance. The situation is further complicated by the fact that the features are highly context-sensitive. The predictiveness of a feature depends heavily on the current context. They use so-called 'pulling' and 'pushing' operations to adjust the feature weights. Given a target T and two cases A and B. If it is judged that A is a correct solution to T but B is not, the learning method will 'push' B away from T, and 'pull' A closer to T. As to its weight updating policy, their introspective learning method uses a decaying learning process. The following two formulas show this.

$$\text{increase: } W_i(t+1) = W_i(t) + \Delta_i \frac{F_c}{K_c}$$

$$\text{decrease: } W_i(t+1) = W_i(t) - \Delta i \frac{F_c}{K_c}$$

where K_c represents the number of times that a case has been correctly retrieved, F_c represents the number of times that a case has been incorrectly retrieved, and Δ_i determines the initial weight change. The ratio between F_c and K_c is used to reduce the influence of the weight update as the number of successful retrievals increases. Here, when to trigger the adjustment of the weights using the above two formulas is a crucial point, and will definitely influence the adjustment delta, affecting the retrieval quality afterwards. The paper does not discuss this. In addition, the paper does not consider that the 'push' and 'pull' operations are local to target T. Given another target S which is relevant to B, the adjustment of B, i.e. the 'push' or 'pull' operation, might make it closer to T, which is contradictory to the first action that pushes B away from T. The paper reports good result based on their empirical tests. It is claimed that the failure-oriented rather than the success-oriented learning contributes most to this result. They have also state that their learning method does not work well for pivotal cases(A pivotal case is one that provides coverage not provided by other cases in a case base[48]) as the redundancy in a case base is essential in such a learning process.

2.2.5 Other Learning Methods

If we consider the weight update policies[1] as the punishments or rewards, then we can regard the introspective learning as a kind of *reinforcement learning*[41]. In CBR literature, there are also some research work in the integration of reinforcement learning with CBR. The advantage of reinforcement learning[15] is that it requires only knowledge of its

present state and infrequent real values of reward to learn the actions necessary to bring the system into some desired goal state. Another advantage is that it learns even when the information available is very limited. Ricci et al.[40] introduce a new approach to compute nearest neighbor based on a local metric called AASM(asymmetric anistropic similarity metric), which is based on two assumptions. The first one(anisotropic) states that the case metric is defined locally: the space around a case in memory is measured using the metric attached to that case. The second one(asymmetric) states that the distance between two points in a continuous feature space F_i is not symmetric. In order to implement the second assumption, two different sets of weight are used for each feature. The model introduced in the paper employs a reinforcement learning procedure for adapting the local weights to the input space. It implements an anytime nearest k-NN algorithm, in which, given an input case c, if the nearest neighbor nn correctly classifies c, the distance between them will be decreased(rewards), while if the nn incorrectly classifies c, the distance will be increased(punishments). The paper presents a procedure that, starting from a case base C, and a set of weight w, iteratively changes the weights in w with the goal to improve the accuracy of the retrieval computed with respect to a given goal function G. In the paper, a *learning step* is defined as a mapping.

$$T : W \times C \longrightarrow W$$

One possible interpretation of this mapping is that $Weights + Cases = NewWeights$. The learning procedure in that method is an algorithm that iteratively chooses a case and calls the learning step on it until an exit condition is satisfied. The paper shows the detailed formulas for the learning step. However, the biggest arguable point is that it never proves that those formulas will be convergent, which is required to satisfy the final exit condition.

The paper claims that one advantage of the method is that AASM can be run as a black box without setting problem-specific parameters. Another advantage is that AASM can be assigned a fixed amount of memory and a fixed time to reply to a query. The system will perform better and better as the amount of memory and the time to reply increase. The paper also claims that a drawback of the model is that for each case it is needed to save two additional sets of weight for the input space. The experimental results are shown on the basis of the model to support their claims. It is pointed that the improvement in run-time performance and the reduction of the number of cases needed to obtain the same accuracy of nearest neighbor compensate that drawback.

In [22], Howe et al. claim the same as in [24] that some case bases use a set of weight globally which remains constant throughout the whole testing, while others contain features that vary in weights across the case space. In the latter situation, local weighting methods, in which feature weights can vary from case to case or feature value to feature value, can be applied in such domains. The locality of particular weighting algorithms can be visualized on a spectrum, from global methods that compute a single weight set for all cases, to extremely local methods that compute a different weight set for each case. The paper focuses on the intermedium of such spectrum. It presents a coarsely local feature weighting method, which allow weights to vary at the class level. This method is called *class distribution weighting*(CDW). The intuition behind this method is that although classes are certainly not always homogeneous, it is reasonable that for many domains the dominating features of a class are the same for most or all the member cases belonging to it. Instead of a single global set of weights, CDW computes a different weight set for each class in the set of training cases. The mathematical foundation for the computation is the statistical properties of the subset of a class in the training data set. The weights for a particular class on a given feature are based on the comparison between the distribution of the feature values for the cases in that class and the distribution of these values for cases in all classes. If the distributions are highly similar, the feature is considered useless for distinguishing that class from others, and thus it is assigned a lower weight. If the distributions are highly dissimilar, the feature is considered useful and will be assigned a higher weight.

The paper examines three variants of CDW. The first one is Global Mean CDW, in which the feature weights will be averaged across all classes to produce a single global weight set. It can be easily seen that this variant works well in domains where the relevant features are the same for all classes. The second one allows the weights to vary locally according to the feature values of the test case. Specifically, the feature set is expanded so that each case is described by a set of binary features corresponding to all the feature-value combinations in the original training set. Since the expanded data set uses a separate feature for each of original feature-value pair in the training cases, applying CDW to it will generate the weights that vary for individual feature-value pairs. This variant is called EF-CDW, which is identical to CDW for domains whose features all have binary values. EF-CDW has a finer locality than the standard CDW since it varies among individual feature-value pairs. The combination of GM-CDW and EF-CDW is Global Mean Expanded Feature CDW. The features will be transformed to expanded-feature format, the standard CDW algorithm will

then be applied to it, and the expanded-feature weights on the classes are averaged to get a global weight for each expanded feature. The paper claims that it should perform especially well on tasks where only certain feature-value pairs are relevant, but do not vary from class to class. The paper selects eleven tasks in order to experiment their claims. It is shown that at least one of the CDW variants significantly improves the performance for nine of the eleven tasks. Because there is no single technique appropriate for all the tasks, it is concluded that different tasks require different degree of locality in feature weighting. Basically the variants of CDW is based on the static situation of classification tasks. The foundation it is based on is the statistical information, which does not take into consideration the dynamic context as indicated in [24] and this chapter. It gives us some hint that the statistical information behind the distribution of the feature-value pairs across a case base does hide some useful knowledge which can be used to help us in the weight assignment.

For some other research results on the introspective learning methods, reinforcement learning methods, feature weighting methods, and the improvements on the efficiency and accuracy of the case retrieval algorithms in CBR, see [15, 14, 11, 31, 19, 8].

2.3 My Method

Case-base maintenance is a key task in CBR. How to assign the weight automatically and how to maintain an updated set of feature weights in response to the changes in a dynamic environment are the tasks of the case-base maintenance problem(See discussions in Chapter 1).

As discussed above, there are some useful static information hidden in a case base when it has been constructed. The distribution of the feature-value pairs within a case base is not uniform, reflecting different views on different pairs from the domain experts. What can we do with this pool of information? My method will employ statistical method to analyze the static distribution of the feature-value pairs within a case base, with the hope that the irregular distribution of the feature-value pairs would provide us some hint as how to assign weights to them.

The feature weights in a case base are not static from their nature, since they need to reflect the changing outside environment. Hence, only using the static adjusting method can not maintain a set of suitable and updated weights over time. In this thesis we will examine how to integrate CBR with another machine learning method in order to facilitate

its use in a changing context. Undoubtedly, a dynamic CBR system, which changes over time according to the changes in its environment, will not only further strengthen its original power as a reasoning and learning agent, but also definitely enhance its popularity in the real world applications.

2.4 Outline of Thesis

In this thesis, we will present the theoretical and experimental frameworks for a static weight adjusting method and a dynamic weight learning method. The underlying algorithms for these two feature-weighting methods will be discussed, and the relevant empirical test results will be demonstrated.

In Chapter 3, we will discuss the static weight adjusting method which explores the statistical information inherent in a case base to adjust the feature weights statically and globally. The mathematical foundation for such an adjusting method will be presented and justified.

In Chapter 4, a novel dynamic weight learning method will be integrated into CBR for feature weighting as a further improvement over the static adjusting method. We will discuss the practical motivations as well as the theoretical foundations for such an integration.

Chapter 5 will be focused on the system design and implementation of the two methods as presented in the previous chapters.

Empirical test results will be presented in Chapter 6 in order to evaluate the performance of the two feature-weights methods. Various practical considerations will be discussed in the light of accuracy and efficiency.

Finally, in Chapter 7, we will conclude the thesis with a summary of our work. We will also show and discuss some further work along the same direction.

Chapter 3

Static Weight Adjusting Method

3.1 Introduction

The domain experts construct a case base using their previous problem-solving experiences. In the process of the construction, they will extract and derive various surface and abstract characteristics[24] from the case base, which are often represented as feature-value pairs. They will assign a set of such feature-value pairs to the individual cases. In addition, they will also attach a feature weight to each of these pairs, expressing how important they think a pair will be when it participates in the computation of k-NN algorithm. The weight of each feature-value pair for a particular case is heavily dependent on the experts' previous experiences. It encodes the domain-specific knowledge. The feature-value pairs and their weights form a context in which a case will be retrieved.

An example using Table 2.1 on the influence of weights on the case retrieval result is shown in Section 2.1.4. In that example, the weight assigned to the feature monthly income is the same across all the possible values of that feature. The same applies to the feature monthly repayment. Here, we will use the weights which are constant at a more granular level - feature-value pairs. Each feature-value pair will be assigned a weight, and the weight will be the same for all its associated cases. But different feature-value pairs may have different weights, even if they share the same feature. For instance, not only the weight for feature-value pair $(repayment, 2units)$ may be not the same as that for feature-value pair $(monthlyincome, 50units)$, but also the weight for $(monthlyincome, 50units)$ may be different from the weight for $(monthlyincome, 40units)$. Therefore the weight is local to each feature-value pair. Obviously such a more granular weight assignment may produce

different case retrieval result from the weight assignment at the level of features.

In practice, for two cases C_1 and C_2, it is not necessary that the set of feature-value pairs associated with C_1 be exactly the same as that associated with C_2 - not only the number of the features are not necessarily the same, but the feature-value pairs for each case are not, either. Maybe a feature-value pair is associated with C_1 but not associated with C_2. This will provide a great flexibility since it does not impose the requirement that all the cases in a case base use a single standard set of features. This can be explained convincingly in some real situations. For instance, in the domain of computer sales, a new computer model has a new feature which is unique among all the other features. Undoubtedly, there is no need to assign this feature to all the other computer models. On the other hand, if a new problem, which requires to find some specific computer model, has a requirement that is corresponding to that unique feature, definitely that computer model should be retrieved with a relatively high rank. This is reasonable as currently only that model has that unique feature; clearly it will be the most potential candidate for further consideration. Although matching other features of that computer model may not produce a higher ranking score for it, matching this unique feature ought to make it rank higher than other models, or change its rank dramatically as compared to other models.

This is a useful phenomenon. In constructing a case base, if a feature-value pair is just assigned to one case, then we hope that it would have more importance when it participates in the similarity computation. We expand this small hint into a larger scenario. What will be if a feature-value pair is associated with two cases, or with three cases, and finally with all the cases in a case base. In that extreme situation, we do think this feature-value pair is less useful since matching it will contribute to the ranking scores of all the cases, making it difficult for us to distinguish which case is a potential candidate to be the goal we desire, while which is not.

Now we reach the very intuition on which our static feature-weighting method is based.

3.2 Formal Description of Static Adjusting Model

The intuition we have drawn is that the more cases a feature-value pair is associated with, the lower weight it is assigned, and conversely, the fewer cases, the higher weight.

After a case base has been constructed, there is some information hidden in the associations between the cases and the feature-value pairs. Such associations describe the

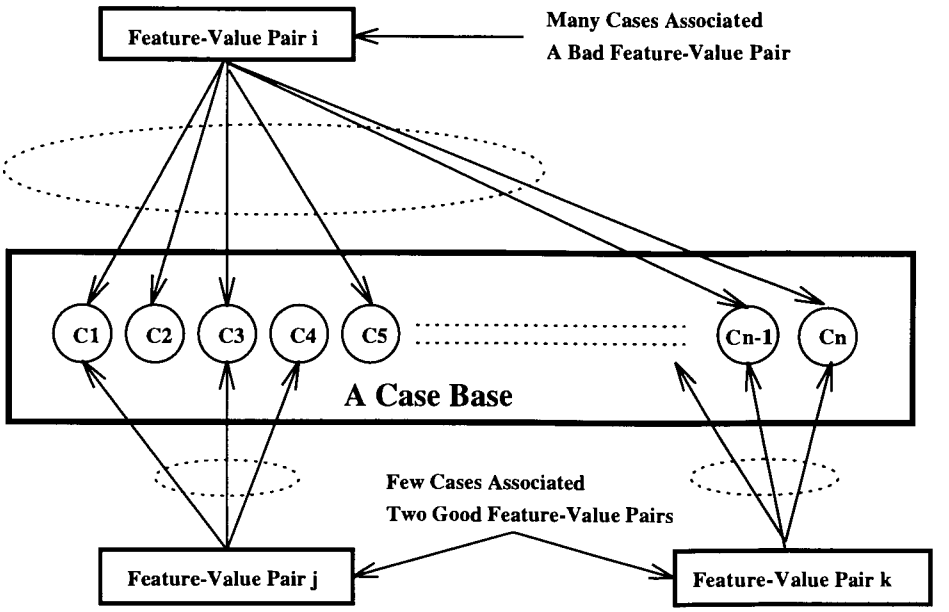

Figure 3.1: When is A Feature-Value Pair Good or Bad?

distribution of feature-value pairs within the case base. We will attempt to measure the usefulness of each feature-value pair by measuring its distribution across the whole case base. The weight for a given feature-value pair will be determined from the number of cases it is associated with versus the total number of cases in the case base.

We use Figure 3.1 to show the distribution of individual feature-value pairs within a case base. Feature-Value Pair i is associated with almost all the cases, such as C_1, C_2, C_3, C_5, $\cdots$, C_{n-1}, and C_n. We say it is less useful because it conveys little information about the cases we need. If it matches one of the feature-value pairs of a new problem, then all these cases will be changed with no one more distinguishable than others. On the contrary, *Feature-Value Pair j* is associated with only three cases C_1, C_3, and C_4. Matching it means that the number of candidate cases will be narrowed down greatly, making only a few distinguishable. Therefore, we consider it useful. The same argument applies to *Feature-Value Pair k*. As indicated in the figure, we call *Feature-Value Pair i*, which is associated with almost all the cases, a bad feature-value pair, while we call *Feature-Value Pair j* and *Feature-Value Pair k* good feature-value pairs, since they provide more information we want.

Let C represent a case base, in which there is a total of T_C cases. In C, each case is represented as C_1, C_2, C_3, $\cdots$, C_{T_C}. For this case base, there is a total of N features. For

each feature F_i, there are m_i values, where $i = 1, 2, \cdots, N$.

We use $V_{i,j}$ to represent the jth value of feature F_i, where $i = 1, 2, \cdots, N$ and $j = 1, 2, \cdots, m_i$. The total number of feature-value pairs in this case base is

$$T_{FV} = \sum_{i=1}^{N} m_i \tag{3.1}$$

For each feature F_i and its jth value, define

$$D_{i,j} = \sum_{k=1}^{T_C} \delta_{i,j,k} \tag{3.2}$$

where

$$\delta_{i,j,k} = \begin{cases} 1 & \text{if } V_{i,j} \text{ is attached to } C_k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Thus $D_{i,j}$ represents the statistical distribution information of individual feature-value pairs in this case base. Obviously, $0 \leq D_{i,j} \leq T_C$ holds. We also can see that we treat each $V_{i,j}$ equally, independent of the feature it is associated with. In fact, $V_{i,j}$ belongs to the expanded-feature set discussed in Chapter 2. If a case base contains N features defined by $F = \{F_1, F_2, \cdots, F_N\}$, and each feature F_i has a value set $V_{F_i} = \{V_{i,1}, V_{i,2}, \cdots, V_{i,m_i}\}$, then the expanded-feature set is $F^e = V_{F_1} \cup V_{F_2} \cup \cdots \cup V_{F_N}$.

We call Formulas 3.2 and 3.3 the information-collecting step, in which the statistical distribution information will be collected from a case base.

To find the raw weight for each $V_{i,j}$, we compare the number of its associated cases with the number of all the cases in a case base:

$$W_{i,j} = 1.0 - \frac{D_{i,j}}{T_C} \tag{3.4}$$

where $W_{i,j}$ represents the weight of feature-value pair $V_{i,j}$.

We call Formula 3.4 the weight-adjusting step. In this step, we assign the statistical information that each feature-value pair provides back to its weight.

If $V_{i,j}$ is not associated with any case, we will leave it alone, and do not compute its weight.

All these work can be done after a case base has been constructed and before its practical use to solve new problems. All the information it needs to do such an adjustment is static, which is hidden in the associations between the cases and the feature-value pairs in a case base. This is why we call it static adjusting method.

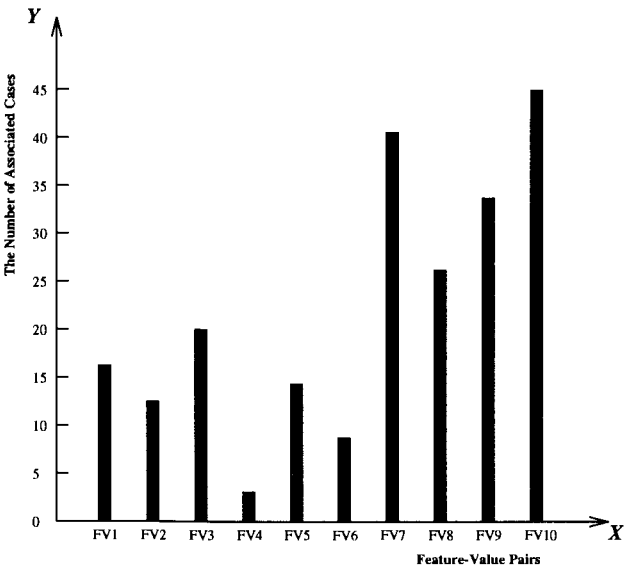

Figure 3.2: Distribution of Feature-Value Pairs

In this adjustment, we do not take into account the dynamic context in which a case will be retrieved. Neither do we consider the interactions between the individual features. In addition, once adjusted, the weight for each feature-value pair will be the same for all the cases it is associated with. Therefore, such a set of weights is *global.* As indicated in [24, 22] (also discussed in Chapter 2), the global weights work well for the domains in which the importance of each feature-vale pair is constant across the whole case base. They have little sensitivity to the problem context.

3.3 Algorithm for Static Adjusting Method

3.3.1 An Example

As an example, we create an artificial case base with 45 cases. We graph the feature-value pairs along with the number of their associated cases. For simplicity, we do not use $V_{i,j}$ to represent a feature-value pair. We just use FV_i to represent it since, as mentioned above, each feature-value pair is a member in the expanded-feature set F^e, and we can use just one subscript to represent it. Also for simplicity, we do not show all the feature-value pairs. We just show the first ten.

In Figure 3.2, the X-axis represents the feature-value pairs discretely, while the Y-axis

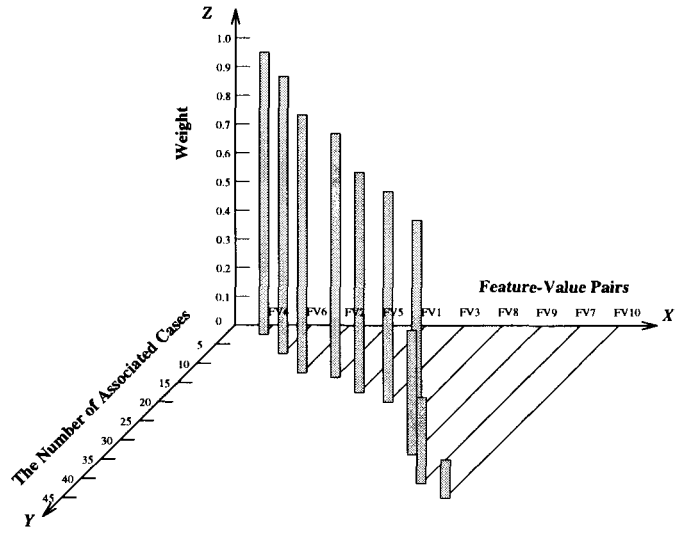

Figure 3.3: Weights of Feature-Value Pairs

represents the number of cases each feature-value pair is associated with. From the figure, it can be seen that FV_4 is associated with just 4 out of 45 cases, while FV_{10} is associated with all the cases. They occupy the two extremes. Other feature-value pairs take the places in between them. For instance, FV_5 is associated with 14 cases, and FV_7 is associated with 41 cases.

Our desired picture after we apply the static adjusting algorithm to this artificial case base is plotted in Figure 3.3. The X-axis and Y-axis represent the same as those in Figure 3.2, but the order of the feature-value pairs is re-arranged from the lowest to the highest in the number of their associated cases along the X-axis. Therefore FV_4 stands at the first position, the next is FV_6, and so on, until the last one FV_{10}. The Z-axis represents the weight adjusted for each of these feature-value pairs. The range of the weight is from 0 to 1.0. From the figure, we can see that because FV_4 is just associated with 4 cases, it is adjusted to have the highest weight of 0.95, while FV_{10}, the highest in the number of associated cases, has the lowest weight of only about 0.1.

3.3.2 Implemental Considerations

From Figures 3.2 and 3.3, we also notice that if we use Formula 3.4 to compute the weight for FV_{10}, it will be 0. However, in Figure 3.3, its weight is about 0.1. Although very low, it is not 0. The rationale behind this is that although this feature-value pair is considered

to be less useful, it still contributes a little information to the final ranking of the cases. It is better than nothing.

Therefore we reconsider the weight computation method represented in Formula 3.4 from an implementation angle. Using a more generic expression, Formula 3.4 can be rewritten as follows.

$$W_{i,j} \propto (\frac{D_{i,j}}{T_C})^{-1} \tag{3.5}$$

This formula states a property about the relationship between the weight of a feature-value pair and the number of the cases it is associated with. It is that the weight of a feature-value pair is the inverse of the result of the number of its associated cases versus the total number of cases in a case base. The realization of Formula 3.4 will depend on the concrete implementation and different applications as long as it observes the property stated as above.

Another consideration is that the size of a case base plays an important role in the weight computation. Obviously, in a case base of 100 cases, if a feature-value pair is associated with 10% of all the cases, then we can think this feature-value pair is relatively good since matching it only changes ten cases' ranks dramatically. However, if a case base has 1000 cases, then we would not like to think that 10% of all the cases is so good since matching it will be changing the scores of 100 cases, which still does not convey too much information about which case is more useful. If we use Formula 3.4 to compute the weights for these two situations, both weights will be 0.9. Obviously, we do not want this result in the case base that has 1000 cases. From this, we can see that for the case bases with different sizes, there are different standards to judge whether a feature-value pair is good or not.

In our implementation, we define six types of case bases according to their sizes. They are *micro* case base, *small* case base, *medium* case base, *large* case base, *huge* case base, and *very huge* case base with 21-100 cases, 101-200 cases, 201-500 cases, 501-1000 cases, 1001-2000 cases and 2001 or more cases, respectively. We also consider an extreme situation of the size of a case base. If the number of cases in a case base is below 21, then we think it is too small to have any statistical information inside with regard to the associations between the cases and the feature-value pairs. In this situation, we do nothing to the weights of the feature-value pairs.

In order to use different standards to judge the goodness of the feature-value pairs for different case-base sizes, we define five groups for each case-base type. They are *very good*, *good*, *average*, *bad*, and *very bad*. We preset both the minimum and maximum percentage

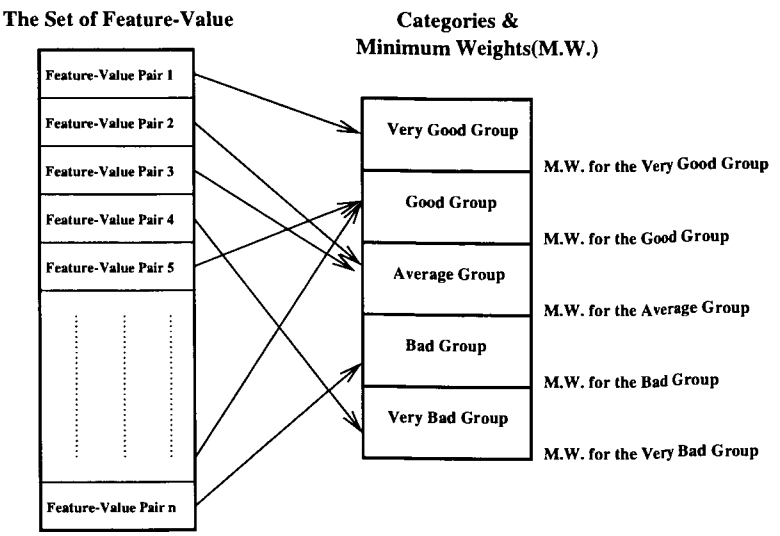

Figure 3.4: Mapping from Feature-Value Pairs to Groups

ranges of cases for each group. For instance, in a micro case base of 80 cases, a feature-value pair associated with 11%-20% of all the cases is in the good group. However, in a medium case base of 400 cases, a feature-value pair associated with 11%-20% of all the cases might belong to the average group. Thus, all the standards to judge whether a feature-value pair is very good, good, average, bad, or very bad in a case base will be relative to the size of the case base.

In addition, we also preset five minimum weights for the five groups in each case-base type. Any feature-value pair falling into a particular group should have at least the minimum weight for that group. This will map the weights of all the feature-value pairs into five distinct value sets, which will distinguish the feature-value pairs belonging to different groups.

All these parameters are preset in the system, and the domain experts can, based on their options, change them.

3.3.3 General Algorithm

The static weight adjusting algorithm can be considered as a mapping shown in Figure 3.4. For a feature-value pair, the weight adjustment algorithm will map it into one of the five groups according to the number of cases it is associated with, and then assigns it a weight which is equal or more than the minimum weight corresponding to that group.

The static adjustment algorithm is described as follows:

1. *According to the number of cases T_C, decide the minimum and maximum percentage ranges of cases for each group; convert these percentage ranges into the actual number of cases*

2. *For each feature-value pair*

   Collect in $D_{i,j}$ the number of cases it is associated with, where i and j represent the jth value of ith feature;

   According to $D_{i,j}$, decide to which group this feature-value pair belongs;

   Store in MIN_{num} the minimum range of the group represented by the number of cases;

   Store in MAX_{num} the maximum range of the group represented by the number of cases;

   Store in W_m the minimum weight of the group;

   Store in W'_m the minimum weight of the next higher group (For instance, if the current group is average, then the next higher group is good. If the current group is very good, then store 1.0 to W'_m);

   Assign the weight to this feature-value pair using the following formula.

$$W_{i,j} = W_m + \frac{(MAX_{num} - D_{i,j}) * (W'_m - W_m)}{MAX_{num} - MIN_{num}} \tag{3.6}$$

Formula 3.6 shows the desire that we want to give the domain experts a chance to show their own views on how to judge a feature-value pair in different case-base types. The domain experts may have different views when they assign feature weights. Thus they would like to use different standards to judge whether a feature-value pair is good or not. Also they may want to distinguish the feature-value pairs, which are judged good, from other pairs to a great extent. In Formula 3.6, the weight of a feature-value pair depends not only on the group it belongs, but also on the minimum weight specified for that group. We hope that once adjusted, the weights of the feature-value pairs in a case base would distribute into different value sets. Although the difference among the feature-value pairs belonging to the same group is not so apparent, the pairs belonging to different categories will have weights quite different from each other. This is helpful when searching similar cases, since

matching a good feature-value pair is quite different from unmatching it, or quite different from matching the pairs which are not judged good.

We will show the test results of this algorithm in Chapter 6. We will see that Formula 3.6 works well for our testing data. However, we do not rule out other alternatives for this adjustment computation. We think that the concrete implementation considerations and empirical experiments are essential in choosing such an alternative.

3.3.4 Computational Analysis of Algorithm

It can be seen that for each feature-value pair in a case base, we need to check each case to see whether it is associated with the feature-value pair. If we use N to represent the number of cases in a case base, and M to represent the total number of feature-value pairs, then we need at most $O(N * M)$ to statically adjust the weights for all the pairs. However, in practice the cases and the feature-value pairs in a case base are always not fully connected. The total time for this algorithm is, in fact, less than $O(N * M)$. In Chapter 6, we will show this situation.

3.4 Summary

Compared to the methods of Howe et al.[22](also discussed in Chapter 2), we can find that if the feature-value pairs in our algorithm are applied to the classes of cases, rather than the individual cases, then our algorithm is much like the third variant of CDW-Global Mean Expanded Feature CDW. Both methods are based on the statistical information from a case base. Their statistical information is about the classes of cases while ours is concerned with the individual cases. Besides that, we employ a different formula to compute the weight adjustment.

The static weight adjusting method is based on the statistical information hidden in a case base. Such an adjustment may not accurately reflect a user's real intention. Sometimes a user wants to adjust the weights of some particular feature-value pairs with more attention but put less importance on others. The static adjusting method is unable to fulfill this goal. It will just follow some pre-defined rules, say a formula, a threshold, or some other standards, to make its decisions for the adjustment without any discrimination on whose weight actually needs to be adjusted and whose in fact does not. In the dynamic learning method to be presented next we will try to overcome this flaw.

Chapter 4

Dynamic Weight Learning Method

4.1 Introduction

In Chapter 3, a static weight adjusting method is introduced into CBR. We call it static adjustment because the process involved in the adjustment totally depends on the statistical distribution of the feature-value pairs along a case base. We also observe that after the adjustment, the weight of each feature-value pair will be the same for the whole case base. Hence, this adjustment is global which is appropriate when the importance of each feature-value pair is constant across a case base.

We can easily find that the static weight adjustment is far behind what we conceive with regard to the dynamic nature of feature weights discussed in Chapter 2. Sometimes we need to attach different weights locally to each feature-value pair for individual cases in order to reflect a user's real intention more accurately and serve different purposes for different users. Why does a user choose this case rather than the others? It is because, after inputting a new problem description and specifying the feature-value pairs, s/he thinks this is the right case desired. On the other hand, this means the current feature-value pairs contribute much to the ranking score of this case. Although s/he does not express this explicitly, s/he desires to assign these feature-value pairs higher weights because next time when the same problem occurs again, the corresponding case will be scored higher and identified quickly. The weights for these feature-value pairs need to be strengthened. In contrast to this, a user may be unhappy with the current cases that have higher scores. 'Why does this case get such a high score? It is actually not the case I want', a user may ask this. The user's real intention at this time is that this case is not so helpful based on the current problem

description and the feature-value pairs. The user does not want to see it when the same problem is encountered again the next time. This means that the weights of the current feature-value pairs should not contribute so much to this case; they should be weakened. This is the real intention from a user. Undoubtedly, a system capable of learning a user's behavior needs to capture and reflect this intention in its own behavior accordingly. We also observe that different users may have different views on a particular characteristics of a case; thus they may want to assign different weights to it, showing their own personal interests. Furthermore, a user's behavior is changing over time, making it reasonable for us to think about the feature weights as time-related.

Given the facts as above, there is a strong need to introduce a learning component into a CBR system. In our imagination, this component is very similar to that introduced in [29], and should have the following functionality. First of all, it will act as an interface between a user and the core of a CBR system. Any action a user takes will be captured by the system. Second, the actions thus captured will be fed back into the system, and based on the learning model employed, the difference between the system's actual behavior and a user's desired behavior(which can be specified by the user implicitly or explicitly) will be computed, and the result will be incorporated back into the system(i.e. the weights for indexing the cases). Third, the learning component will accommodate different sets of weights, representing the fact that different users can have their own set of weights in order to show their individual interests when they use a case base.

We will attempt to explore the idea of introducing a backpropagation neural network into a CBR system. As pointed out by Aamodt et al. in [3], *in knowledge-intensive approaches to CBR, learning may also take place within the general conceptual knowledge, for example by other machine learning methods or through interactions with the user.* In our attempt, we combine a neural network and CBR into an integrated entity. While the reasoning part is still case based, the feature-value indexing part is now shouldered by a backpropagation neural network. A user's feedback or intention will be captured, memorized and incorporated into the weights of the feature-value pairs in a case base. We do such an integration with the hope that the case retrieval quality would be improved and refined gradually in order to approximate a user's behavior, while the whole integrated framework would be active over time to keep the same pace with its user's latest interests.

Scenario	$f(K, L)$	Meaning
1.	$L \Rightarrow K$	L before K
2.	$K \Rightarrow L$	K before L
3.	$L \rightleftharpoons K$	K supports L
4.	$K \rightleftharpoons L$	L supports K
5.	$K \Leftrightarrow L$	K and L support each other

Table 4.1: Scenarios for Knowledge-Base Systems(K) and Learning Systems(L)

4.2 Machine Learning in Knowledge-Based Systems

Aha[5] indicates that a hybrid system involving large knowledge base may profitably integrate an expert system and a learning system, provided that the learning system can manipulate the knowledge base appropriately. This integration is based on the argument from Simon[46] that *learning can be more efficient than programming for large knowledge-based systems.*

As shown in Table 4.1[5] the combinations of positions of K and L form different scenarios, where K represents a knowledge-based system, and L represents a learning system.

The simple and basic functionality of such a hybrid system can be expressed as follows.

$$RawData \rightarrow f(K, L) \rightarrow Output$$

where function $f(K, L)$ is a black box representing one of the scenarios as above.

In Table 4.1, scenarios 1 and 2 represent a serial processing diagram in which the result from one system feeds directly into the other. The examples for such kinds of hybrid systems are in [30, 47]. Scenarios 3 and 4 represent a master-slave relationship, in which one system supports the other. For example in scenario 3, the learning system will consult with the domain knowledge before producing output. One such hybrid system is CN2[35]. Similarly, the knowledge-based system in scenario 4 is supported by a learning system before producing output. One example for such a combination is described in [4]. Scenario 5 represents another hybrid diagram in which a knowledge-based system and a learning system support mutually and cooperatively, taking advantage of each other while overcoming their own drawbacks. In [34], the expert system modifies the parameters used by the learning system, which in turn uses a set of domain-dependent rules for making predictions.

Within these five scenarios, we are particularly interested in scenario 4. In our integrated framework the knowledge system will be CBR while the learning system will be a backpropagation neural network.

In CBR itself the reasoning and learning processes are combined together. The learning process takes place through the addition of new cases and the refinement of feature weights. In order to observe the scenarios designated as above to show a clear picture, we move the refinement of feature weights out as a separate learning component. The component for the addition of new cases will still be kept in CBR.

When they construct a case base using their domain knowledge and previous experiences, the domain experts will assign weights to the feature-value pairs of all the cases, which represent the desired context in which each individual case will be retrieved and reused. However, such prior specifications may not be always accurate and updated. Thus, after the initial construction and during the practical use, we employ a backpropagation neural network to learn and refine these weights dynamically, which will produce the optimal case retrieval result a user desires. This combination can be represented by scenario 4 in Table 4.1. In our integration, such a hybrid system which can learn to adjust the weights of feature-value pairs for a particular case or solution based on the success or failure of using that case or solution to solve a user's problem. For those matching feature-value pairs that have been judged to result in a successful case or solution, their weights need be strengthened, while for those causing a failure, their weights will be weakened.

4.3 Neural Networks

4.3.1 Components in A Neural Network

A neural network is motivated by the model of biological brains[54, 28]. It is one of the connectionist approaches which rely on a large number of simple computational units. This approach undermines the distinction between the data and the process inherent in the traditional computing practice. The units of computation, much like biological neurons, function independently. As a machine learning method, a neural network is different from others in that it represents knowledge implicitly in the patterns of interactions between components rather than expressing knowledge explicitly and manipulating it through an inference engine.

The basis of a neural network is an artificial neuron, which is shown in Figure 4.1.

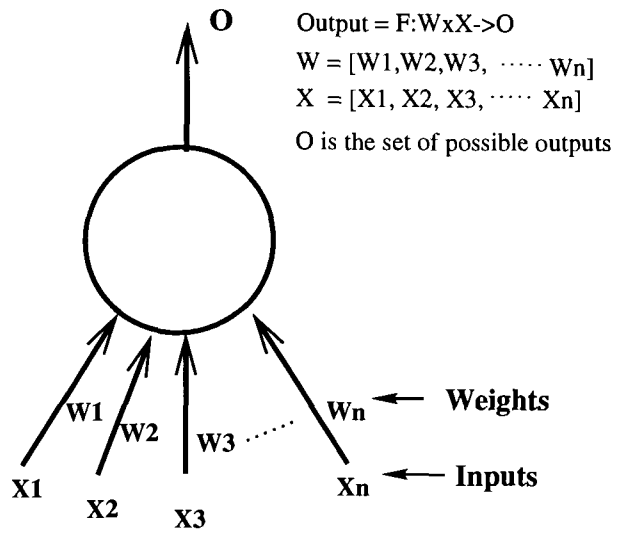

Figure 4.1: An Artificial Neuron

Basically an artificial neuron consists of the following[28]:

- *Input values,* X_i. These input values may come from the environment or the activation of other neurons. Each of them may be of a discrete value from set $\{-1, 1\}$ or $\{0, 1\}$, or a real-valued number;
- *A set of real-valued weights,* W_i;
- *A set of activation values.* The neuron computes its activation value from its weights and inputs. This value may become the input to other neurons or affect the result of the system as a whole;
- *An activation function,* F, that computes a neuron's activation value as a function of its weights and input data.

In addition to those properties for individual neurons, there are also some global characteristics for a neural network, such as

- *The network topology* of the connections between the individual neurons;
- *The learning algorithm* used in its learning process;
- *The environment* which includes the interpretation imposed on the input data to the network and the processing output result.

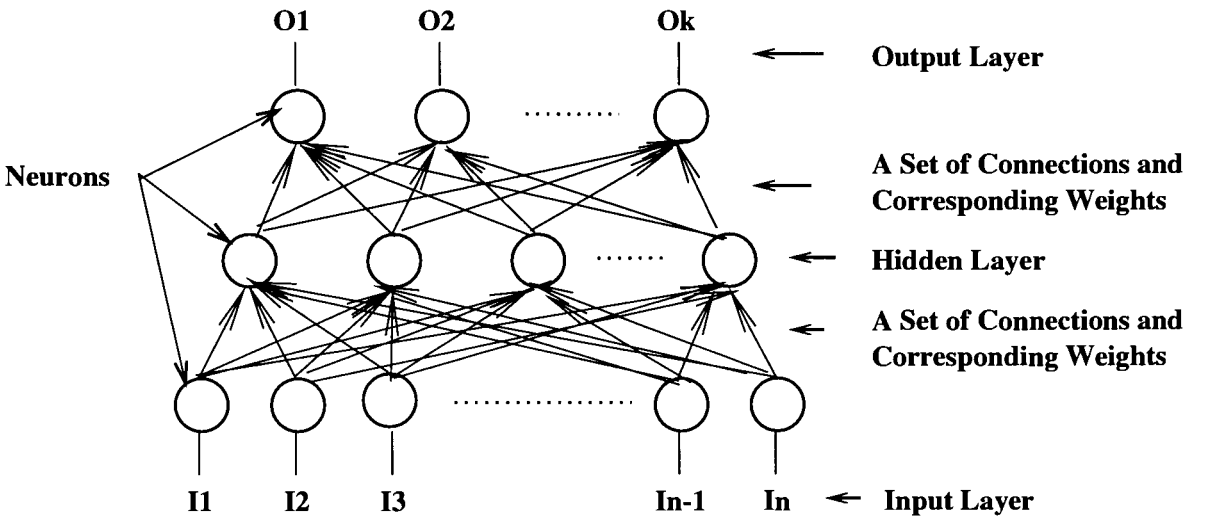

Figure 4.2: A Possible Architecture of Neural Network

Take a look at Figure 4.2 for one kind of a neural network's topology.

4.3.2 Delta Rule

There are many possible learning rules in a neural network, such as Hebbian learning rule, delta learning rule, Widrow-Hoff learning rule, and etc.. For further detailed information, see [54]. Of these learning rules, we are most interested in delta learning rule.

Given an activation function F(which will be given a realization in this chapter) as shown in Figure 4.1, we may define the amount of error as the difference between the actual output and desired output. The delta rule adjusts the weights as a function of this error.

For neuron i, let O_i be the actual output and D_i be the desired output. As shown in Figure 4.1, W_i represents the weight from input X_i to this neuron. Let F' represent the derivative of F. The following formula is used to compute the learning delta value which will be used in the weight adjustment of W_i.

$$\delta_i = (D_i - O_i)F'(\sum_{i=1}^{n} W_i * X_i) \tag{4.1}$$

From the mathematical perspective, the delta rule is often referred to *gradient descent learning*[28].

The delta rule is valid for continuous activation functions and can be used in the supervised-learning process(A learning process in which at each time step when the input is

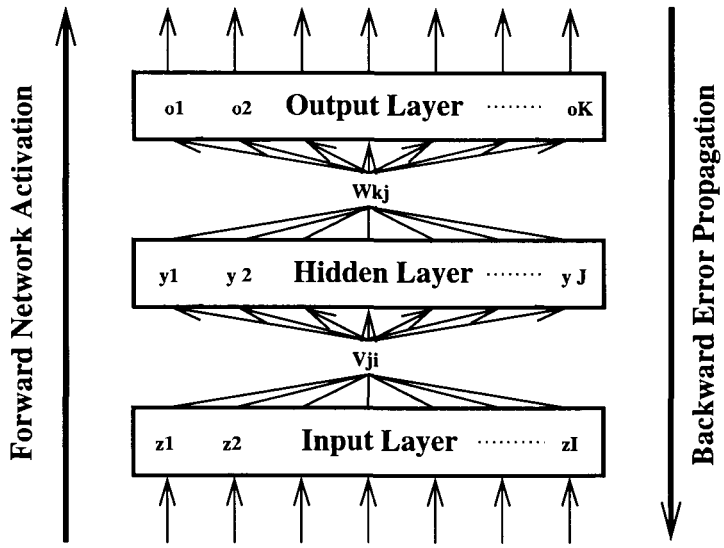

Figure 4.3: A BackPropagation Neural Network

applied, the desired response of the learning system is provided by the outside environment, and then compared with the actual response[54]). It is central to the functioning of the backpropagation learning in a multilayer network.

4.3.3 Backpropagation Neural Networks

A backpropagation neural network is one kind of neural networks. It works in the supervised-learning mode. Its learning rule is the generalized delta rule(which we will show later). The algorithm in this learning process is called an *error backpropagation training* algorithm.

Figure 4.2 is a typical structure which can be used to construct a backpropagation neural network. The backpropagation training algorithm gains its input/output mapping knowledge within a network by experiential accumulations. Input will be fed into the network sequentially during the backpropagation training. If an input is submitted and its output is determined to be erroneous, the connection weights will be adjusted so that the current overall error is reduced. The input/output mapping, the comparison between the actual and desired outputs, and the adjustment if necessary, will continue until all the data samples from the training set are learned within an acceptable error range.

The above learning process is also shown in Figure 4.3. The neurons in a backpropagation neural network are connected in layers, with the neurons at layer k passing their activations

only to layer $k+1$. In solving a problem, output is computed first at the input layer, through a hidden layer(or possible several hidden layers), and finally to the output layer or the outside environment. Given the desired output, the network calculates the error in the output neurons at the output layer. The error for a neuron at the hidden layer is a function of the errors on all the neurons that use its output. In general, the error for a neuron at layer k is a function of the errors of all the neurons at layer $k+1$ that use its output.

In a backpropagation neural network, activations move forward through the network in a layer-by-layer fashion while the error propagations transfer backward in a similar fashion.

4.3.4 Mathematical Description

In a backpropagation neural network, usually the activation function is

$$F(x) = \frac{2}{1+e^{-\lambda x}} - 1 \tag{4.2}$$

whose value range is $(-1,1)$, or

$$F(x) = \frac{1}{1+e^{-\lambda x}} \tag{4.3}$$

whose value range is $(0,1)$. In both formulas, $\lambda > 0$. It is called the activation rate which decides how fast the curve of the function will change with the change of x.

The computation of output begins from the input layer. As shown in Figure 4.3, the input to the network is represented as z_i for $i = 1,2,3,\cdots,I$.

The output of the hidden layer is $y_j (j = 1,2,3,\cdots,J)$, defined as follows.

$$y_j = F(\sum_{i=1}^{I} V_{j,i} * z_i) \tag{4.4}$$

where $V_{j,i}$ is the weight attached to the connection between neuron j at the hidden layer and neuron i at the input layer.

The output of a neuron at the output layer is $o_k (k = 1,2,3,\cdots,K)$, which can be computed using the following formula.

$$o_k = F(\sum_{j=1}^{J} W_{k,j} * y_j) \tag{4.5}$$

where $W_{k,j}$ is the weight attached to the connection between neuron k at the output layer and neuron j at the hidden layer.

Now compute the overall error:

$$E_k = \frac{1}{2}(d_k - o_k)^2 + E_{k-1} \tag{4.6}$$

where d_k is the desired output designated by the outside environment.

The computation of errors begins from the output layer. The delta rule discussed above is now generalized to compute the necessary weight adjustment for each layer in a back-propagation neural network. For the output layer, the learning delta value for neuron k will be computed as follows.

$$\delta_{o,k} = \frac{1}{2}(d_k - o_k)^2(1 - o_k^2) \tag{4.7}$$

where $k = 1, 2, 3, \cdots, K$.

The delta value for neuron j at the hidden layer will be computed using the following formula.

$$\delta_{y,j} = \frac{1}{2}(1 - {y_j}^2)\sum_{k=1}^{K} \delta_{ok} w_{k,j} \tag{4.8}$$

where $j = 1, 2, 3, \cdots, J$.

Then the weights attached to the connections between the output layer and the hidden layer will be adjusted using the above learning delta value(Formula 4.7).

$$w_{k,j} = w_{k,j} + \eta \delta_{o,k} y_j \tag{4.9}$$

where $k = 1, 2, 3, \cdots, K$, and $j = 1, 2, 3, \cdots, J$.

Finally the weights attached to the connections between the hidden layer and the input layer will be adjusted using the above learning delta value(Formula 4.8).

$$v_{j,i} = v_{j,i} + \eta \delta_{y,j} z_i \tag{4.10}$$

where $j = 1, 2, 3, \cdots, J$, and $i = 1, 2, 3, \cdots, I$.

In Formulas 4.9 and 4.10, η is the learning rate which decides how fast the adjustment speed is.

Repeat Formulas 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, and 4.10 for the each data sample in the training set. If this process finishes, check $E < \varepsilon$ to see if it holds, where ε is a small positive number specified by the domain experts. If true, stop the train process. Otherwise, the whole training process will be restarted from the beginning using the same training data set.

For details on the mathematical foundations, the generalized delta rule, and the applications of backpropagation neural networks, see [54, 28, 41, 32].

4.3.5 Bayesian Neural Networks

Bayesian neural networks parallel the backpropagation neural networks in that both employ the local, gradient-descent learning rule. They even have the same network structures. However, a Bayesian neural network is used to learn the representation of a probabilistic function, particularly a belief network[43]. Neal in [33] shows that a Bayeisan neural network can be used to define the probabilistic models for regression and classification tasks. It uses the outputs from the network to define a conditional distribution for one or more targets, given the various possible input values. The result of Bayesian neural-network learning is a trade-off between the prior belief in a model against its degree of agreement with the observed data.

From this observation a Bayesian neural network is quite different from a backpropagation neural network in their learning targets. In addition, although both have the same network structures, nodes in a Bayesian neural network represent the well-defined semantics and well-defined probabilistic relationships with each other. These properties seldom appear in a backpropagation neural network.

In CBR, if we consider that a user's behavior is encoded in the weights assigned to the feature-value pairs of all the case in a case base, then we can find that this behavior is variable from user to user, and from time to time even for the same user. It is difficult to find a prior probabilistic model to describe it. It is also difficult to predict how a user's behavior changes with time. Thus, with these facts in mind, we select the backpropagation neural network as the learning component in CBR to explore the possibility of the integration of a knowledge-based system and a learning system.

4.4 Motivations for Integration

The desire to use a backpropagation neural network as the learning component in CBR is also motivated by the study of a case base itself. In our experiences with different domains, our first observation is that in many domains some information in the content of a case can be shared concurrently by several cases. This piece of information, if copied to anywhere a case uses it, will be redundant and possibly inconsistent, making the maintenance of a case base difficult[38]. Thus we hope to separate such a piece of information from each case, and use a pointer to represent it anywhere it is needed. Of the pieces of information that compose a case(see Chapter 1), we find that while problem description is relatively unique,

problem solution maybe shared by several cases. Another observation is that a problem may have several possible solutions, with each solution having different importance, based on the current problem description as well as the feature-value pairs matched. The same solution has, possibly, different ranking scores in different cases. These observations also can be supported by the example of route planner in [20]. There are always several routes connecting the start point and the end point. Another example is that a medicine advisor, when dealing with the same symptom, may suggest several prescriptions appropriate for it.

In the original structure of a case base, there are basically two layers. We can regard that the cases are situated at the output layer of the structure while the feature-value pairs are at the input layer. The feature-value pairs are connected to their associated cases, forming an index into the case base. The weights are attached to the connections between the feature-value pairs and the individual cases to decide which case will be retrieved and how it will be scored when a new problem is presented. Now in the light of the observations as above, we need to reconsider this structure.

We change the original two-layer structure of a case base into a three-layer structure. We extract the solutions from each case, and put them onto a third layer. This makes it possible to share a solution by several cases, and meanwhile reduces the redundancy in the case base. In order to make these changes possible, a second set of weights is essential, which will be attached to the connections between cases and their possible solutions. This second set of weights represents how important a solution is to a particular case if this solution is a potential candidate for this case. In addition, it distinguishes a solution within several cases if the solution belongs to several cases at the same time.

A case base, after the changes as above, will be three-layered with connections between two adjacent layers. The whole structure is graphed in Figure 4.4. In order to examine the locality of each feature-value pair attached to individual cases, we still use the expanded-feature set discussed in Chapters 2 and 3. In the figure, the feature-value pair layer and the case layer are the same as before. But we call the case layer the problem layer(it is also shown in the figure) from now on in this chapter in order to emphasize that we have separated the solution part from the problem part in a case, and a case is now mainly represented by its problem description. The solutions in each case are separated out, forming a solution layer. The second set of weights is a new one, which will be specified in the construction of a case base by the domain experts. Just like the weights attached to the connections between the feature-value pairs and the problems, the second set of weights will also be

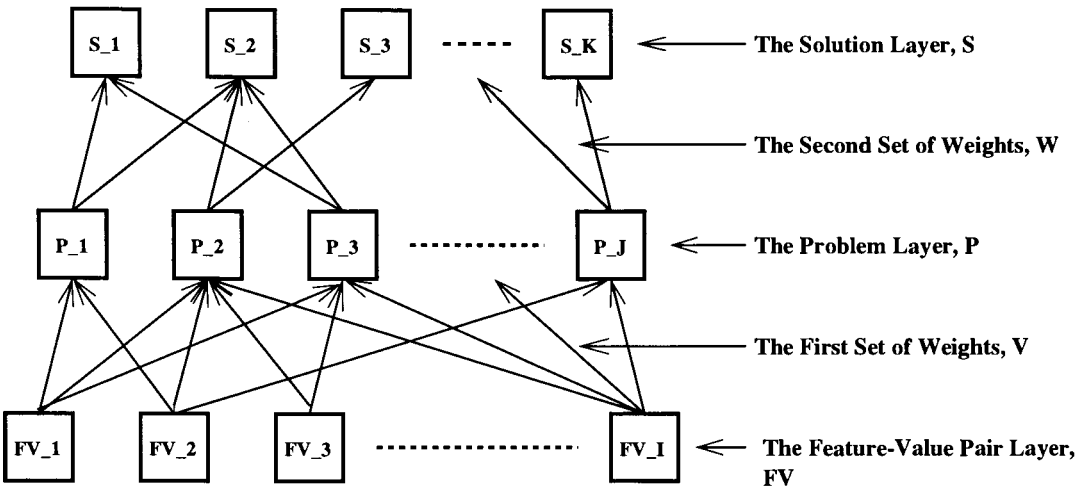

Figure 4.4: New Structure of A Case Base

learned dynamically in the use of the system.

Based on the discussions on the learning component(see Chapter 2), as well as the necessary changes we have made into our system as above, we can see that, on one hand, we need an underlying learning component which employs a certain learning mechanism in order to fulfill our goal, and on the other hand, such a learning component needs to accommodate the changes we have made as above.

It can be easily seen that the three-layer structure shown in Figure 4.4 is very similar to that of a backpropagatoin neural network. Furthermore, the two sets of weights inspire us to adapt the learning mechanism in a back-propagation neural network, which is very mature in the AI literature. However, as we also can see, although the structure in Figure 4.4 has the similarity to that of a backpropagation neural network, it is actually not a neural network. In order to adapt it into CBR, some modifications are essential.

4.5 Learning Model and Mathematical Description

4.5.1 User's Learning Model

The learning process in our method is similar to that in a backpropagation neural network. However, our learning process is an interactive one, while in a backpropagation neural network the learning process is a batch and automatic one.

In our learning model, there is no explicitly-defined training data. The training process

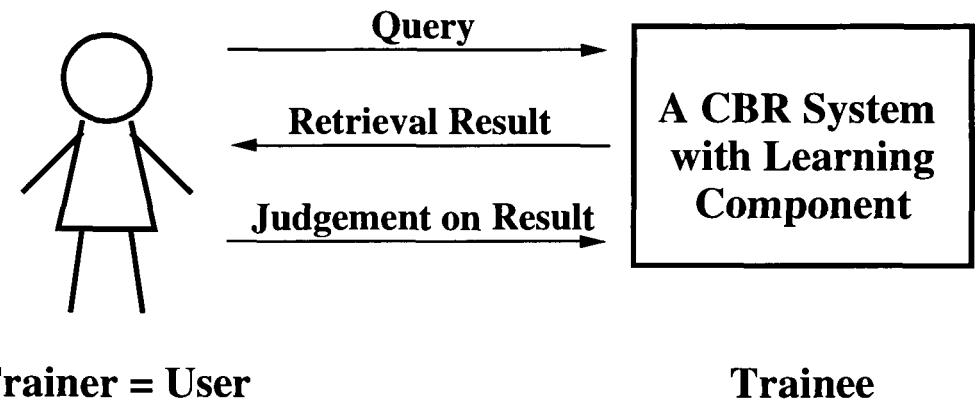

Figure 4.5: User's Learning Model

is incorporated into the use of the CBR system. A user's responses to the system's behavior form an implicit source of training data. Besides this, in contrast to a neural network, a trainer is not distinguished from a user in this model.

Figure 4.5 shows this model. The model has two parts. One is the trainer or user. The other is the trainee or the CBR system with a neural-network learning component. The two parts will interact with each other with the trainee simulating the behavior of the trainer. A learning loop is as follows. The trainer initializes a query to the CBR system. According to this query the system produces some retrieval result, and returns it back to the trainer. The trainer might make judgement on the retrieval result, and feedback this information to the system. Then the system will capture and learn this feedback information.

In the following discussions, we make assumptions about our learning model.

1. In our learning model, an end-user also plays the trainer's role. They can be the same person. Our model serves its user individually. The training process proceeds with her/his use of the system, and will never stop.

2. The inputs to the system might not cover the whole input space in a domain. However, they will cover the inputs which will occur with higher possibilities.

3. Our learning model is an interactive one. It needs an end-user to feedback information in order to change itself.

In neural-network learning process, there will be a training stage and a testing stage. However, in our model, under the first assumption, there will be no testing stage. Only the

training stage exists in our learning model. The training stage will last forever with the retrieval results approximately converging to the optimal ones specified by the user.

Under the second assumption, our system will learn the inputs which will appear with higher possibilities. The retrieval error for these frequently-used inputs will become smaller and smaller. Even though the retrieval error for the uncovered inputs might be large, as the possibilities under which these uncovered inputs occur are very low, therefore the overall error of the system will still be very low.

Under the third assumption, our system needs the feedback information from a user as its learning source. The more information a user provides, the faster the learning will be. Only the information fed back from the user will make the learning process possible.

4.5.2 Mathematical Description

With regard to the structure shown in Figure 4.4, we introduce some symbols in order to facilitate our description.

In a case base, there is a total of N features. For each feature F_i, there are m_i values, where $i = 1, 2, \cdots, N$. The case base contains J problems and K solutions.

For the structure shown in Figure 4.4, there is a total of $I = \sum_{i=1}^{N} m_i$ feature-value pairs, or nodes in the feature-value pair layer. We label these feature-value pairs as FV_i, where $i = 1, 2, 3, \cdots, I$. In the problem layer, we use P_j to represent each problem, where $j = 1, 2, 3, \cdots, J$. In the solution layer, we use S_k to represent each solution, where $k = 1, 2, 3, \cdots, K$.

The first set of weights $V_{j,i}$ is attached to the connection between problem P_j and feature-value pair FV_i if there is an association between them, and the second set of weights $W_{k,j}$ is attached to the connection between solution S_k and problem P_j if S_k is a solution to P_j.

Computation of A Problem's Score

A problem's score is computed first based on the feature-value pairs which are selected by a user at the feature-value pair layer.

The score will be computed using the following formula.

$$S_{P_j} = \frac{2}{1 + e^{-\lambda * \sum_{i=1}^{I} (V_{j,i} * X_i)}} - 1 \tag{4.11}$$

where $j = 1, 2, 3, \cdots, J$, S_{P_j} is the score of the problem P_j, and X_i is 1, if there is connection between problem P_j and feature-value pair FV_i **and** FV_i is selected. Otherwise X_i is 0.

We can see that the computation of S_{P_j} is exactly the one employed in a backpropagation neural network shown in Formula 4.4. However, in order to adapt it into CBR system, we need to add a parameter which will be discussed in the practical system design and implementation in Chapter 5.

Formula 4.11 has the property that the higher $\sum_{i=1}^{I} V_{j,i}X_i$ is, the higher S_{P_j} is. This property is also demonstrated in k-NN algorithm in the case retrieval process discussed in Chapter 2.

When the problem scores are computed, they will be presented to a user for her/his judgement. The user may select a problem for further consideration. S/he may confirm or disapprove it based on her/his preference.

Computation of A Solution's Score

We only compute the scores of the solutions associated with the current **selected and confirmed** problem. The computation of a solution's score is corresponding to the computation of an output in a backpropagation neural network. However, they have some differences, requiring that we use the following formula to compute it.

$$S_{S_k} = \frac{2}{1 + e^{-\lambda * \sum_{j=1}^{J} (W_{k,j} * S_{P_j} * \alpha)}} - 1 \tag{4.12}$$

where S_{S_k} is the score of solution S_k, and S_{P_j} is the score of problem P_j. If there is no connection between solution S_k and problem P_j, then we do not include it in $\sum_{j=1}^{J} (W_{k,j} * S_{P_j} * \alpha)$

We can easily see that except for the parameter α, all the remaining are the same as those in Formula 4.5. We call α the *bias factor*. The reason we introduce bias factor is that in the structure shown in Figure 4.4, a user should first select which problem at the problem layer is the most desired one based on her/his current preferences. This information needs to be reflected in the following computations of the solution scores. We imagine that the selected problem has a higher bias factor whereas the unselected problems have a lower one, making the selected problem contribute more than the unselected ones in the computation of the solution scores; thus the solutions of the selected problems will possibly have relatively higher scores. However, this is only what we desire. As shown in Formula 4.12, the actual final solution scores will be computed globally, giving the true picture of why a solution gets a higher score, why a solution gets different scores in different associated problems, and which problem contributes most in the computation of a particular solution score.

Delta Learning Rule For Solutions and Problems

As soon as the score of a solution is computed, it will be presented to a user for her/his judgement. If the user thinks that this solution is the right one and has an appropriate score, s/he can confirm it. Otherwise, s/he can disapprove it. In both situations, a user can have the option to specify what the desired score of the solution is. This information will be captured by the learning component, and will be used in the computation of the errors. We have also considered another situation that if a user does not specify the desired score, but just makes confirmation or disapproval on a solution, a default adjustment value will be added or deducted from the computed solution score automatically to get the desired score. For instance, if a solution gets an actual computed score of 80, and is disapproved, the desired score of this solution will be 75 if the current default adjustment is 5. However, a user can specify the desired score to 71, 73, or some value else as long as the specified value is less than 80.

The computation of the learning delta value is first done at the solution layer. According to the computation of solution scores(see above), we only compute the learning delta values for the solutions associated with the current **selected and confirmed** problem. The following formula is employed.

$$delta_{S_k} = \frac{1}{2} * (D_{S_k} - S_{S_k}) * (1 - S_{S_k}^2) \tag{4.13}$$

where $delta_{S_k}$ is the learning delta value for solution S_k, and D_{S_k} is the desired score for S_k. The formula here is the same as Formula 4.7 used in a backpropagation neural network.

The learning delta values then are propagated back to the problem layer. The computation of the learning delta value at this layer is done using the following formula.

$$delta_{P_j} = \frac{1}{2} * (1 - S_{P_j}^2) * \sum_{k=1}^{K} (delta_{S_k} * W_{k,j}) \tag{4.14}$$

where $delta_{P_j}$ is the learning delta value of problem P_j. If there is no connection between solution S_k and problem P_j, then we do not include it in $\sum_{k=1}^{K} (delta_{S_k} * W_{k,j})$. Again the formula here is the same as Formula 4.8.

Weight Adjustments

After computing the learning delta values for weight adjustments, next we need to adjust the weights from the solution layer to the problem layer, and then from the problem layer to the feature-value pair layer.

We will adjust the weights attached to the solutions which are associated with the current **selected and confirmed** problem. The weights attached to the connections between the problems and the solutions will be adjusted first using the learning delta values computed in Formula 4.13, and the problem scores computed in Formula 4.11. The formula for this adjustment is:

$$W_{k,j}^{new} = W_{k,j}^{old} + \eta * delta_{S_k} * S_{P_j} \tag{4.15}$$

where $W_{k,j}^{new}$ is the new weight to be computed, and $W_{k,j}^{old}$ is the old weight attached to the connection between solution S_k and problem P_j.

Compared with Formula 4.9, η is the learning rate. However its meaning is different from that in a backpropagation neural network. Recall that in the computation of the solution scores, we introduce the *bias factor* in order to adapt a backpropagation neural network into a CBR system. Accordingly, when we adjust the weights we still have to consider this fact. We hope that the learning rate would be higher for the selected problem than that for the unselected ones, which means that the speed of the weight adjustment for the selected problem is faster.

The weights attached to connections between the problems and the feature-value pairs will be adjusted next using the learning delta values computed in Formula 4.14. It is shown as follows:

$$V_{j,i}^{new} = V_{j,i}^{old} + \eta * delta_{P_j} * X_i \tag{4.16}$$

where $V_{j,i}^{new}$ is the new weight to be computed, and $V_{j,i}^{old}$ is the old weight attached to the connection between problem P_j and feature-value pair FV_i. X_i is 1, if there is a connection between problem P_j and feature-value pair FV_i **and** FV_i is selected in the new problem. Otherwise X_i is 0.

The learning rate η is the same for all the weights. This is in accordance with the computation of the problem scores. The computation formula for adjusting weights between the problems and the feature-value pairs is exactly the same as that employed in a backpropagation neural network.

Adjustments for Problems

After a new problem is presented, all the previous problems will be scored according to Formula 4.11 with the promising ones at the higher ranks. If a user wants to feed back some information to the system directly at the problem layer as whether a problem is desired or

not, what should we do? We can employ a single layer neural network to fulfill this task. If we do not consider the solution layer in Figure 4.4, the structure is a single layer neural network.

We still use Formula 4.11 to compute the problem scores. But we will use the following formula to compute a new weight.

$$V_{j,i}^{new} = V_{j,i}^{old} + \frac{1}{2} * \eta * (DS_{P_j} - S_{P_j}) * (1 - S_{P_j}^2) * X_i \tag{4.17}$$

where each symbol represents the same as before. However, as X_i can be 1 only when there is a connection between feature-value pair FV_i and problem P_j **and** FV_i is selected, such an adjustment is local only to a selected problem as compared to the global adjustment computed in Formula 4.16, which might affect all the weights connecting the selected feature-value pairs and the problems.

A Simple Example

In order to introduce a neural network into a CBR system, we have discussed many formulas which can be used to fullfil the task. For simplicity's sake, we will use a small example to show how to adjust the weights associated with the feature-value pairs and the problems.

We take the small case base shown in Table 2.1 as our demonstration. We select column 2, 3, and 4 of the table as the features to form the indices for a problem. All the relevant feature-value pairs' weights will be initialized to 0.5.

Assume there is a target T which is described by the feature-value pairs as (monthly income, 30 units), (job status, salaried), and (monthly repayment, 5). Using Formula 4.11, the score of Case 1 will be $\frac{2}{1+e^{-(0.5*0+0.5*1+0.5*0)}} - 1 = 0.25$, the score of Case 2 will be $\frac{2}{1+e^{-(0.5*0+0.5*1+0.5*1)}} - 1 = 0.46$, the score of Case 3 will be $\frac{2}{1+e^{-(0.5*1+0.5*0+0.5*0)}} - 1 = 0.25$, and the score of Case 4 will be $\frac{2}{1+e^{-(0.5*0+0.5*1+0.5*0)}} - 1 = 0.25$.

After these problems, along with their scores, are presented to a user, s/he may be not satisfied with the score of Case 2. s/he thinks it is desirable to have its score of 0.8. Thus s/he can tell the system that the score of Case 2 should be 0.8. The system will capture this response, and use Formula 4.17 to compute the necessary adjustment for the feature-value pairs associated with Case 2. In this example(assume $\eta = 0.9$), all the adjustments will be $\frac{1}{2} * 0.9 * (0.8 - 0.46) * (1 - 0.46^2) = 0.032$. After the weights are adjusted, the system will compute all the problem scores again to see whether Case 2 has the desired score of 0.8. This process sometimes needs to repeat several times.

4.5.3 Learning Policy and Updating Policy

There are two aspects we need to consider when constructing a learning component for feature weighting: learning policy and updating policy. Basically, four different learning policies are possbile[1]:

1. If there has been a retrieval success, then the learning component will increase the weights of matching features;
2. If there has been a retrieval success, then the learning component will decrease the weights of unmatching features;
3. If there has been a retrieval failure, then the learning component will increase the weights of unmatching features;
4. If there has been a retrieval failure, then the learning component will decrease the weights of matching features;

Different learning algorithms choose different combinations of these four policies. Our strategy employs (1) and (4), i.e., for a problem or solution retrieval success or failure, we will increase or decrease the weights of matching feature-value pairs. In Figure 4.4, only the feature-value pairs which are selected will get input as 1, making themselves contribute to the problem scores and the final solution scores, and thus getting adjusted.

In our integrated framework, the updating policy is a set of Formulas 4.13, 4.14, 4.15, 4.16, and 4.17 shown above.

4.5.4 Learning Process

In a backpropagation neural network, there is a training sample set. The sample data in that set will go through the network one by one - a learning process in which the weights connecting adjacent layers are adjusted. This process will have to repeat several times in order to get the final satisfactory goal.

In the integration of CBR with a backpropagation neural network, we assume that there is no such an explicit sample training data set. A user will just tell the system whether a retrieved problem or solution is desired or undesired. The system will capture these responses, and feed them back into the neural network learning component.

In the short term, we can not see this as a learning process. But in the long run, it actually is. A user's preferences are demonstrated in their responses to the system's behavior during the use of the system. If we take only one preference at a time, we can adjust the weights to accurately approximate it. However, because of the interactions between the feature-value pairs, for a set of a user's preferences, maybe it is not possible for the adjustments to be accurate for all the preferences in just one loop. This requires an iterative process. Every iteration through all the preferences will tend to approximately delineate them gradually, trembling around the desired case scores at increasingly small distances.

Interestingly, a user may not know this. Every time when s/he thinks that a problem's or a solution's score is too high, s/he just tells the system to lower it to a score specified explicitly or implicitly. If the score is thought to be too low, s/he can also tell the system to increase it. This process iteratively proceeds, until there is no such abnormalities happening. Although the final score of each problem or solution may not be exactly the same as its correspondent in the user's ideal preferences, the difference is within an acceptable range. The relative ranks among the problems or solutions will reflect which problem or solution is more relevant and which one is less.

The learning process in such an integrated CBR system will never stop in the sense that the environment is changing over time. The changes of a user's preferences will also be melted into the above process.

The dynamic weight learning algorithm is described as follows:

Forever

1. *For the current problem description and input feature-value pairs, retrieve similar problems using Formula 4.11;*

2. *If the score of a selected problem is too high or too low, repeat adjusting the weights attached to the connections between this problem and its feature-value pairs using Formula 4.17 based on its desired score specified implicitly or explicitly;*

3. *If the selected problem gets confirmed, retrieve its associated solutions using Formula 4.12;*

4. *For each of the associated solutions, decide whether its score is too high or too low, and repeat adjusting the two sets of weights using Formulas 4.13, 4.14, 4.15, and 4.16;*

5. *Recompute the problem and solution scores. Wait for the user's next action(maybe s/he adjusts the other problems or inputs a new problem).*

4.5.5 Computational Analysis of Algorithm

Based on the structure of the integrated framework we describe as above, we now attempt to analyze the rough complexity of the dynamic weight learning algorithm. Because the learning process in our integrated framework will be iterative until an exit condition is satisfied(which will depend mainly on a user's specified desired score of a problem or a solution. See above discussions), we only analyze the computational complexity for the learning process of just one iteration.

Given a new problem which is input as a set of feature-value pairs, in order to compute the similarity between it and every previous problem, we take each problem in turn, and check whether it is associated with some feature-value pairs describing the new problem.

Under the above three-layer structure, for the feature-value pair layer, we set the *selected flag* of a feature-value pair to 1 if it is in the input set of feature-value pairs for the new problem. Assume that there are I feature-value pairs at the feature-value pair layer, J problems at the problem layer, and K solutions at the solution layer. For each problem at the problem layer, in order to use Formula 4.11 to compute its score, we need to check each feature-value pair at the feature-value pair layer to see whether there is an association between them **and** the feature-value pair is selected for the new problem. This will be done in $O(I * J)$. For the same argument, for each solution at the solution layer, in order to use Formula 4.12 to compute the score, we need to check each problem at the problem layer to see whether there is an association between them. Obviously, this is done in $O(J * K)$. Therefore we need a total of time $O(I * J) + O(J * K)$ to get the final solution scores at the solution layer. However, recall the algorithm as above. When the problem scores at the problem layer are computed, the problems will be presented to a user for her/his judgement, and we only compute the solution scores for the **selected and confirmed** problem. Thus on average, the total forward time of the three-layer structure is reduced to $O(I * J) + O(J * K/2)$.

The weight adjustment will be computed along the opposite direction. If a user feeds back her/his response directly at the problem layer as whether a retrieved problem is right or not, as discussed before, the adjustment is local to this selected problem. We need to check each feature-value pair at the feature-value pair layer to see whether it is associated

with the selected problem **and** currently selected as an input feature-value pair. This will be done in $O(I)$. If a user feeds back her/his response at the solution layer, the learning delta for that layer can be done in $O(K)$ using Formula 4.13. The learning delta for the problem layer can be done in $O(J * K)$ using Formula 4.14. Thus the total time for the computation of learning deltas will be in $O(K) + O(J * K)$. After the computations of the learning delta values, the adjustment for the weights attached to the associations between the solution layer and problem layer will be done in $O(J * K)$ using Formula 4.15 while the weights attached to the associations between the problem layer and the feature-value pair layer will be adjusted in $O(I * J)$ using Formula 4.16. Therefore the total time for the adjustment will be $O(I * J) + O(J * K)$.

Based on these analysis, we can see that for one iteration of the algorithm the time will be at most $O(n^2)$, where n is the maximum among the number of feature-value pairs, the number of problems, and the number of solutions in a case base. However, in our implementation, the neural network structure is not fully connected. Only if an association exists between a feature-value pair and a problem or between a problem and a solution do we store the connection to represent it. Thus, in the practical implementation, the total time is less than $O(n^2)$.

4.5.6 Real-Time Response

It can be easily seen that the algorithm introduced as above is an interactive process. Thus the system will be changed every time a user feeds back her/his response.

After a user feeds back her/his judgement to the system, the system will employ the learning algorithm to adjust the relevant weights. Then the scores of the problems and the solutions will be computed again and presented to the user for further judgement. All these are done in the background. The foreground output of the system will keep updated over time as a user uses the system.

4.6 Summary

The scenario of an interactive problem solving is now introduced into a CBR system. With the integration of a backpropagation neural network, a CBR system is now capable of learning from its environment.

Originally, maintaining an updated set of weights in CBR is on the shoulder of the domain experts. Now it becomes an internal self-adjusting component within an integrated CBR framework. The system itself can learn to adjust its own behavior to simulate a user's behavior, and furthermore, it no longer plays a slave part in the relationship with its human users. It is now a self-autonomous component which is responsible for its own behavior. The human users' interactions form a source of useful information the system can absorb and use.

The integration of CBR with a backpropagation neural network provides a possible solution to the problem of dynamic nature of a CBR system. The characteristics of the structure of a case base, the knowledge encoded in the connections between solutions, problems, and feature-value pairs, and the repetitive using and learning processes over time in CBR are the inspirations of such an integration. However, although all these sound reasonable, it is essential that this integrated framework be validated through the experimental tests.

Chapter 5

System Design and Development

5.1 Introduction

We implement both the static weight adjusting algorithm and the dynamic weight learning algorithm in the framework of the CaseAdvisor™ system, which is introduced in Chapter 1. CaseAdvisor™ is a case-based reasoning system implemented using both C++ and Java. This system is domain-independent, and is appropriate for many domains. It can be running in the PC and Internet environments as either a stand-alone or a client/server system. Up to now, it has been applied to many industrial applications and is now on its way to the commercial market.

Figure 5.1 shows the working process of the CaseAdvisor™ system. The whole system is divided into two separate modules, with the first one called Case-Authoring Module, and the second one called Problem-Resolution Module.

5.1.1 Case-Authoring Module

The left side of Figure 5.1 is the Case-Authoring Module, in which the domain experts will convert the unstructured documents from a domain into cases. In the system, a case has two generic parts. They are *problem description* and one or several possible *problem solutions*. Each part is not further partitioned. One such case is shown in Table 1.4 in Chapter 1.

The module provides a form to create the content in a case, i.e. case name, problem description, and problem solution(s). In order to accommodate various applications, they will also, based on their needs, attach some accessories to each case, such as a file, a decision

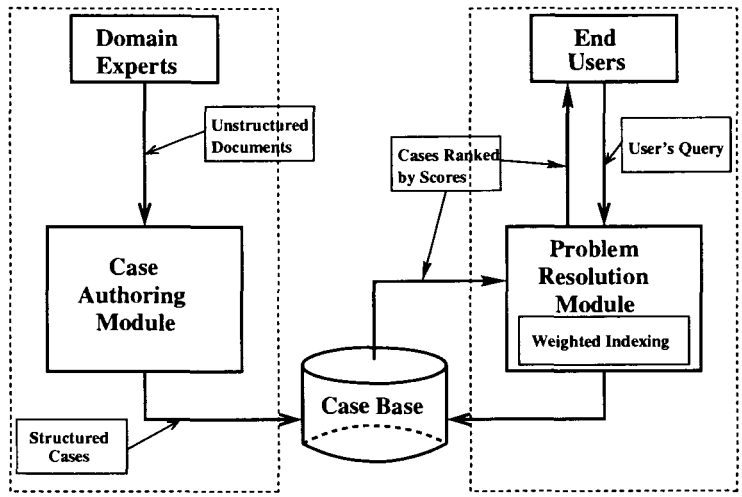

Figure 5.1: Case-Advisor System

tree and a list of the keywords extracted from the case name and/or the problem description. The keywords can also be generated directly by the domain experts.

The feature-value pairs in a case base are now represented as the question-answer pairs. When creating a case, the domain experts will also associate a set of question-answer pairs to each case, and assign a weight to each of them in order to index this case in the Problem-Resolution Module. This process generates a local weight set for each case, which provides great flexibility since if the domain experts want to assign weights globally, they can set the weight of each question-answer pair to be equal for each case it is associated with. Furthermore, it is not required that each case be associated with the same set of question-answer pairs. This provides more flexibility, since a particular case can be associated with a unique question-answer pair which might make it distinguish from others.

5.1.2 Problem-Resolution Module

The Problem Resolution-Module is on the right side of Figure 5.1. After a case base has been constructed in the Case-Authoring Module, it can be used to solve new problems in a domain in the Problem-Resolution Module. When encountering a new problem, a user first gives a high level description on what the problem is, using the keywords in that domain as many as possible. The system will use this description to retrieve a set of potential cases which *might* be similar to the current problem description. All the retrieved cases will be ranked from high to low by their scores. Thus, the scores in this module show the relative

comparison of the relevancy of each case to the new problem.

Then, the user may want to answer some questions, specifying what characteristics the new problem has. The system will use the weights assigned to these question-answer pairs to index the potential cases, making the promising cases have relatively high scores. This process further ranks the potential cases, reducing the case searching space. With the answering of more questions, the score of the more relevant case will get higher and higher, and at the same time, the less useful case will be ranked lower and lower. The more question-answer pairs are matched in a case, the higher score that case will finally get. It can be seen that the case retrieval strategy in the CaseAdvisor™ system is a variant of k-NN algorithm.

Although the CaseAdvisor™ system gets positive appraisal from its industrial applications, in order to further facilitate its use and improve its performance, we still need to do more with it.

In the system, the Case-Authoring Module takes the responsibility of constructing a case base. A user's real concern is the case retrieval quality in the later use of the Problem-Resolution Module. Although the domain experts try to use the weights to represent how important each characteristic is for each case, it is not easy for such a representation to be appropriate for every user. A user may have her/his own views on individual cases. On the other hand, in the Problem-Resolution Module, the weights assigned in the Case-Authoring Module play a very important role in the case retrieval process, since, a user will use them, based on her/his interests, to narrow down the number of cases with the hope that the desired ones would have higher scores while the undesired ones would have relatively lower scores. The process will give the user more confidence on which case is the one s/he needs.

To improve the case retrieval quality of the system, we introduce the static weight adjusting and dynamic weight learning components into the system. We hope that after importing these two components, the system will be a living system which will participate in the activities of a user in a dynamic environment. On one hand, the system can statically obtain its weights under the instructions of the domain experts, on the other hand, it can dynamically tune its weights in response to a user's behavior.

5.2 Design for Static Weight Adjusting Component

We rely on the statistical information hidden in the associations between the cases and the question-answer pairs in a case base for our static adjusting task. Of the two modules, the static weight adjustment will be executed in the Case-Authoring Module, just after a case base has been constructed. In the Problem-Resolution Module, the statically adjusted weights will then be used to retrieval the most similar cases. Thus the main design work is put into the Case-Authoring Module.

5.2.1 Adjusting Weights Statically

Information Collection

Recall the formal description of the adjusting model in Chapter 3, the first part of the static adjusting algorithm is to collect the statistical information from the associations between the cases and the question-answer pairs in a case base.

Figure 5.2 shows such information for each question-answer pair. In the figure, the first list shows the question-answer pairs(i.e., the feature-value pairs) while the second list displays the weight for each case initially specified by the domain experts. The third one lists the statically adjusted weights. Initially, the information in the third list is the same as that in the second list. However, after adjusted, they might be different.

Also in order to demonstrate a clear picture of the static information about a case base, the total number of cases and the number of cases associated with each question-answer pair will be shown out for reference at the bottom of Figure 5.2. These information will be used to adjust the weights statically.

Adjustment Process

We have discussed in Chapter 3 that we divide a case base into five groups. They are *very good* group, *good* group, *average* group, *bad* group, and *very bad* group.

The adjusting process is composed of the following three steps.

- Step 1. As shown in Figure 5.3, the five groups are represented by five different colors. Each group corresponds to one part in a range bar. From left to right in the bar, the ranges are very good, good, average, bad, and very bad, each of which is represented by its corresponding color. The whole range bar is 100 percent. Also you can see there

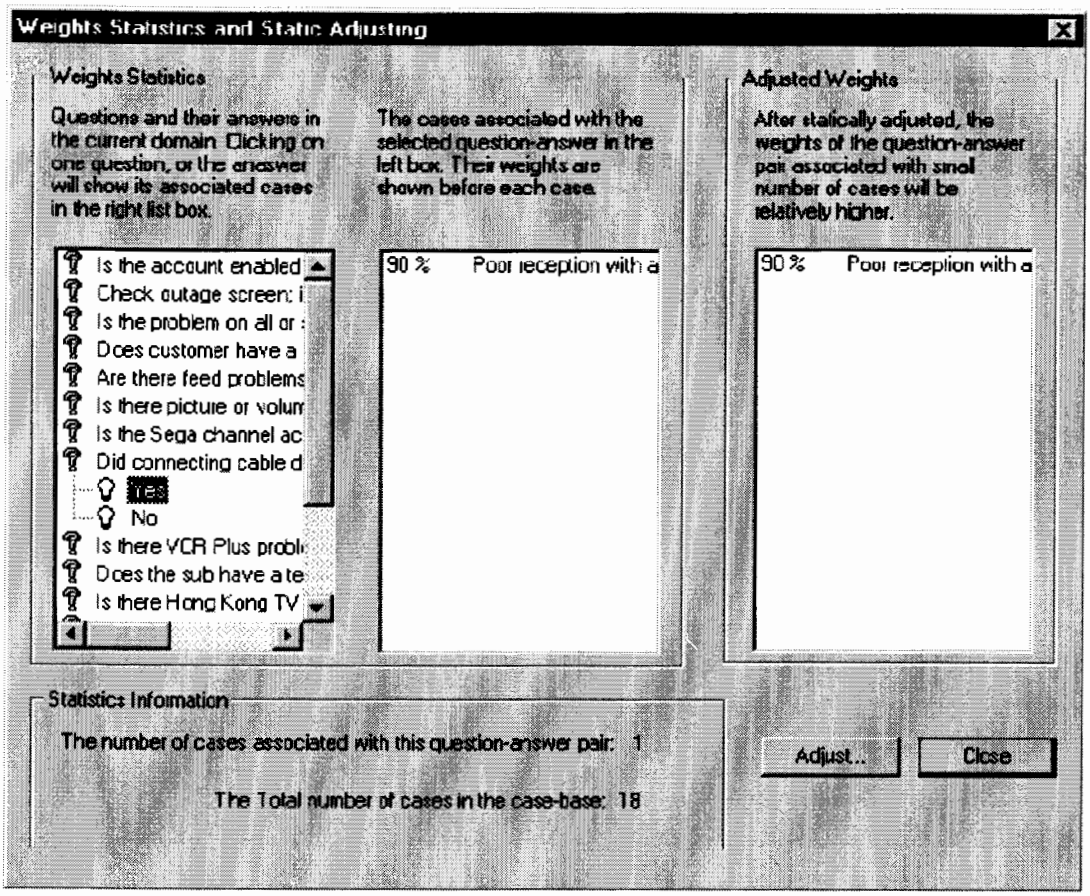

Figure 5.2: Static Information in A Case Base

are four small carets above or below the range bar with which the domain experts can drag to specify each range.

The domain experts can specify the ranges based on their domain knowledge. They can also click on the default button to retrieve the system default percentage ranges for each group.

- Step 2. Figure 5.4 shows the corresponding minimum weight for each group. If one question-answer pair falls into one particular group, it should have at least the minimum weight for that group. The domain experts can specify the minimum weight for each group. After specified, each minimum weight will be checked to make sure that they fall between 0 and 100, and they are in the order that the one for the very good group is the highest while the one for the very bad group is the lowest. The

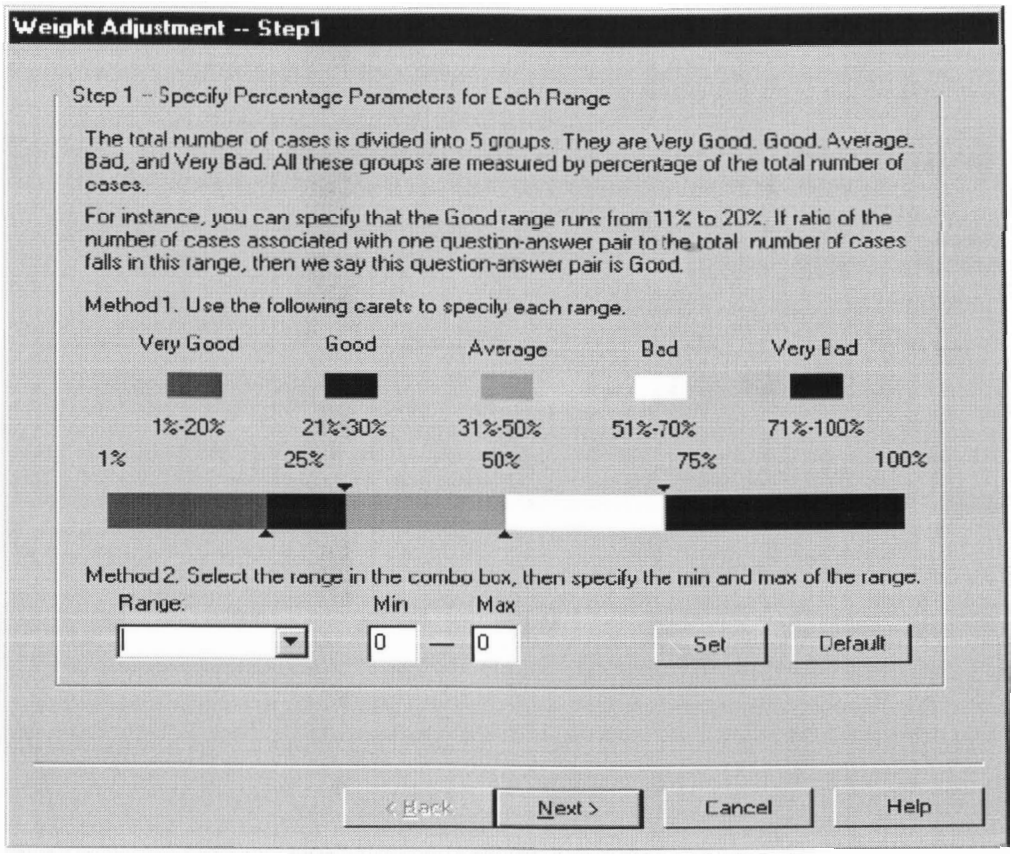

Figure 5.3: Step 1 in Static Weight Adjusting Process

system, according to the size of a case base, has its own default minimum weight for each group, which can be retrieved.

- Step 3. The information specified in Steps 1 and 2 will be displayed in Step 3, as shown in Figure 5.5. This step will not let the domain experts do anything but trigger the adjustment. The question-answer pair which is being adjusted will be highlighted in the box under the title **Question-Answer**. The whole adjusting process will be displayed in the progress bar under the title of **Adjustment Progress**.

All the parameters in this process have their default values set by the system. However, the above three steps provide the domain experts a chance to modify them, according to their experiences, their domain knowledge, and their own views.

The size of a case base plays an important role in determining these parameters. We consider an extreme situation in which a case base is too small, say - only 20 cases. In this

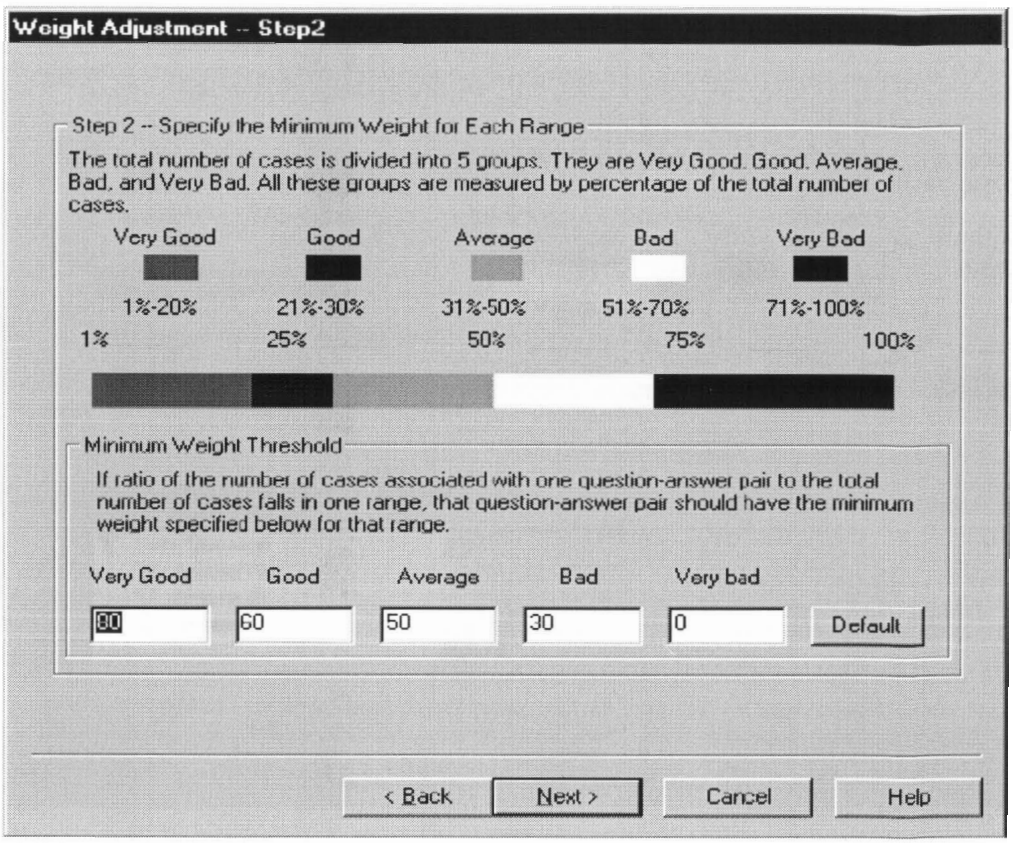

Figure 5.4: Step 2 in Static Weight Adjusting Process

situation, we do not do the adjustment, since such a case base is too small for the collected *statistical* information to be reliable.

5.2.2 Using Statically Adjusted Weights

There is no particular user-interface in the Problem-Resolution Module for the static weight adjusting component, since the weights specified in the Case-Authoring Module are transparent to the end-users(which is not true in the dynamic weight learning component). However, there is still a menu item which give an end-user an option to decide which set of weights they prefer to use, the one specified by the domain experts or the one adjusted statically in the Case-Authoring Module.

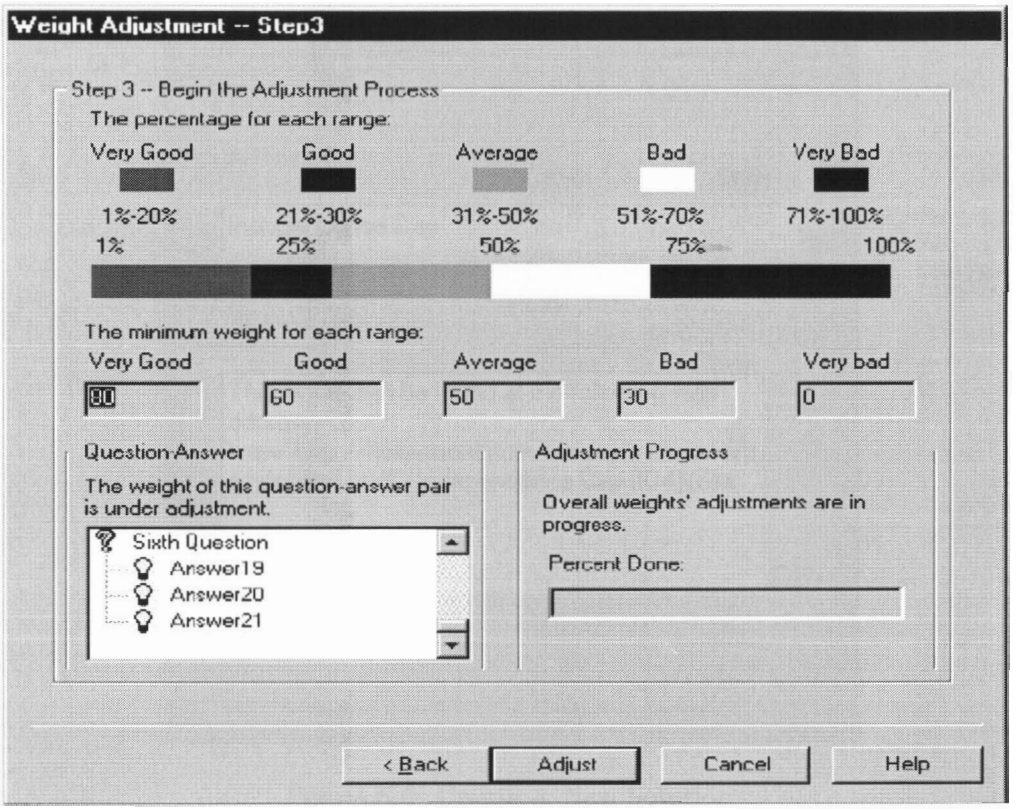

Figure 5.5: Step 3 in Static Weight Adjusting Process

5.3 Design for Dynamic Weight Learning Component

In contrast to the static weight adjusting component, the dynamic weight learning component moves the weight adjustment from the Case-Authoring Module to the Problem-Resolution Module. The main work for this component is done in the Problem-Resolution Module. Using the same convention in Chapter 4, we will use problem instead of case when referring to a case in our description.

5.3.1 Constructing and Using Solution Base

In Chapter 4, we have discussed the extraction of solutions from cases. Constructing a solution base for a case base is relatively easy as compared to the construction of a case base.

A solution is composed of two parts, the *solution name* and the *solution body*. The

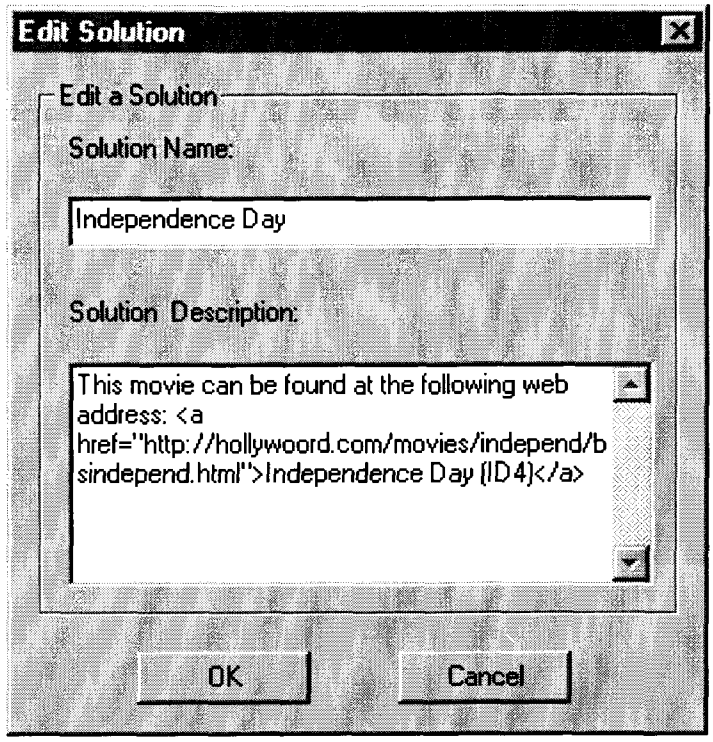

Figure 5.6: Create A New Solution

solution name is a pointer that will be used in any problem that has this solution attached, while the solution body provides the actual solution. A solution base can be constructed before the construction of a case base if the domain experts, based on their domain knowledge and experiences, can determine beforehand all the solutions in a case base. See Figure 5.6 for the user-interface of constructing a solution base.

When creating a problem, the domain experts will analyze its solution(s). They will search for each solution in the solution base, and then associate it with the problem. Meanwhile, they will assign a weight representing the importance of the solution in this problem. Figure 5.7 shows this process. Because all these weights, along with the weights attached to the associations between the problems and the question-answer pairs in the case base, will be adjusted dynamically in the Problem-Resolution Module, so the system sets a default value of 0.5 for each of them. However, the domain experts can change it

All these work is done in the Case-Authoring Module. Compared to the static adjusting component the most important work for the dynamic learning component is in the Problem-Resolution Module. The work involved in the Case-Authoring Module is easier for the

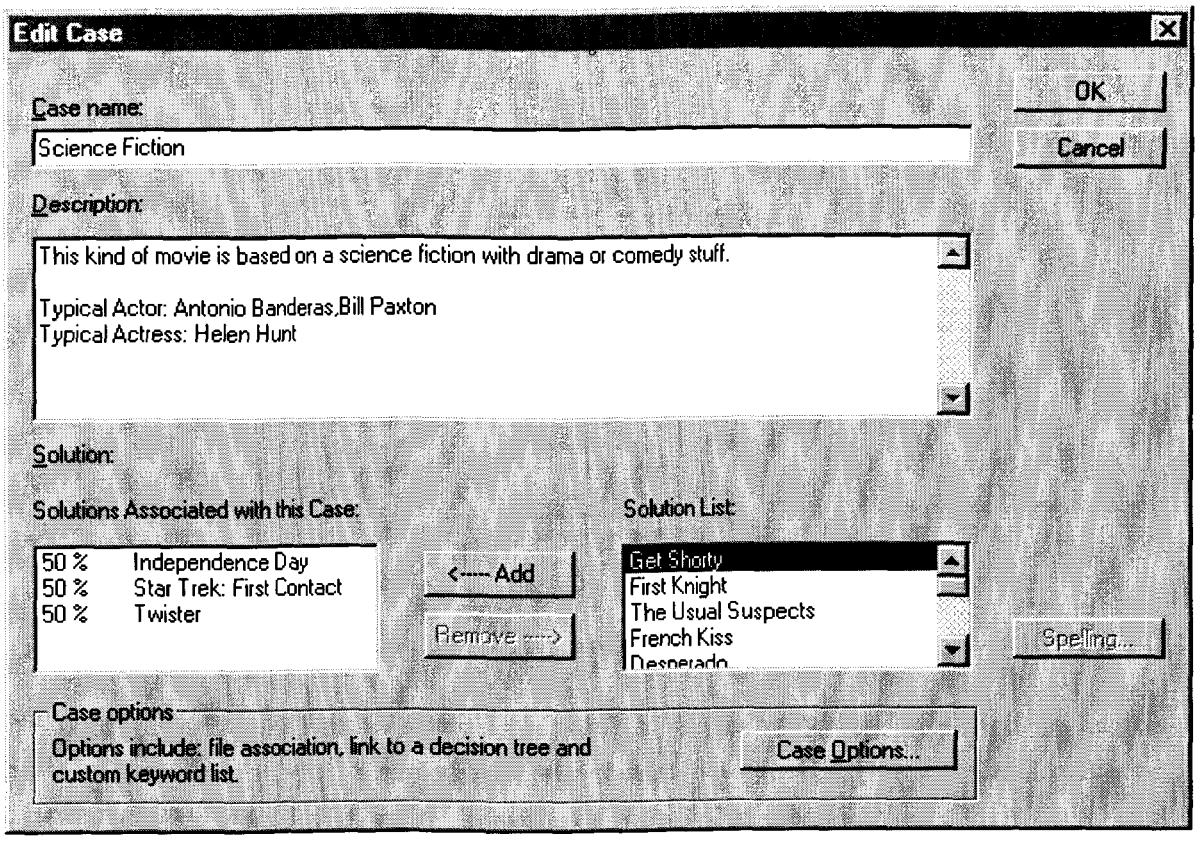

Figure 5.7: Associate A Solution to A Case

dynamic learning component.

5.3.2 Using and Learning Weights Dynamically

After introducing a neural-network learning component into the Problem-Resolution Module, the system is now capable of learning a user's behavior from her/his use of the system. The whole process of the Problem-Resolution Module is now shown in Figure 5.8. We will explain this figure later.

Users Own Weights

As discussed in Chapters 2 and 4, different users may want to have different sets of weight in order to demonstrate their personal behaviors and interests. Figure 5.9 shows this process. When a user uses the system, s/he needs to identify her/himself by choosing her/his own

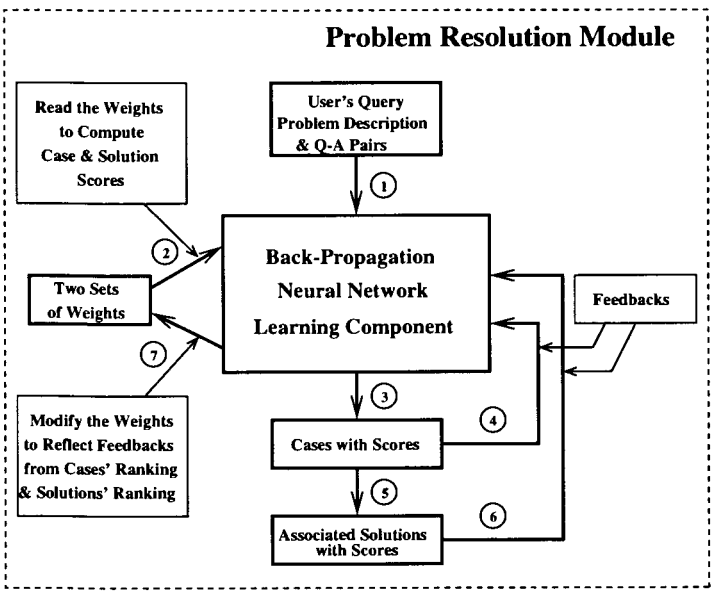

Figure 5.8: Back-propagation Neural Network Learning in Problem-Resolution Module

sets of weights. Also when a new user is going to use the system, s/he can create her/his own weights which will be copied from the default ones specified in the Case-Authoring Module by the domain experts or from the ones statically adjusted there.

Thus, although all the users are using the same case base, their personal behaviors and interests can be different. This scenario also anticipates the future for a distributed case base, in which multi-users can share the same case base but use different weights.

Learning Weights Dynamically

In order to use the result of the trigram matching(see [25] for more discussions on trigram matching) in the problem retrieval process when computing the problem scores, we reconsider Formula 4.11, and change it as follows.

$$score_{P_j} = \frac{2}{1 + e^{-\lambda * \sum_{i=1}^{I} V_{ji} X_i}} - 1 + T_j$$

where T_j is the result of the trigram matching for each problem P_j.

This formula not only incorporates the result of trigram matching, but also represents a situation where even entering a problem description without answering any questions still retrieves some problems, although their relevancy might be low. However, if a case base is very small, then this result may give some useful hint for choosing the most similar problems.

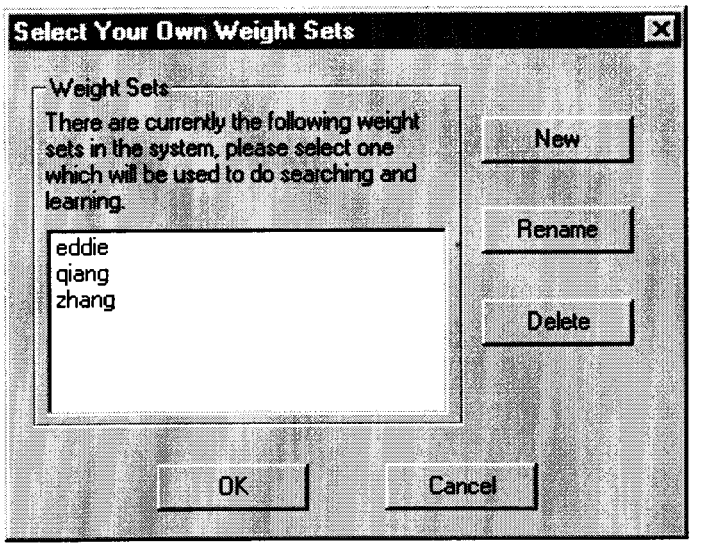

Figure 5.9: A User's Own Weights

As shown in Figure 4.4, the two sets of weights – one attached to the associations between the question-answer pairs and the problems, and the other attached to the associations between the problems and the solutions in a case base – are now not transparent to the users(in contrast to that in the static weight adjusting component).

In Figure 5.8, the system, according to a user's current problem description and a set of selected question-answer pairs(label 1), will access the weights(label 2) to compute the problem scores, and present the result to the user for her/his judgement(label 3). If the user feeds back her/his judgement to the system(label 4), the system will use the learning component to learn this information, and if necessary, modify the weights accordingly(label 7). For the problem which is confirmed positively, the scores of its associated solutions will be computed(label 2), and be presented to the user for further judgement(label 5). If at this time the user feeds back some information to the system(label 6), the system will again use the learning component to learn it and modify the corresponding weights(label 7). It can be easily seen that this process is in accordance with the learning algorithm discussed in Chapter 4.

We now demonstrate the implementation of this algorithm from an angle of user-interface. When a user enters the description of new problem and answers a set of questions, the problem scores will be computed. They will be ranked according to their scores with the problems having higher scores placed at the beginning. If a user chooses a problem, the

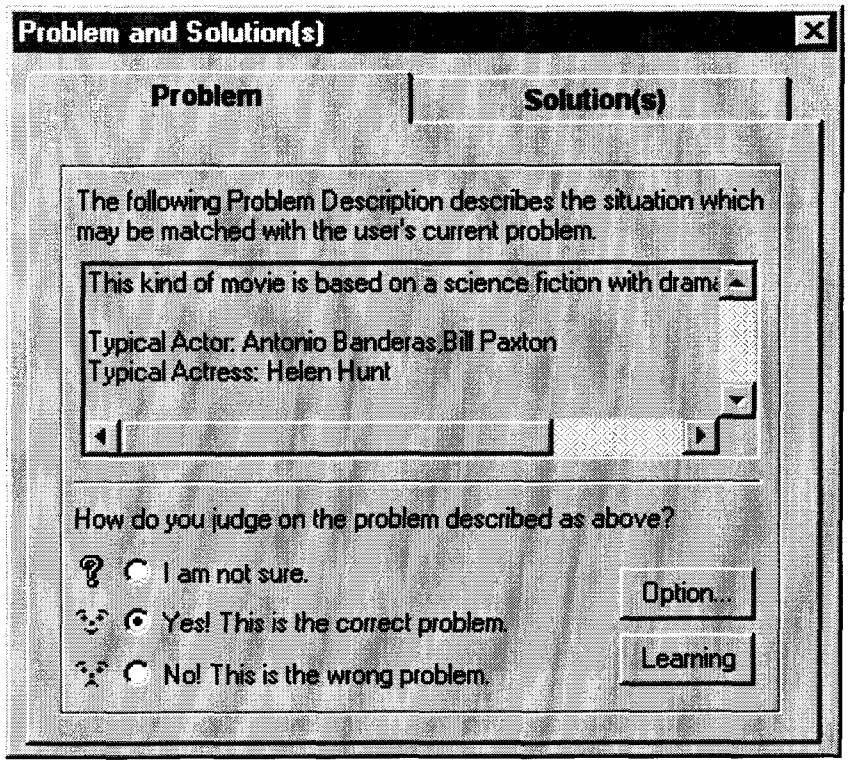

Figure 5.10: Problem's Confirmation or Disapproval

system will prompt a box shown in Figure 5.10 to let her/him decide how s/he judges it. Is it the one s/he desires, the one s/he does not want, or the one that s/he is not sure right now? This information will be fed back to the system for the corresponding adjustments.

After a problem is positively confirmed, the user can take a look at its associated solutions, which are also ranked by the scores computed using the formulas in Chapter 4. In Figure 5.11, for each solution, again the user has a chance to decide whether it is correct, it is wrong, or it is not decided yet. The system will learn from this judgement to adjust its weights.

In Figures 5.10 and 5.11, there are two button. One is labelled *Option...* which can be used by a user to specify explicitly the desired score of a problem or a solution. The other is labelled *Learning* which will be used to trigger the learning process for the current desired score and actual score of a problem or a solution. Thus whether to learn the current judgement of a problem or a solution is still up to the user.

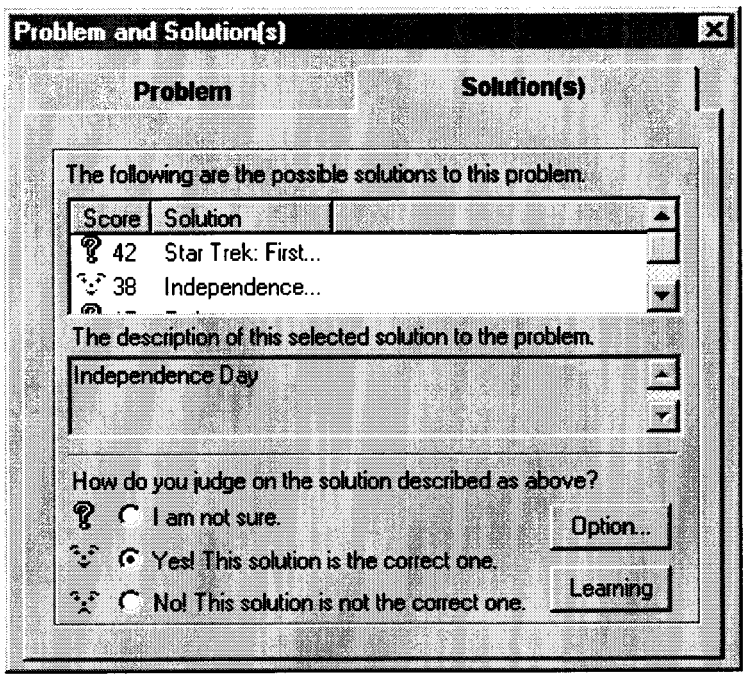

Figure 5.11: Solution's Confirmation or Disapproval

5.4 Summary

The static weight adjusting algorithm is mainly implemented in the Case-Authoring Module, while the dynamic weight learning algorithm is constructed in the Problem-Resolution Module. This is in accordance with the different sources of information on which these two algorithms are based. The information for the static weight adjusting algorithm is from a case base itself, while the information for the dynamic weight learning algorithm is from the end-users. The user-interfaces presented in this chapter for these two feature-weighting components are still in their laboratory stage. The feedbacks from the real use of the system will be essential in order to further improve and tune them.

Chapter 6

Empirical Tests

6.1 Introduction

We have introduced two feature-weighting methods into the framework of CBR, and implemented them in the CaseAdvisor™ system. In the implementation of the system, the feature-value pairs are represented by question-answer pairs, which can be specified by the domain experts when they construct a case base, and used by the end-users when they solve new problems.

The static weight adjusting method uses the statistical information hidden in the associations between question-answer pairs and cases. The size of a case base and the number of cases a question-answer pair is associated with are the two factors in determining how much weight should be assigned to that pair. There are no interactions between the system and its end-user. However, the domain experts can specify adjustment parameters based on their domain knowledge and previous experiences. In our adjusting method, the weight adjusting work is done in the Case-Authoring Module.

The dynamic weight learning method learns the unpredictable information hidden in an end-user's behavior. A user's interactions with the system provide the guidance in determining how much the weight for a particular question-answer pair is in order to capture the user's behavior. In our learning method, the weight adjusting and using are interleaved in the Problem-Resolution Module; the weights for question-answer pairs will be adjusted in response to the feedbacks from the end-user, and the adjusted weights, in turn, will be used in the following case-searching processes.

6.2 Test for Static Weight Adjusting Method

As discussed in Chapter 3, the static weight adjusting algorithm is appropriate for a case base in which the importance of each feature-value pair is constant, or globally the same across the whole case base.

We hope to confirm through the experiments that the static weight adjusting method is feasible to compute a weight statically for each question-answer pair. We first use an artificial case base to demonstrate that the static weight adjusting component we design and implement can be employed to take advantage of the statistical information hidden in a case base; we will also show that different adjustment parameters in the adjusting process will result in different weights, which might affect the final score for the same case even when the same question-answer pairs are selected. Then we use a case base in the real world which is from Roger's Cable Company to show that the weights adjusted by our method delineate roughly the weights initially assigned by the domain experts from the company.

6.2.1 Experiment of An Artificial Case Base

Experiment Setup

We create an artificial case base of 75 cases with six questions. There are 21 question-answer pairs within these questions. Because we use question-answer pairs to index the case base, we are not really concerned with the content in each case. We just take a close observation on the associations between the question-answer pairs and the cases.

Table 6.1 lists the distribution information about the number of associated cases(N.A.C) for each question-answer pair in the case base. From the table we can find that this distribution is not uniform. For question-answer pair QA.9, there are 70 cases associated with it, while for another question-answer pair QA.8, the number of its associated cases is only five. This irregular distribution information reflects the specific domain knowledge for each case conceived in the domain experts' minds when they construct a case base. It is also the information we need to grasp in the static weight adjusting method.

We also need a set of adjusting parameters for the adjustment work. Recall in Chapter 3, we require the domain experts to specify, in their minds, how to judge whether a question-answer pair belongs to *very good*, *good*, *average*, *bad*, or *very bad* group, and meanwhile to specify their corresponding minimum weights, according to the size of the case base, as well as their own domain knowledge and experiences. Although we preset these parameters

QA Pairs	N.A.C.	QA Pairs	N.A.C.
QA.1	37	QA.12	22
QA.2	48	QA.13	59
QA.3	20	QA.14	29
QA.4	24	QA.15	6
QA.5	13	QA.16	21
QA.6	14	QA.17	19
QA.7	43	QA.18	10
QA.8	5	QA.19	28
QA.9	70	QA.20	39
QA.10	41	QA.21	53
QA.11	33		

Table 6.1: Distribution of Cases among Question-Answer Pairs

Groups	Very Good	Good	Average	Bad	Very Bad
Percentage Ranges	1%-10%	11%-20%	21%-40%	41%-60%	61%-100%
Number of Cases	1-7	8-15	16-30	31-45	46-75
Minimum Weights(1)	0.8	0.6	0.5	0.3	0
Minimum Weights(2)	0.9	0.8	0.6	0.4	0.1

Table 6.2: Percentage Ranges and Minimum Weights for Two Tests

based on the size of a case base in the system, they can still be respecified by the domain experts to show their own views on how goodness a question-answer pair is.

We hope that the selections of a small number of question-answer pairs would result in the desired cases we want. Thus, reasonably, the very good group will correspond to a small portion of the case base. By doing so, only a small number of question-answer pairs will be falling into this group, and if they are selected in the case-searching process, only the scores of a small number of cases will change. For the same reason, the very bad group will correspond to a large portion of the case base because the question-answer pairs falling into that group convey little information in the search process.

We show in Table 6.2 the percentage ranges of each group and their corresponding minimum weights for our test.

Q-A Pairs	Groups	Weight	Q-A Pairs	Groups	Weight
QA.1	bad	0.41	QA.12	average	0.55
QA.2	bad	0.27	QA.13	very bad	0.16
QA.3	average	0.57	QA.14	average	0.50
QA.4	average	0.54	QA.15	very good	0.83
QA.5	good	0.65	QA.16	average	0.56
QA.6	good	0.62	QA.17	average	0.57
QA.7	bad	0.32	QA.18	good	0.74
QA.8	very good	0.86	QA.19	average	0.50
QA.9	very bad	0.05	QA.20	bad	0.38
QA.10	bad	0.32	QA.21	very bad	0.22
QA.11	bad	0.47			

Table 6.3: First Adjusting Result for Each Question-Answer Pair

Cases	Q-A Pairs Associated
Case2	QA.1, QA.8, QA.9, QA.11, QA.16, QA.13, QA.19
Case8	QA.1, QA.8, QA.9, QA.10, QA,12, QA.13, QA.15, QA,17, QA.21
Case20	QA.1, QA.2, QA.7, QA.9, QA.10, QA,13, QA,15, QA18, QA.20, QA.21
Case34	QA.1, QA.2, QA.4, QA.5 QA.9, QA.14, QA.17, QA.19, QA.21

Table 6.4: Typical Cases and Their Associated Q-A Pairs

Experiment Result

- **A First Experiment**

  We show in Table 6.3 the adjusting result using the first set of minimum weights specified in Table 6.2. From the table we can see that since question-answer pair QA.8 belongs to the very good group, it is assigned at least the minimum weight for that group. Another example shows that as QA.12 belongs to the average group, it is assigned 0.55, more than the minimum weight of the average group but less than that of the good group.

  If we graph the data in this table in the same way as that in Figure 3.3 in Chapter 3, it can be easily found that they share the same distribution pattern of the weights for the feature-value pairs or question-answer pairs.

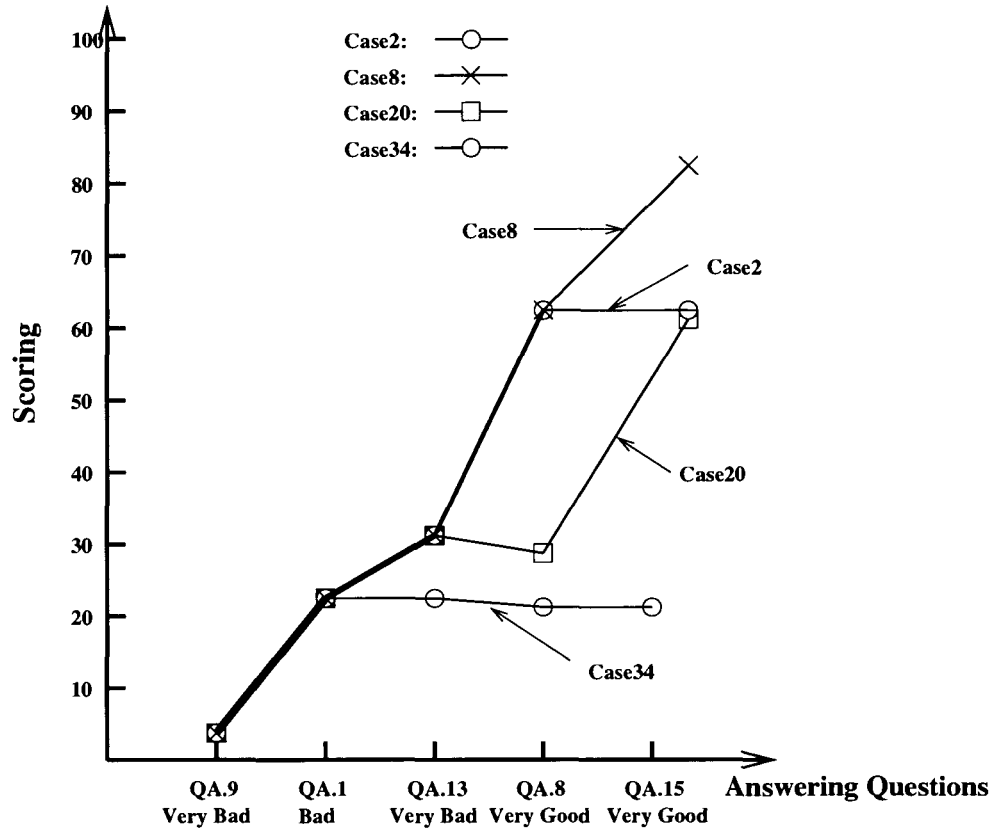

Figure 6.1: Cases-Scoring Process of First Test

In our first experiment, we examine a number of cases. The adjusted weights reflect our views discussed in Chapter 3. As an example, we selected four cases from the case base as shown in Table 6.4 with their associated question-answer pairs.

We check the case-scoring process for these four cases when some questions are answered. In Figure 6.1, the X-axis represents the process of answering questions, while the Y-axis represent the case-scoring process. We first select QA.9. As this question-answer pair belongs to the very bad group, it cannot contribute much to the case scores. From the figure, we can see that Case2, Case8, Case20, and Case34 are all scored 3 out of 100. For the same reason the selection of QA.1, which belongs to the bad group, still does not contribute much to all the case scores. The selection of QA.13 will increase the scores of Case2, Case8, and Case20 a little bit, since QA.13 is also a member in the very bad group. Case34 remains unchanged since it is not

Q-A Pairs	Groups	Weight	Q-A Pairs	Groups	Weight
QA.1	bad	0.51	QA.12	average	0.71
QA.2	bad	0.37	QA.13	very bad	0.26
QA.3	average	0.74	QA.14	average	0.61
QA.4	average	0.68	QA.15	very good	0.91
QA.5	good	0.82	QA.16	average	0.72
QA.6	good	0.81	QA.17	average	0.75
QA.7	bad	0.42	QA.18	good	0.87
QA.8	very good	0.93	QA.19	average	0.61
QA.9	very bad	0.15	QA.20	bad	0.48
QA.10	bad	0.42	QA.21	very bad	0.32
QA.11	bad	0.51			

Table 6.5: Second Adjusting Result for Each Question-Answer Pair

associated with QA.13. However, when we select QA.8, a member of the very good group, the scores of Case2 and Case8 migrate dramatically from 31 to 62. Because Case20 is not associated with that pair, so its score remains almost the same. Selection of QA.15, which belongs to the very good group, is very interesting; it makes two cases change their scores dramatically. In this situation, Case8 migrates from 62 to 82, while Case20 migrates from 29 to 61. Because Case2 is not associated QA.15, its score does not change.

When the case-scoring process ends, Case8 ranks first, with Case2 and Case20 in the second place and third place. Obviously, because this answering process involves two very good question-answer pairs, their associated cases change the scores dramatically. On the other hand, because Case34 is not associated with these pairs, its score changes little. We can also find this for Case20. From the selection of QA.9 to the selection of QA.8 along the X-axis, its score changes steadily, from high to low or from low to high. Only after its associated pair QA.15 is selected does its score change greatly.

- **A Second Experiment**

Different percentage ranges and minimum weights will result in different weights, affecting the case scores in the case retrieval process. We use the second set of minimum weights specified in Table 6.2 for our second experiment. The percentage ranges for

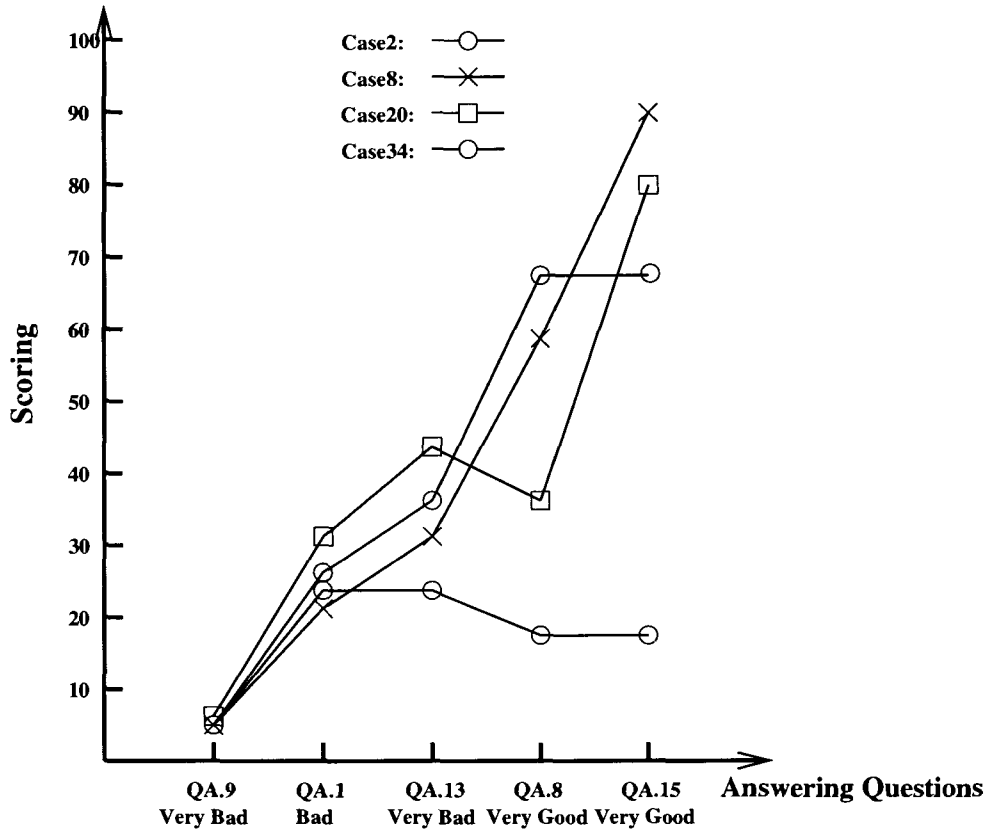

Figure 6.2: Case-Scoring Process of Second Test

each group is still the same. It can be seen from the table that we increase the minimum weights for each group. Therefore, the weights, once adjusted, will have higher values than those in the first experiment.

The result is shown in Table 6.5. To examine those weights, we also use them to test the typical cases specified in Table 6.4. Figure 6.2 is the scoring process. Again it can be easily seen that the question-answer pairs belonging to the very good group are responsible for the dramatic changes of the case scores. From the figure, we can see that, because of the higher weight of each question-answer pair, the four cases start to separate from each other from the selection of QA.1. When we check the case scoring process for all the four cases during the selections of question-answer pairs, it can be seen that they all have similar patterns to those in Figure 6.1.

However, there is one exception in the figure. Note that Case20 jumps from the score

Q-A Pairs	N.A.C.	Initial Weight	Q-A pairs	N.A.C.	Initial Weight	Q-A Pairs	N.A.C.	Initial Weight
QA.1	26	0.10	QA.11	1	0.95	QA.21	2	0.41
QA.2	1	1.00	QA.13	1	0.82	QA.22	3	0.56
QA.3	1	1.00	QA.14	1	0.90	QA.24	1	0.80
QA.4	25	0.11	QA.15	1	0.95	QA.25	1	0.90
QA.5	1	0.99	QA.16	3	0.73	QA.26	1	0.75
QA.7	2	0.63	QA.17	2	0.90	QA.27	1	0.75
QA.8	1	0.80	QA.18	8	0.70	QA.28	1	0.85
QA.9	1	0.90	QA.19	1	0.90	QA.29	1	0.80
QA.10	5	0.61	QA.20	1	0.90	QA.30	1	0.80

Table 6.6: Distribution Information in Roger's Cable Case Base

of 36 to 80 when QA.15 is selected, ranking itself before Case2. In contrast, in Figure 6.1, at this point, Case2 ranks higher than Case20. We attribute this to the higher weight of QA.15 than that in the first experiment. From here we can see that the specifications of different percentage ranges and their corresponding minimum weights will sometimes result in different case scores. Although we can see this as the different views of different domain experts, however, sometimes it is undesirable. Therefore we think that the static weight adjusting process is a repetitive work, involving alternating between the adjusting and testing processes in order to get a better weight quality.

6.2.2 Experiment for Roger's Cable Case Base

Experiment Setup

The case base which is being used in Roger's Cable Company is created using the Case-Authoring Module in the CaseAdvisor™ system. This case base is used by the technical representatives of the company to solve the customers' problems on the help desk. Up to now, this case base has collected 28 cases and five features or questions. Within the five questions, there is a total of 30 question-answer pairs. We label each question-answer pair as QA.i, where $i = 1, 2, \cdots, 30$. Table 6.6 shows all the question-answer pairs and their number of associated cases(N.A.C.). Also shown the table is the weights assigned initially to the individual question-answer pairs by the domain experts from the company. As question-answer pairs QA.6, QA.12, and QA.23 are not associated with any case, based on the discussions in Chapter 3, we will not consider them in the static weight adjusting

Groups	Very Good	Good	Average	Bad	Very Bad
Percentage Ranges(1)	1%-10%	11%-30%	31%-50%	51%-70%	71%-100%
Number of Cases(1)	1-2	3-8	9-14	15-19	20-28
Minimum Weights(1)	0.70	0.60	0.50	0.30	0
Percentage Ranges(2)	1%-20%	21%-30%	31%-50%	51%-70%	71%-100%
Number of Cases(2)	1-5	6-8	9-14	15-19	20-28
Minimum Weights(2)	0.80	0.60	0.50	0.30	0.05
Percentage Ranges(3)	1%-10%	11%-30%	31%-50%	51%-70%	71%-100%
Number of Cases(3)	1-2	3-8	9-14	15-19	20-28
Minimum Weights(3)	0.70	0.60	0.50	0.30	0.05

Table 6.7: Three Sets of Adjustment Parameters for Roger's Cable Case Base

process.

Of all the cases, QA.1 is associated with 26 out of them, and all the associations are attached a weight of 0.10. The same situation is true for QA.18, all of whose associated cases have a weight of 0.70. QA.4 has the same weight 0.10 for 24 out of its associated 25 cases. The exception one is attached with 0.20. This demonstrates that in some situation it is reasonable for the weight for a question-answer pair to be constant for all of its associated cases.

For a question-answer pair, if the weights attached to its associations of cases are not equal, we will average them in order to compare them with the weights assigned by the static adjusting method. For instance, QA.10 is associated with five cases with the weights of 0.90, 0.70, 0.30, 0.40, and 0.75. The average weight for this question-answer pair is 0.61.

We use three sets of adjustment parameters to do our tests. These parameters are shown in Table 6.7.

Experiment Result

We show the adjustment result in Table 6.8. If we sort the initial weights for all the question-answer pairs from low to high, we can see from the table that this relative order is maintained in the first and third sets of adjusted weights. For instance, the initial weight for QA.1 is less than that for QA.7, whose initial weight, in turn, is less than that for QA.11. This order is preserved in their corresponding adjusted weights. In the second set of adjusted weights, the weights for QA.21 and QA.22 are reversed from their positions in the initial weights.

Q-A Pairs	Initial Weight	Weight (1)	Weight (2)	Weight (3)
QA.1	0.10	0.07	0.11	0.11
QA.2	1.00	1.00	1.00	1.00
QA.3	1.00	1.00	1.00	1.00
QA.4	0.11	0.11	0.14	0.14
QA.5	0.99	1.00	1.00	1.00
QA.7	0.63	0.70	0.95	0.70
QA.8	0.80	1.00	1.00	1.00
QA.9	0.90	1.00	1.00	1.00
QA.10	0.61	0.66	0.80	0.66
QA.11	0.95	1.00	1.00	1.00
QA.13	0.82	1.00	1.00	1.00
QA.14	0.90	1.00	1.00	1.00
QA.15	0.95	1.00	1.00	1.00
QA.16	0.73	0.70	0.90	0.70
QA.17	0.90	0.70	0.95	0.70
QA.18	0.70	0.60	0.60	0.60
QA.19	0.90	1.00	1.00	1.00
QA.20	0.95	1.00	1.00	1.00
QA.21	0.41	0.70	0.95	0.70
QA.22	0.56	0.70	0.90	0.90
QA.24	0.80	1.00	1.00	1.00
QA.25	0.90	1.00	1.00	1.00
QA.26	0.75	1.00	1.00	1.00
QA.27	0.75	1.00	1.00	1.00
QA.28	0.85	1.00	1.00	1.00
QA.29	0.80	1.00	1.00	1.00
QA.30	0.80	1.00	1.00	1.00

Table 6.8: Adjustment Results for Three Sets of Adjustment Parameters

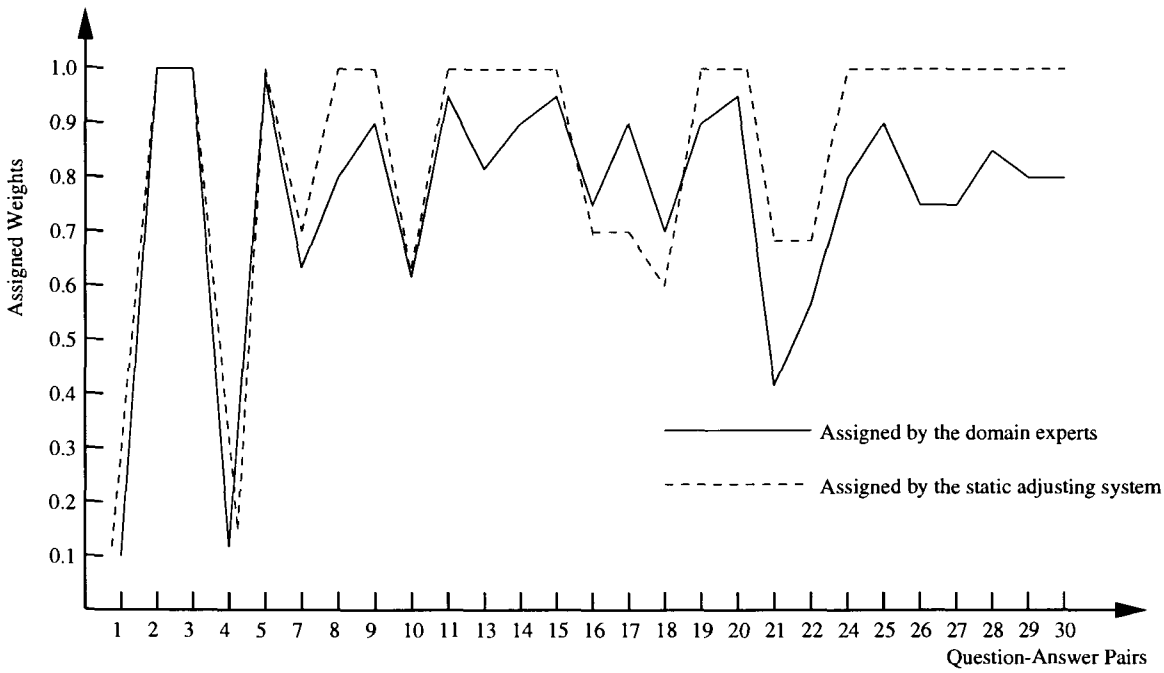

Figure 6.3: Adjusted Weights vs. Initial Weights for Roger's Cable Case Base

All the other pairs in that adjustment maintain the relative order.

In Figure 6.3, we graph the initial weights assigned by the domain experts from the company and the adjusted weights assigned by our static adjustment method using the third set of the adjustment parameters. In the figure, the X-axis represents the question-answer pairs. For space's sake, we just label each question-answer pair as 1,2,$\cdots$,30. Because QA.6, QA.12 and QA.23 are not associated with any case, so we omit them. The Y-axis represents the weights ranging from 0.00 to 1.00. From the figure, it can be clearly seen that the curve of the adjusted weights approximates the curve of the initial weights at more than half of the points(question-answer pairs). Of the 27 question-answer pairs, the adjusted weights of 15 pairs out of them are very close to their initial counterparts, while the remaining 12 pairs maintain their relative order as discussed above.

However, some question-answer pairs which have different initial weights now get the same weights. For instance, the weights for the question-answer pairs which are just associated with one case are now all assigned with the same weight of 1.00. In our static adjusting method such a question-answer pair belongs to the very good group. This phenomenon is obvious in the above figure in the last several question-answer pairs. Although they have

different initial weights, they all have the same adjusted weight, with no one being more distinguishable(Their relative order is still the same). This is in accordance with what we have predicted in Chapter 3.

We think that the most important argument for such a static adjusting method is that the relative order is the same for both the initial weights and the adjusted weights. From this experiment result, it can be seen that our method fulfills the goal.

6.2.3 Repetitive Adjusting

It is difficult for the parameters in the static weights adjusting component to be accurate when they are first specified. This requires that the static weight adjusting algorithm be a repetitive process, in which the domain experts' knowledge and previous experiences will be applied. Adjusting and testing processes maybe repeat alternatively several times before the final adjusted case base is put into the practical use.

6.2.4 Running Time for Static Weight Adjusting Method

In the practical implementation, the cases and the question-answer pairs in a case base are usually not fully associated. If we save the information beforehand about the associations between cases and question-answer pairs (represented as $(question, answer)$) as a set of triples $(case, question, answer)$ along the construction of a case base, then the time spent is linearly proportional to the size of this set. This time is less than $O(N * M)$ discussed in Chapter 2. In our experiment, the size of the triple set in the artificial case base is 634, which is less than half of $75 * 21 = 1575$. For the Roger's Cable case base, our method takes about 10 seconds to finish the adjustment work.

6.3 Test for Dynamic Weight Learning Method

As discussed in Chapter 4, the dynamic weight learning method will incorporate a user's behavior and preferences into the weights attached to the associations between the feature-value pairs and the cases, and between the cases and the their associated solutions in a case base.

After integrated with a backpropagation neural network as a learning component, a CBR system is a responsive system which will be changing with its environment. We hope

to confirm through the experiments that our dynamic weight learning method is feasible to adjust the weights in a CBR system. A user's preferences can be learned by our system in an efficient and timely fashion. Besides, the learning quality is optimal, approximating a user's preferences gradually. Furthermore, the system will change over time in response to its user's changes

6.3.1 A User's Preference

In the practical use of a CBR system, when a new problem is encountered, a user will extract its features, and use them to search in the case base. Thus, for a new problem, there is a set of features corresponding to it.

Let us further consider from another angle about this set of features. After a user inputs a set of features into a CBR system, s/he desires to get a set of most similar cases with the scores ranking from high to low. Thus there is a correspondence between a set of features and cases and their associated scores. For a different new problem, there is a different set of features. Even for the same set of features, different users may have different views(i.e. weights), thus resulting in different case scores. We can consider that a set of features form the context in which a set of cases and their scores will be produced.

For a new problem in a domain, there is always a set of appropriate features which can be used to describe it. However, not every set of features can represent a new problem, since maybe there is some inconsistent and contradictory information in it. For instance, two particular feature-value pairs might be mutually exclusive. Take the small case base in Table 2.1 as an example, the feature-value pair (job status, salaried) and (job status, waged) is mutually exclusive.

Using the same convention as that in Chapter 4, in order to emphasize that we have separated the problem solution from the problem description in a case, we use *problem description* or just *problem* instead of *case* in the following discussions. This is also in accordance with the three-layer structure we introduce in Chapter 4.

Let Q be the set of feature-value pairs in a case base. Let Q_v be a subset of 2^Q, where 2^Q is the power set of Q, and in Q_v, each member is a valid combination of feature-value pairs in the case base. For instance, if $Q = \{(f_1, v_1), (f_2, v_2), (f_3, v_3)\}$, and the feature-value pairs (f_1, v_1) and (f_2, v_2) are mutually exclusive, then the set Q_v will be $\{\{(f_1, v_1)\}, \{(f_2, v_2)\}, \{(f_3, v_3)\}, \{(f_1, v_1), (f_3, v_3)\}, \{(f_2, v_2), (f_3, v_3)\}\}$. Let P be the set of all the problems in the case base. Let S be the set of all the solutions in the case base. Let

$I = \{1, 2, 3, \cdots, 100\}$ represents the possible ranking scores of a problem or a solution. In the following, we define a user's preference in a case base.

Definition *In a case base, a user's preference is defined as two functions. The first function is called p-function shown as follows:*

$$p : Q_v \longrightarrow 2^{P \times I}$$

The second function will be executed after p-function. It is called s-function:

$$s : Q_v \times P \longrightarrow 2^{S \times I}$$

We can see from the above definition that a user's preference is composed of two parts(p and s) represented by two functions. p-function takes a valid set of feature-value pairs as its input and produces a set of problems and their associated ranking scores. After this, the user may select a promising problem for further consideration. s-function takes the same valid feature-value pair set and this selected problem as its input and produces a set of solutions and their associated ranking scores.

Let us take a look at a simple example for this definition. Using the same feature-value pair set Q in the previous example, assume that $P = \{p_1, p_2, p_3\}$, and $S = \{s_1, s_2\}$. Given a valid feature-value pair set $q_v = \{(f_1, v_1), (f_3, v_3)\}$ in Q_v, $p(q_v)$ might produce $\{(p_1, 70), (p_2, 50)\}$. If problem p_1 is selected for further consideration and solution s_1 is associated with p_1, then $s((q_v, p_1))$ might produce $\{(s_1, 70)\}$.

From this viewpoint, the neural network learning component we introduce into a CBR system is attempting to simulate these two functions in its weights. It takes a user's preference and outputs a set of problems, and a set of solutions afterwards. If the user is not satisfied with the output, s/he can feed back her/his response to the system, which, in turn, will change its weights to reflect the response.

6.3.2 Experiment for Movie Case Base

Experiment Setup

The CBR Group at SFU participated in the Open House sponsored by SFU in 1996, and there they demonstrated their product CaseAdvisor™ using a movie case base. The case base consists of problems(cases) with each representing a movie. A user can input some keywords about the movie s/he desires and answer some questions such as *what type of*

movie do you like?, *who is your favorite actor?* and *etc.*. The system will retrieve a set of possible movies for the user to choose.

In order to do our experiment, we restructure this case base. Right now, each problem represents a specific type of movie, such as *comedy movie*, *science fiction movie*, *family movie*, *action movie*, and *etc.*. For each type of movie there are several possible solution movies which belong to that specific type. The question-answer pairs(feature-value pairs) are the same as before. Thus, it can be seen that the whole case base, once restructured, is a three-layer hierarchy with the question-answer pairs at the bottom layer, the problems at the middle layer, and the solutions at the top layer.

There are 25 question-answer pairs $QA.1$,$QA.2$,$\cdots$,$QA.25$, ten problems or types of movie $P1$,$P2$,$\cdots$,$P10$, and 15 solution movies $S1$,$S2$,$\cdots$,$S15$ in the case base. In our experiment, all the weights are initialized to 0.5.

A First Experiment

The first experiment is designed for the demonstration of a situation where a user is satisfied with all the problem retrieval results except one. The user's preference contains only one valid set of question-answer pairs. It can be learned very quickly. Recall in Chapter 4 that we discuss that the weights for a problem are local, independent of the interactions from other problems in the dynamic weight learning component.

- **Experiment for Desired Problem Scores**

  The experimental data is shown in Table 6.9. The question-answer pairs involved in this experiment are QA.5, QA.7, QA.12, and QA.20, while the problems involved are P2, P3, P4, P5, P6, P7, P8, P9, and P10. Problem P1 is not included since it is not associated with any of the four question-answer pairs. This is in accordance with Formulas 4.11, and 4.17. However, if a user does specify such a problem in her/his valid set, our system will detect and report it.

  Before the learning, the retrieval result for these problems from the system is shown in Table 6.10. After the first round through all the problems' desired scores specified Table 6.9, the retrieval result is shown in Table 6.11. From the table, we can find that because there is no interactions among problems, our system quickly brings the retrieval result into the optimal state we desire.

Q-A Pairs	Problems & Scores	Problem & Scores	Problem & Scores
QA.5	$P2 = 55$	$P6 = 30$	$P10 = 70$
QA.7	$P3 = 95$	$P7 = 50$	
QA.12	$P4 = 20$	$P8 = 20$	
QA.20	$P5 = 40$	$P9 = 40$	

Table 6.9: A User's Preference(Part p)

Q-A Pairs	Problems & Scores	Problems & Scores	Problems & Scores
QA.5	$P2 = 56$	$P6 = 34$	$P10 = 73$
QA.7	$P3 = 86$	$P7 = 56$	
QA.12	$P4 = 34$	$P8 = 34$	
QA.20	$P5 = 34$	$P9 = 34$	

Table 6.10: Initial Problem Retrieval Result

- **Experiment for Desired Solution Scores**

  Next we will try to adjust the weights attached to the connections between problems and their associated solutions as shown in a user's preference(part s). Under this situation, we do not care too much about the actual scores of the problems, we focus our attention on the final solutions of those promising problems. In practice, we would like to choose the problems retrieved with higher scores for further analysis. However, we still need to maintain the order of their ranking positions in order to show their relative importance in the user's preference.

  Table 6.12 shows the solutions associated with the four problems which are at the first four highest positions in Table 6.9.

  In this experiment, we take each problem in turn. We first select problem $P3$, and try to adjust the scores of its associated solutions. Then we process problem $P10$, after that we take problem $P7$, and the last problem we take is $P2$. This forms one training round. There is a total of 4 rounds during our training in this experiment.

  After each round, we sample the scores of the solutions associated with all the four problems. We show the initial solution retrieval result, and the corresponding retrieval

Q-A Pairs	Problems & Scores	Problems & Scores	Problems & Scores
QA.5	$P2 = 55$	$P6 = 30$	$P10 = 70$
QA.7	$P3 = 93$	$P7 = 50$	
QA.12	$P4 = 20$	$P8 = 20$	
QA.20	$P5 = 39$	$P9 = 39$	

Table 6.11: Problem Retrieval Result after First Round

Problems	Solutions & Scores
$P3$	$S4 = 95$, $S9 = 95$, $S14 = 70$
$P10$	$S11 = 90$,$S1 = 80$
$P2$	$S3 = 90$, $S6 = 70$, $S14 = 60$
$P7$	$S3 = 80$, $S6 = 90$, $S7 = 70$, $S14 = 40$

Table 6.12: A User's Preference(Part s)

result after each round until round 4 in Table 6.13. We can see from the table that after four rounds of adjustments, the final solution scores are getting to the desired scores.

We can find some interesting information from the table. Solutions $S11$ and $S1$ are only associated with problem $P10$, so their desired scores are reached faster than other solutions. We attribute this to the fact that there are no other solutions interacting with them. Solution $S14$ belongs to problems $P3$, $P2$ and $P7$. However, it has different scores in these cases. A more interesting observation is made between solutions $S3$ and $S6$. These two solutions belong to problems $P2$ and $P7$. But their ranking position within these two problems are totally reversed. Solution $S3$ is higher in problem $P2$ while solution $S6$ is higher in problem $P7$. All these are in accordance with the prediction we make in Chapter 4.

After four rounds of adjustments, we also take a look at the relative positions of the four problems. The scores of these four problems have been changed. But they still occupy the first four positions within all the problems with their relative positions unchanged.

	$P3$	$P10$	$P2$	$P7$
Initial	$S4 = 96$ $S14 = 97$ $S9 = 96$	$S11 = 90$ $S1 = 88$	$S3 = 87$ $S14 = 89$ $S6 = 86$	$S6 = 83$ $S7 = 83$ $S3 = 85$ $S14 = 88$
Round 1	$S4 = 96$ $S14 = 70$ $S9 = 96$	$S11 = 90$ $S1 = 80$	$S3 = 89$ $S14 = 62$ $S6 = 72$	$S6 = 89$ $S7 = 69$ $S3 = 79$ $S14 = 40$
Round 2	$S4 = 94$ $S14 = 69$ $S9 = 94$	$S11 = 90$ $S1 = 80$	$S3 = 89$ $S14 = 60$ $S6 = 70$	$S6 = 89$ $S7 = 70$ $S3 = 80$ $S14 = 40$
Round 3	$S4 = 94$ $S14 = 67$ $S9 = 94$	$S11 = 90$ $S1 = 80$	$S3 = 89$ $S14 = 56$ $S6 = 73$	$S6 = 89$ $S7 = 70$ $S3 = 80$ $S14 = 40$
Round 4	$S4 = 94$ $S14 = 69$ $S9 = 94$	$S11 = 90$ $S1 = 80$	$S3 = 89$ $S14 = 59$ $S6 = 70$	$S6 = 89$ $S7 = 70$ $S3 = 80$ $S14 = 40$

Table 6.13: Training Process of Solutions

A Second Experiment

- **Experiment for Desired Problem Scores**

  In this experiment, a user's preference includes several valid sets of question-answer pairs in the learning process. Different from the last experiment, the interactions between different problems will make the learning process longer, which is predicted in the learning algorithm shown in Chapter 4. The experimental data is shown in Table 6.14. There are 4 valid sets of question-answer pairs in the table.

  In our training, we take each valid set in turn. *Valid set 1* is first input to the system, and its problems and their scores will be learned. Then *valid set 2*, *valid set 3*, and *valid 4 set* will be learned one by one. This forms one training round. In this experiment, the training process is composed of five rounds.

	Q-A Pairs	Problems & Scores	Problems & Scores	Problems & Scores
Valid Set 1	QA.1 QA.7 QA.9	$P1 = 40$ $P2 = 40$ $P3 = 70$ $P4 = 20$	$P5 = 95$ $P6 = 40$ $P7 = 40$ $P8 = 10$	$P9 = 60$ $P10 = 60$
Valid Set 2	QA.3 QA.7 QA.11 QA.22	$P3 = 20$ $P4 = 95$ $P6 = 20$ $P7 = 20$	$P8 = 80$ $P9 = 40$ $P10 = 60$	
Valid Set 3	QA.5 QA.7 QA.20	$P2 = 60$ $P3 = 95$ $P4 = 20$ $P5 = 45$	$P6 = 10$ $P7 = 55$ $P9 = 45$ $P10 = 20$	
Valid Set 4	QA.6 QA.9 QA.20	$P2 = 95$ $P3 = 85$ $P3 = 85$	$P10 = 50$	

Table 6.14: A User's Preference(Part p)

We use two data-sampling procedures.

1. For the first four rounds, after each round finishes, we input the four valid sets into the system one by one again. But this time, we do not require the system to learn. We just take a look at the problem retrieval result for each valid set. We sample the retrieval result after each round. Thus, there are four such result data sets, each of which is composed of the problem scores under each valid set. They are result data set 1, 2, 3, 4.

2. For round 5, we take the other data-sampling procedure. After *valid set 1* is presented to the system, and its problems and their scores are learned, we input other three preferences to the system but do not trigger the learning process. We take a look at the retrieval results for all these 4 valid sets, and sample them. When this finishes, we take *valid set 2* and repeat the same procedure. The same procedure then is applied to *valid set 3* and *valid set 4*. Thus, when round 5 finishes, we also get four result data sets. They are result data set 5, 6, 7, 8.

Result Data Set 1	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Valid Set 1	39	43	75	20	94	25	48	10	45	39
Valid Set 2			40	91		10	26	74	40	46
Valid Set 3		63	94	20	44	10	57		45	24
Valid Set 4		94	84				74			49
Result Data Set 5	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Valid Set 1	39	40	69	20	94	39	39	10	69	59
Valid Set 2			23	94		11	20	79	43	58
Valid Set 3		61	94	20	44	11	54		69	22
Valid Set 4		94	86				75			49
Result Data Set 6	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Valid Set 1	39	40	67	20	94	46	39	10	68	60
Valid Set 2			20	94		19	20	79	40	59
Valid Set 3		61	93	20	44	19	54		68	23
Valid Set 4		94	86				75			49
Result Data Set 7	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Valid Set 1	39	40	67	20	94	38	39	10	45	59
Valid Set 2			21	94		10	20	79	11	58
Valid Set 3		60	94	20	44	10	54		45	20
Valid Set 4		93	86				75			48
Result Data Set 8	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Valid Set 1	39	41	67	20	94	38	39	10	45	59
Valid Set 2			21	94		10	20	79	11	58
Valid Set 3		60	94	20	44	10	54		45	21
Valid Set 4		94	85				75			49

Table 6.15: Problem Retrieval Result after Five Rounds

Thus we obtain a total of eight retrieval result data sets. For simplicity, we show only five of them in Table 6.15. They are result data set 1, 5, 6, 7, and 8.

We look at each problem in turn in Table 6.15. It can be found that of the ten problems, eight problems $P1$, $P2$, $P3$, $P4$, $P5$, $P7$, $P8$, and $P10$ satisfy our designated scores. Their actual scores are tending to approximate around the desired scores at small distances. We take problem $P3$ as an example. We sample its 8 scores for each valid set in the eight result data sets during the five training rounds. Its scoring process is plotted in Figure 6.4.

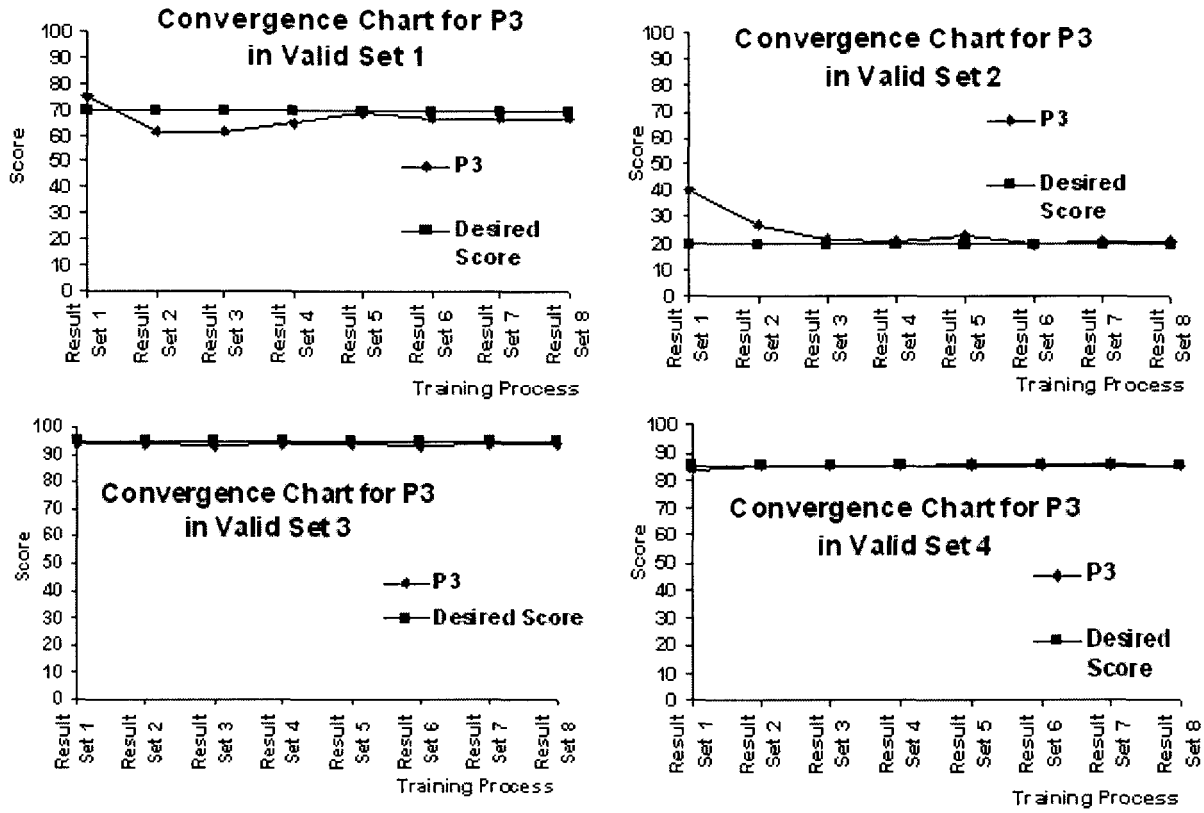

Figure 6.4: Training Process for Problem $P3$

In the figure, the scoring process in each valid set is represented by a small graph, where the X-axis represents the No. of the result data set, and the Y-axis represents the rough score of a problem. It can be seen that scoring process of problem $P3$ is around its desired score at a small distance for all the four valid sets.

The two abnormal problems are $P6$ and $P9$. Problem $P6$ satisfies *valid set 1* with a score close to the desired score of 40. However, it is abnormal for *valid set 2*. Of all the question-answer pairs in the four valid sets, problem $P6$ is associated only with QA.1 and QA.7. QA.7 only appears in valid sets 2 and 3. So problem $P6$ should get the same scores in these two sets. However, as indicated in these two sets, the scores for problem $P6$ are different from each other. So only within these four valid sets, it is not possible to get the different scores of problem $P6$ in *valid set 2* and *valid set 3*. The same argument applies to problem $P9$. Problem $P9$ is associated with QA.3 and QA.7. QA.7 is in valid sets 1, 2, and 3 while QA.3 is only in *valid set 2*. Therefore,

	Problems	Solutions & Scores
Valid Set 1	$P5$	$S4 = 95$, $S6 = 80$, $S8 = 60$, $S9 = 40$
	$P3$	$S9 = 90$, $S4 = 80$, $S14 = 50$
	$P10$	$S11 = 80$, $S1 = 70$
Valid Set 2	$P4$	$S11 = 90$, $S12 = 60$
	$P10$	$S11 = 80$, $S1 = 70$
Valid Set 3	$P2$	$S3 = 90$, $S6 = 60$, $S14 = 60$
	$P3$	$S9 = 95$, $S4 = 90$, $S14 = 50$
Valid Set 4	$P2$	$S3 = 95$, $S6 = 60$, $S14 = 60$
	$P3$	$S9 = 90$, $S4 = 80$, $S14 = 60$
	$P7$	$S3 = 80$, $S7 = 70$, $S6 = 40$, $14 = 20$

Table 6.16: One User's Preference(Part s)

when computing the problem scores, in valid sets 1 and 3, only QA.7 contributes to the score of problem $P9$, so it should get the same scores in these two valid sets, which is not true in the actual valid sets. This gives rise to the abnormality.

We also note that for problem $P6$, it satisfies *valid set 1* since it is associated QA.1, which gives it a chance to adjust its score. This is the result of interactions among the different question-answer pairs in the learning process. We think that with more valid sets of question-answer pairs in a user's preference, the learning quality will become better and better. In a real application, a user might not have a preference consisting of only several valid sets as specified exactly as those in Table 6.14. Different combinations of valid sets interact with each other, contributing to the final optimal retrieval result.

If we just think the problems at the higher ranking positions are promising, then we can see from the table that our learning component produce the optimal retrieval result within only several rounds.

- **Experiment for Desired Solution Scores**

Next we will try to adjust the weights attached to the connections between problems and their associated solutions in this user's preference(part s). Also we do not care too much about the actual scores of the problems, we pay our attention to the final solutions of those promising problems.

Round 1	Problems	Solutions & Scores
Valid Set 1	$P5$	$S4 = 92, S6 = 73, S8 = 71, S9 = 44$
	$P3$	$S9 = 90, S4 = 74, S14 = 20$
	$P10$	$S11 = 80, S1 = 78$
Valid Set 2	$P4$	$S11 = 80, S12 = 93$
	$P10$	$S11 = 87, S1 = 83$
Valid Set 3	$P2$	$S3 = 67, S6 = 37, S14 = 27$
	$P3$	$S9 = 97, S4 = 94, S14 = 42$
Valid Set 4	$P2$	$S3 = 86, S6 = 48, S14 = 39$
	$P3$	$S9 = 86, S4 = 75, S14 = 31$
	$P7$	$S3 = 80, S7 = 70, S6 = 40, 14 = 20$
Round 2	Problems	Solutions & Scores
Valid Set 1	$P5$	$S4 = 95, S6 = 82, S8 = 61, S9 = 57$
	$P3$	$S9 = 85, S4 = 83, S14 = 39$
	$P10$	$S11 = 69, S1 = 63$
Valid Set 2	$P4$	$S11 = 80, S12 = 54$
	$P10$	$S11 = 80, S1 = 69$
Valid Set 3	$P2$	$S3 = 77, S6 = 48, S14 = 37$
	$P3$	$S9 = 96, S4 = 93, S14 = 61$
Valid Set 4	$P2$	$S3 = 86, S6 = 53, S14 = 45$
	$P3$	$S9 = 88, S4 = 77, S14 = 48$
	$P7$	$S3 = 74, S7 = 62, S6 = 39, 14 = 19$
Round 3	Problems	Solutions & Scores
Valid Set 1	$P5$	$S4 = 94, S6 = 82, S8 = 61, S9 = 58$
	$P3$	$S9 = 80, S4 = 80, S14 = 44$
	$P10$	$S11 = 71, S1 = 65$
Valid Set 2	$P4$	$S11 = 83, S12 = 56$
	$P10$	$S11 = 80, S1 = 69$
Valid Set 3	$P2$	$S3 = 86, S6 = 57, S14 = 48$
	$P3$	$S9 = 94, S4 = 91, S14 = 66$
Valid Set 4	$P2$	$S3 = 92, S6 = 62, S14 = 58$
	$P3$	$S9 = 89, S4 = 80, S14 = 61$
	$P7$	$S3 = 79, S7 = 69, S6 = 39, 14 = 19$

Table 6.17: Solution Retrieval Result after Three Rounds

Questions asked technical representative	Answers from Customer
What type of problem is sub experiencing?	Hong Kong TV
Which channels have the problem?	All
Is the problem affecting more than 1 outlet?	No, Only 1 outlet is being affected.
Is the account enabled/in pay status?	Yes

Table 6.18: A User's Query

We apply our learning component to the data shown in Table 6.16. The whole training process is composed of 3 rounds. We sample the solution retrieval result after each round, which takes and learns each valid set one by one. If we only take the relative ranking positions of the solutions in each problem in each valid set, we can find that there is no abnormality within all the solutions in Table 6.17 after only three rounds. Their relative ranking positions observe the ones designated in the user's preference as shown in Table 6.16.

If we set that the range of ± 5 around a desired score of a solution is acceptable, then we find from Table 6.17 that of 29 solutions produced by four valid sets, only seven solutions fall into the unacceptable score range. If we consider that this process only takes three rounds, then such a result is optimal, confirming our hypotheses discussed in Chapter 4 and at the beginning of this section.

6.3.3 Experiment for Roger's Cable Case Base

Experiment Setup

We have briefly discussed the case base from Roger's Cable Company in the test for static weight adjusting method. A customer will call the technical representatives on the help desk in the company. S/he will describe what her/his current problem is. A technical representative will input such a description into the Problem-Resolution Module. Then s/he will select some questions and ask them to the customer. Based on the answers to these questions, the Problem-Resolution Module will retrieve a set of relevant cases and their scores.

As an example, assume that a customer has a problem to watch some channels. S/he can call and tell a technical representative about this. The technical representative will input the

Case 1	
Score	86
Name	Hong Kong TV will not work with VCR or converter
Description	TV set requires re-tuning to accommodate NTSC signal.
Solution	Advise Sub to phone K.S. Video at 876-8320 for re-tuning. If sub is unable, generate trouble ticket for FSR to re-tune TV.
Case 2	
Score	46
Name	Converter hookup problems
Description	Converter will change channels but TV set does not.
Solution	Check connections with converter, TV set, and any other equipment. Make sure TV set is on channel 3.

Table 6.19: Case Retrieval Result

description of the problem, such as *problem with channel*, or just *channels* to the Problem-Resolution Module. Then the representative will ask the customer some questions. We show in Table 6.18 the questions posed by the technical representative and the answers replied by the customer. After these questions are answered, the Problem-Resolution Module retrieves a set of cases. These cases are ranked by their scores from high to low. We list in Table 6.19 the two cases with the highest scores produced by the query shown in Table 6.18.

Now in this experiment we will also apply our dynamic learning method to this case base. In order to do our experiment, we set up two copies of the case base with different sets of weights. The first copy of the case base uses the weights specified by the domain experts from the company. The second copy has the weights initialized to 0.5. If we think the weights in the first copy represent a user's preference in the company, then we will learn these weights in the second copy using our dynamic learning method.

We select 13 out 28 cases(problems) from the case base. These 13 cases are *frequently* used in the company. Accordingly we also select seven valid sets of question-answer pairs in the case base. In the first copy of the case base, the seven case retrieval results produced by these seven valid sets cover all these 13 cases. Each of them occupies one of the three highest positions in one of the seven case retrieval results. Thus we can think that these seven valid sets represent the user's preference in the company.

The case base in the company is used for the purpose of solving a customer's technical problem. All the technical representatives in the company use the same weights when

dealing with new problems. Therefore there is only one user's preference during their work. Besides this, this preference is constant within a relative long time. We also observe that the interactions among question-answer pairs are little, i.e. a case's coverage is in little common with another case's. For this situation, we claim that our dynamic learning method will quickly simulate the user's desired preference. Another fact we need to notice is that there is no solution layer in this case base. The connections are only between cases and question-answer pairs. As indicated in Chapter 4, our learning model can employ single-layer neural network to do weight adjustments if the user wants to input judgement at the problem layer.

The whole learning process is as follows. We open the two copies of case base in two separate Problem-Resolution Modules. For the first copy, we input one of the seven valid sets of question-answer pairs. The module produces a set of cases and their scores. For the second copy, we input the same valid set. Also a set of cases and their scores is produced. By comparing these two sets of cases and their scores we can find the differences between them. For each of the three highest cases produced by the first copy, we find its correspondent in the cases produced by the second copy. Then according to its score produced by the first copy, we specify its correspondent's score and trigger the learning action in the second copy. The comparison, specification, and learning process will be applied to each of the 7 valid sets in turn until all the seven case retrieval results produced by the second copy are around the ones produced by the first copy.

In the use of a CBR system a valid set of question-answer pairs is actually a query to the case base, which produces a set of cases and their scores. Thus we can call such a valid set a query.

Experiment Result

In our experiment, the whole training process takes four rounds, each of which is composed of query 1 to query 7. All the scores of the 13 cases produced by the second copy converge to their desired scores produced by the first copy.

We can define the error of a case produced by a valid set of question-answer pairs in the learning process is the absolute difference between its computed score and its desired score. In our test, the desired score of a case is produced by the first copy of the case base while the computed score is produced by the second copy.

For each query, we take a look at the case having the highest score. Thus we get a total of seven cases. We check these cases' errors by comparing their computed scores produced

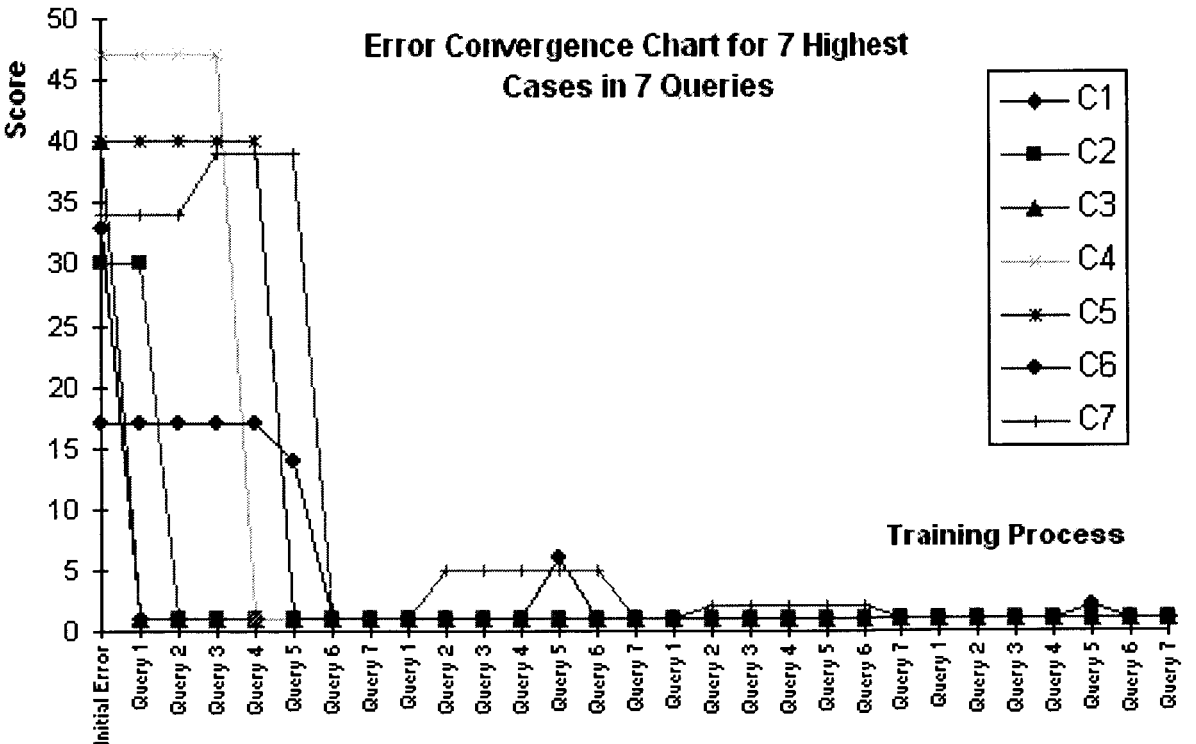

Figure 6.5: Error Convergence Chart for Seven Highest Cases in Seven Queries

by the second copy with their desired scores produced by the first copy during the training process. The error convergence chart for these seven cases is graphed in Figure 6.5.

We also take a look at the case having the second-highest score produced by each query. We still get a total of seven cases. We check these cases' errors as above during the training process. The error convergence chart is now graphed in Figure 6.6.

In the above two figures, the X-axis represents the training process shown as queries. The Y-axis represents the case score. We can find that all the errors converge to 0, which means that all the case scores converge to their desired scores. We attribute this result to the observation we state in the experiment setup(see the beginning of this section). Also the small number of training rounds confirm our claim made there.

If we think the domain experts from the company have the valid sets or queries and their corresponding scores in their minds, then we can ask them to train our second copy as an ordinary user, just as shown in the learning model in Chapter 4. Once trained, such a copy can be used by other technical representatives in the company.

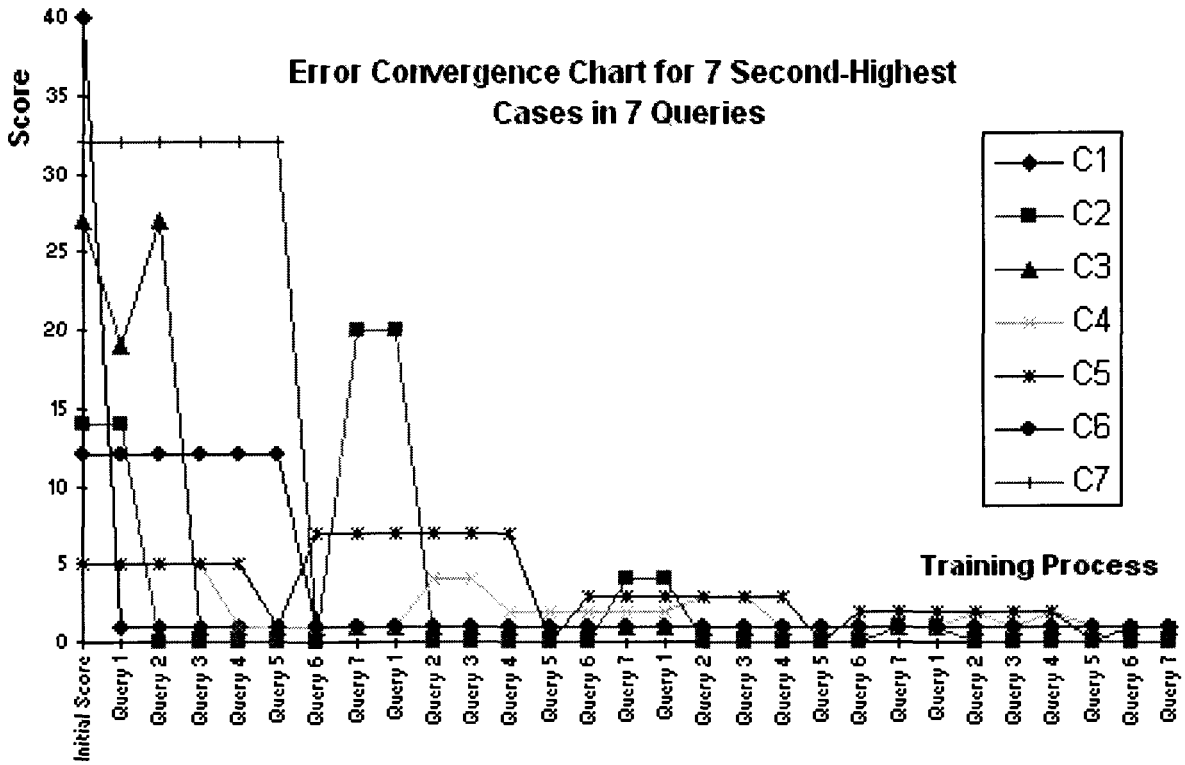

Figure 6.6: Error Convergence Chart for Seven Second-Highest Cases in Seven Queries

However we have to indicate that although our method has brought the 13 cases to their desired scores, we cannot guarantee that after the four training rounds in the learning process, these 13 cases's scores will still converge to their desired ones. Maybe the user will change her/his input distribution. Or a different user might come. All these will destroy the previous convergence state and trigger another learning process.

6.3.4 Some Analysis

Comparison with Bonzano et al.'s Method

We have discussed a learning method proposed by Bonzano et al.[1] for the weight adjustments in Chapter 2. Here, we try to compare our method with theirs. Bonzano et al. employ an introspective learning the adjust the feature weights. We argue that their method could not deal with a situation that two targets will make a problem to take two contradictory operations, 'push' or 'pull'. Our method, taking advantage of a neural network learning

method, tackles this situation effectively. The essence of our algorithm is an overall evaluation of the contribution of each particular weight to the final output error. It takes the error as an overall behavior of the system, thus it modifies the weights so that the error decreases not just for one problem, but for all the problems. For each user's preference, the modification is local to it. However, when a user's several preferences are presented to the system, the system will adjust them as a whole without any locality. Furthermore, with more and more preferences, the system behavior tends to increasingly approximate a user's desired behavior closely. For instance, from the above experiments, after several rounds, the system has already simulated a user's preference to an optimal degree.

As for the pivotal case(problem)[48](See Chapter 2 for its definition), our system can still adjust its retrieval to an acceptable state. Recall in Chapter 4, we argue that if a user responses to a problem retrieved by the system at the problem layer, our system will employ Formula 4.17 to adjust its weights, and the adjustment will be local to it. This means that even a case(problem) is pivotal, its weights can still be adjusted, and do not need the interactions from other problems. This, in fact, provides an approach to dealing with a pivotal case(problem) in the learning process.

Learning Parameters

We have used several parameters in the learning process. The *activation rate* λ is used in Formulas 4.11 and 4.12. The higher this parameter is, the faster the activation function $f(x)$ will change along with of the change of x. In our implementation, we preset this rate to 1.0. A user can change it when using the system. In addition to the activation rate, we also introduce a *bias factor.* In our implementation, when the system uses Formula 4.12 to compute to the scores of the solutions for a selected problem, the bias factor the selected problem is 8.0. For the unselected problems this factor will be $2.0/(n-1)$, where n represents the number of the problems this solution is associated with. We deduct 1 from this number in order to exclude the selected problem itself.

The *learning rate* η in adjustment Formulas 4.9 and 4.10 decides how fast the learning speed is. Based on the discussions in Chapter 4, we hope that learning speed for the selected problem could be faster than that for the unselected problems. In our implementation for Formula 4.9, we choose the learning rate 6.0 for the selected problem, and $6 * 0.66/(n-1)$ for the unselected problems, where n represents the number of problems this solution is associated with. For Formula 4.10, the learning rate is the same for all the problems. Again

a user can change this rate based on her/his option in the system.

For more discussions on the activation rate and learning rate, see [54].

When a problem or solution is judged to be correct or wrong, if a user does not specify the desired score for it, our system will add or deduct an adjustment delta from the actual score. The default value for this parameter is 5.0. A user can also specify this default value as long as it is between 1.0 to 10.0.

Convergence and Real-Time Response

Zurada in [54] discusses various factors affecting the convergence of the training of a back-propagation of neural network, including the initial weights, the activation rate, the learning rate, the momentum method, and so on. It is proposed that the effectiveness and convergence of the backpopagation learning algorithm depend heavily on the value of the learning rate, which should indeed be chosen experimentally for each domain. It is further argued that the values ranging from 10^{-3} to 10 for a learning rate have been reported throughout the technical literature as successful for many computational backpropagation experiments. In our implementation, we choose 6.0 for our learning rate. During the use of our system, on average it takes only about 5-10 seconds for the system to finish the adjustment task, and then make the scores of the problems and/or solutions updated. From this viewpoint our system is real-timed. A user's action will be captured by the system, and be reflected in its own behavior very quickly. This confirms what we have discussed in Chapter 4, and at the beginning of this section about what a learning component should be.

However, we also note that a worst situation happens when a problem has several solutions which need to adjust simultaneously and the adjustment deltas are large. The system will be divergent. For this reason, we limit the adjustment delta within ± 10 from the actual score of a problem or a solution to avoid this situation. Under such a restriction, the system never encounters the divergence in our use up to now.

6.4 Summary

We have shown the empirical test results of the static weight adjusting algorithm and the dynamic weight learning algorithm. Based on the discussions throughout the chapter, we can find the test results show that these two methods fulfill our goals set in the previous chapters, confirming the hypotheses we make about them.

Our static weight adjusting method digs out the statistical information hidden in a case base. In the experiment, although it is not accurate for every feature-value pair, it maintains the order of their relative importance in the minds of the domain experts. In contrast to this, our dynamic learning method captures the interactions between the system and its end-user, and seeks the chance to change itself. In the test, the system gradually approximates a user's behavior within an optimal number of iterations. If we enlarge this scenario, we can find that during the use of the system, a user's behavior is demonstrated through the interactions with the system. Once it changes, the changes will be captured. Therefore, we conclude that our integrated framework changes its behavior accordingly whenever its user changes her/his behavior.

Chapter 7

Conclusion and Future Work

In this chapter, we will summarize our work on the static weight adjusting and dynamic weight learning methods. Besides these we will also discuss the limitations of our two feature-weighting methods. Going further we will discuss what we want to do in order to further enhance the power of our methods with regard to the feature-weighting task in the case-base maintenance problem.

7.1 Summary

Our work aims to improve the predictiveness of the feature weights in the case retrieval process in CBR. During our use and experience of CBR to solve knowledge-intensive problems, the static and dynamic characteristics of a CBR approach inspire us to think about a case base from both static perspective as well as dynamic perspective.

Some of the inspiration is directly from our use of a CBR system to solve new problems. In the case retrieval process, we find that when a feature-value pair or question-answer pair is selected, the scores of a set of cases will be migrated from high to low or from low to high, either dramatically or slightly. If the number of migrated cases is small, then we can easily distinguish these cases from others. We always like to choose the one whose ranking score migrates from low to high dramatically as a potential candidate for further consideration. This lays down the foundation of the intuition on which our static weight adjusting method is based. Further analysis on the associations between the cases and their associated feature-value pairs in a case base reveals that there is some useful statistical information hidden there. The distribution of the feature-value pairs within a case base is not uniform, with

some feature-value pairs associated with more cases while some associated with few. We collect this information, and then accordingly assign back the weights to the corresponding associations. Our hope is that this assignment would demonstrate our observations as shown above for a case base. The empirical tests we have done show that the static weight adjusting method fulfills this goal.

The needs from practical applications of CBR also encourage us to think about its dynamic nature. In practice, we often encounter a situation that an end-user is not satisfied with the result produced by the retrieval algorithm using the feature weights, which are specified by the domain experts during the construction of a case base. In addition, different users may have different purposes when using the same case base. Obviously, a single set of weights is unable to deal with the situation. We project this situation into a larger background. In our imagination, a CBR system is a responsive system. Its behavior needs to simulate its end-user's behavior, incorporating her/his own preferences. Furthermore, a user's behavior is changing, requiring that a CBR system keep its own pace with the changes. After analyzing these needs, we are encouraged to introduce a learning component into CBR so that while CBR is still responsible for the reasoning process, the learning component now shoulders the burden of the learning process in the integrated framework. The learning component will learn its behavior from its end-user's in a dynamic contexts over time. We combine a backpropagation neural network with CBR based on the analysis of their similarities. However we also take into account the dissimilarities between them within the frame of CBR. Empirical experiments in our tests show that such an integrated framework achieves the goal we desire.

The case retrieval process is of key importance in the use of a CBR system. How to maintain a set of suitable and updated weights for feature-value pairs, in order to obtain an ideal retrieval quality for end-users, is a crucial task in CBR. Statistical information hidden in a case base itself needs to be dug out, while the dynamic contexts, participated in by not only the domain experts but also the end-users of a CBR system, also need to be further examined and analyzed. We hope that our work would be a beneficial attempt along these directions.

7.2 Limitations

7.2.1 Limitations of Static Weight Adjusting Method

The static weight adjusting method we introduce into CBR is based on the intuition that a feature-value pair associated with a small number of cases in a case base conveys more information, and thus needs to be assigned a higher weight. Although we have shown some desired result produced by the method, we cannot regard it as a general standard to judge every case base.

Sometimes domain experts do not want only one feature-value pair to dominate the similarity computation between two cases. Rather they want an overall evaluation of all the relevant feature-value pairs. Under this situation, the feature-value pair which is associated with a small number of cases would also be considered bad, just as the pair which is associated with too many cases. The weight assignment in this situation will observe a normal distribution. A feature-value pair associated with too many or too few cases will be considered to convey little information, while a pair associated with almost half of the cases in a case base will be considered to convey more information.

As an example, we show this distribution in Figure 7.1(for a case base of 100 cases). In the figure, the desired weight assignment is a normal distribution, where the X-axis represents the number of associated cases while the Y-axis represents the desired weight. For a feature-value pair, if the number of its associated cases is around 50 cases, then it should have the highest weight. On the contrary, if a feature-value pair is associated with 20 or 90 out 100 cases then it will have a very low weight, which is about 0.11 or 0.09 in the figure. Obviously, our method cannot deal with such a distribution of weight assignment.

7.2.2 Limitations of Dynamic Weight Learning Method

Follows are some limitations with our dynamic weight learning method.

1. Convergence problem. Although in our experimental tests, nearly all the cases converge to their desired scores, we actually encountered divergence several times due to the interactions among different cases. How can we know whether our method will be convergent for a particular domain beforehand? Experiences and domain knowledge might be useful when dealing with such a problem.

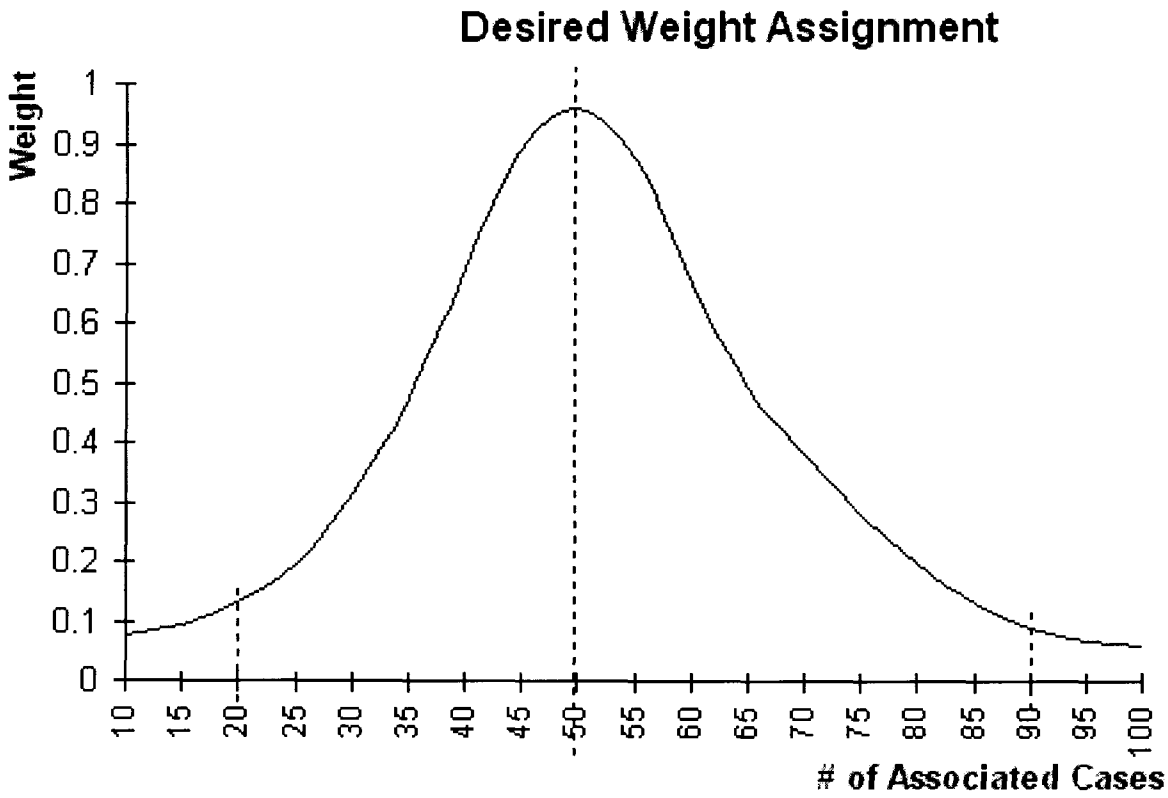

Figure 7.1: Normal Distribution of Weights

2. Different users. One of the assumptions of our learning model is that the user of our system should be one person. If a different user comes, s/he might not satisfy the previous optimal case retrieval result. How can we do to deal with such a situation? Maybe we have to discard the previous learning result, and trigger another learning process.

3. Training and testing stages. In the learning process, the input space at the training stage might not cover the input space at the testing stage. For the input which was trained before, its retrieval result will be optimal. For the input that was never trained before, we cannot guarantee that our method can still produce the optimal retrieval result. How can we do with such a situation?

4. Learning parameters. We have many learning parameters to decide in order to trigger the learning process. Such decisions have to be made based on the analysis of

individual domains. This brings inconvenience to ordinary users.

5. Scaleup problem. Assume a case base, after trained, gets to its optimal retrieval state. At this time, maybe more new cases come, how can we do with them? Because of the interactions among different cases, maybe these new cases' presence in the case base will not only need to learn themselves, but also make other cases to relearn, destroying the previous optimal state. From this viewpoint, our method is more applicable to the case bases which are relatively steady. How to deal with scaleup problem is also the direction along which we want to do further work.

7.3 Future Work

Our proposed methods and their performance provide the convincing arguments on the consideration from both the static and the dynamic perspectives about a CBR system. However, in order to improve the methods we propose, there is still some work to do.

Although we have tested our work using some artificial and real world data, more data will be needed to further tune and validate the underlying algorithms for these methods. For instance, we need to do more tests of static weight adjusting method on different sizes of case bases to decide what the adjustment parameters will be appropriate for each size. For dynamic weight learning method, we need to put it into a real problem-solving environment and let multi-users use it simultaneously. After a period of time, we will check whether the performance of the weights tend to approximate an individual user's preferences to a desired extent.

We collect the statistical information hidden in the associations between cases and feature-value pairs and define five groups for the adjusted weights of individual feature-value pairs using Formula 3.6. Definitely, this is not the only computation method with regard to this information hidden in a case base. We need more attempts on different weight computation methods and compare with each other for their performance. This not only needs theoretical arguments but also practical experiments.

As indicated [54], any function that is monotonically increasing and continuous such that its defining area is R and its value area is $(-1, +1)$ can be used as the activation function in a backpropagation neural network. We have used the one(see Chapter 3), which is currently most used, in our implementation. Although their performance satisfies our desire, we can

never stop searching for the one which is optimal in a CBR framework. Again this attempt not only involves mathematical theory, but also the empirical and practical experiments.

Our attempts can not stop. If we go further along the static nature and dynamic nature of a case base, we can find more interesting topics which deserve further research and application efforts.

In the static weight adjusting method, after we collect the distribution information hidden in a case base, if we find a case whose associated feature-value pairs are all belonging to bad group or very bad group, can we think that this case is not so useful? In our intuition, matching any of its associated feature-value pairs will not be able to make it distinguished. What can we do with such a case? If we take another look at such situation, we may find a feature whose values all fall into the bad group or very bad group. Since matching any of its values does not contribute much to any case, is this feature not a relevant or useful feature in this case base? This is a question we want to ask. On the contrary, if a feature's values are all in the very good or good groups, is there any hidden useful information behind it? From these facts, if we check each individual cases and each individual features after collecting static distribution information, we can dig out a lot of useful information. Because of the time and space limit, we are unable to realize this in our implementation.

Within the learning component we introduce to a CBR system, we maintain a system transcript which records what happened in the past in the learning process. This transcript is much like a relational table with each record consisting of the case identity of the confirmed case, the current keywords, the current feature-value or question-answer pairs matched, and so on. Obviously, there is some useful information hidden in it. For instance, we may find that after several particular feature-value pairs are matched, one particular case must be confirmed for its correctness. Can we borrow some techniques from data mining[21, 18], which combines together the AI and Database technologies in computer science, to find some useful information from this transcript?

If we think that the associations specified between the cases and the feature-value pairs by the domain experts represent their domain knowledge and experiences, after the static weight adjusting and dynamic weight learning, how do we deal with the feature-value pairs with too low weights? Although we do not think they are useful, can we unassociate them with their associated cases? If it is feasible, does this process provide us with an approach to dealing with irrelevant and noisy features?

In the light of the questions posed as above, we still have a lot to do in order to further

our work. With many recent successes of CBR in the industrial world, it can be foreseen that CBR will be gaining more and more acceptance. This, in turn, inspires us to further improve a CBR system's performance. Thus, the static as well as dynamic views we take on CBR are expected to be a promising direction which will attract more and more efforts and attentions.

Bibliography

[1] P. Cunningham A. Bonzano and B. Smyth. Using introspective learning to improve retrieval in car: A case study in air traffic control. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 291–302, Providence RI, USA, 1997.

[2] A.Aamodt. A knowledge-intensive approach to problem solving and sustained learning, ph.d. dissertation. Technical report, Norwegian Insitutue of Technology, University of Trondheim, May 1991.

[3] A. Aamodt and E. Plaza. Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, 1993.

[4] S.M.Weiss A.Ginsberg and P.Politakis. Automatic knowledge base refinement for classification systems. *Artificial Intelligence*, 35:197–226, 1988.

[5] D. W. Aha. Integrating machine learning with knowledge-based systems. In *Proceedings of the First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, pages 150–151, Dunedin, NZ, 1993. IEEE Press.

[6] D.W. Aha. A study of instance-based learning algorithms for supervised learning tasks: Mathematical, empirical, and psychological evaluations. Technical report, Department of Information and Computer Science, Irvine, CA: University of California, 1990.

[7] D.W. Aha. Case-based learning algorithms. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 147–158, Washington, D.C., 1991. Morgan Kaufmann.

[8] D.W. Aha and L. Breslow. Refining conversational case libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 267–276, Providence RI, USA, 1997.

[9] D.W. Ana and R.L. Goldstone. Concept learning and flexible weighting. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 534–539, Bloomington IN, USA, 1992. Lawrence Earlbaum.

[10] K.D. Ashley. Assessing similarities among cases. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 72–76, Pensacola Beach, FL, USA, 1989. Morgan Kaufmann.

[11] M. T. Cox. Loosing coupling of failure explanation and repair: Using learning goals to sequence learning methods. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 425–434, Providence RI, USA, 1997.

[12] P. Cunningham and B. Smyth. Cbr in scheduling, reusing solution components. In *to appear in The International Journal of Production Research*, 1997.

[13] D.W. Aha D. Wettschereck and T. Mohri. A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review*, 11:273–314, 1997.

[14] A. Kinley D.B. Leake and D. Wilson. Learning to improve case adaptation by introspective reasoning and cbr. In *Proceedings of the First International Conference on Case-Based Reasoning*, pages 229–240, Sesimbra, Portugal, 1995. Springer-Verlag.

[15] C. Drummond. Using a case base of surfaces to speed-up reinforcement learning. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 435–444, Providence RI, USA, 1997.

[16] C. Feynman. Nearest neighbor and maximum likelihood methods for social information filtering. Technical report, Media Laboratory, MIT, December 1993.

[17] S. Fox and D. B. Leake. Learning to refine indexing by introspective reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995.

[18] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *1995 International Workshop on Knowledge Discovery and Deductive and Object-Oriented Databases(KDOOD'95)*, pages 39–46, Singapore, December 1995.

[19] F. Weberskirch H. M. and T. Roth-Berghofer. On the relation between the context of a feature and the domain theory in case-based planning. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 337–348, Washington, D.C., 1991. Morgan Kaufmann.

[20] K. Z. Haigh and M. Veloso. Route planning by analogy. In *Proceedings of the International Conference on Case-Based Reasoning, ICCBR-95*, Sesimbra, Portugal, October 1995.

[21] J. Han. Conference tutorial notes: Data mining techniques. In *1996 ACM/SIGMOD International Conference on Management of Data (SIGMOD'96)*, Montreal, Canada, June 1996.

[22] N. Howe and C. Cardie. Examining locally varying weights for nearest neighbor algorithms. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 455–466, Providence RI, USA, 1997.

[23] K. Kira and L.A. Rendell. A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 249–256, Aberdeen, Scotland, 1992. Morgan Kaufmann.

[24] J. L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann Publisers, Inc., 1993.

[25] K.Racine. Design and evaluation of a self clearning agent for maintaining very large case bases(vlcb). m.sc. dissertation. Technical report, School of Computing Science, Simon Fraser University, 1997.

[26] D. B. Leake. Cbr in context: The present and future. In David B. Leake, editor, *Case-Based Reasoning, Experiences, Lessons & Future Directions*, pages 1–30. AAAI Press / The MIT Press, Menlo Park CA, USA, 1996.

[27] D. Lowe. Similarity metric learning for a variable-kernal classifier. *Neural Computation*, 7:72–85, 1995.

[28] G. F. Luger and W. A. Stubbfield. *Artificial Intelligence, Structures and Strategies for Complex Problem Solving.* The Benjamin/Cummings Publishing Company, Inc., second edition, 1993.

[29] P. Maes. Agents that reduce work and information overload. *Communications of ACM*, 37(7):31–40, July 1994.

[30] T. Mitchell. Version spaces: an approach to concept learning. doctoral dissertation.

[31] K. Miyashita and K. Sycara. Improving system performance in case-based iterative optimization through knowledge filtering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 371–376, Montreal, Canada, August 1995.

[32] M.M.Gupta and H.Ding. Foundations of fuzzy neural computation. In Fred Aminzadeh and Mohaamad Jamshidi, editors, *Soft Computing, Fuzzy Logic, Neural Networks, and Distributed Artificial Intelligence*, pages 165–200. PTR Prentice Hall, Englewood Cliffs, New Jersey, USA, 1994.

[33] R. M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag, 1996.

[34] P.E.Clark. Exemplar-bases reasoning in geological prospect appraisal. Technical report, Turing Institute, University of Strathclyde, Glasgow, Scotland, 1989.

[35] P.E.Clark and S.Matwin. Using qualitative models ot guide inductive learning. In *Proceedings of the Elventh International Conference on Machine Learning*, pages 1–10, Amherst, MA, 1993. Morgan Kaufmann.

[36] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[37] R. Edwards R. Ochlmann and D. Sleeman. Changing the viewpoint: Re-indexing by introspective question. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, pages 381–386. Lawrence-Erlbaum and Associates, 1995.

[38] K. Racine and Q. Yang. Maintaining unstructured case bases. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, pages 553–564, Providence RI, USA, 1997.

[39] R.Bareiss. Protos: a unified approach to concept representation, classfication and learning. ph.d. dissertation. Technical report, Department of Computer Science, University of Texas at Austin, 1988.

[40] F. Ricci and P. Avesani. Learning a local similarity metric for case-based reasoning. In *Proceedings of the International Conference on Case-Based Reasoning, ICCBR-95*, Sesimbra, Portugal, October 1995.

[41] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, Inc., second edition, 1991.

[42] C. K. Riesbeck. What next? the future of case-based reasoning in postmodern ai. In David B. Leake, editor, *Case-Based Reasoning, Experiences, Lessons & Future Directions*, pages 371–388. AAAI Press / The MIT Press, Menlo Park CA, USA, 1996.

[43] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach.* Prentice Hall, Inc., 1995.

[44] S.L. Salzbery. A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276, 1991.

[45] B. Sheth and P.Maes. Evolving agents for personalized information filtering. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications.* IEEE Computer Society Press, 1993.

[46] H. A. Simon. Why should machines learn? In J.G.Carbonell R.S.Michalski and T.M.Mitchell, editors, *Machine Learning: An artificial intelligence approach.* Morgan Kaufmann, San Mateo, California, 1983.

[47] M. L. Smith. Cooperating artificial neural and knowlege-based systems in a truck fllet brake-balance application. In *Proceedings of the Second Innovative Applications of Artificial Intelligence conference*, pages 263–280, Menlo Park, California, 1991. AAAI Press.

[48] B. Smyth and M. T. Keane. Remembering to forget. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 377–382, Montreal, Canada, August 1995.

[49] I. Watson. Case-based reasoning tools: An overview. In *Proceedings of the Second UK Workshop on Case Based Reasoning*, pages 71–88, 1996.

[50] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems.* Morgan Kaufmann Publishers, Inc., 1997.

[51] D. Wettschereck and D.V. Aha. Weighting features. In *Proceedings of the First International Conference on Case-Based Reasoning, ICCBR-95*, pages 347–358, Lisbon, Portugal, 1995. Springer-Verlag.

[52] D. Wettschereck and T.G. Dietterich. An experimental comparison of the nearest neighbor and nearest hyperrectangle algorithms. *Machine Learning*, 19, 1995.

[53] M. Matral Y. Lashkari and P. Maes. Collaborative interface agents. In *Proceedings of the National Conference on Artificial Intelligence*, 1994.

[54] J. M. Zurada. *Introduction to Artificial Neural Systems.* West Publishing Company, 1992.