Journal of Intelligent Information Systems, 16, 95–116, 2001

Feature Weight Maintenance in Case Bases Using Introspective Learning

ZHONG ZHANG zzhang@cs.sfu.ca
QIANG YANG qyang@cs.sfu.ca
School of Computing Science, Simon Fraser University, Burnaby, BC Canada V5A 1S6

Abstract. A key issue in case-based reasoning is how to maintain the domain knowledge in the face of a changing environment. During the case retrieval process in case-based reasoning, feature-value pairs are used to compute the ranking scores of the cases in a case base, and different feature-value pairs may have different importance measures, represented as weight values, in this computation. How to maintain a set of appropriate feature weights so that they can be used to solve future problems effectively and efficiently will be a key factor in determining the success of case-based reasoning applications.

Our focus in this paper is on the dynamic maintenance of feature weights in a case base. We address a particular problem related to the feature-weight maintenance issue. In current practice, the feature weights are assigned and revised manually, not only making them highly informal and inaccurate, but also involving intensive labor. We would like to introduce a semi-automatic introspective learning method to partially address this issue. Our approach is to construct a network architecture on the case base that supports introspective learning. Weight learning and weight-evolution are accomplished in the background through the integration of a learning network into case-based reasoning, in which, while the reasoning part is still case based, the learning part is shouldered by a layered network. The computation in the network follows well-known neural network algorithms with well known properties. We demonstrate the effectiveness of our approach through experiments.

Keywords: case base maintenance and indexing, knowledge base indexing, dynamic index update through machine learning

1. Introduction

Case-based reasoning (CBR) has enjoyed tremendous success as a technique for solving problems related to knowledge reuse. Many examples can be found in the CBR literature (Czerwinski et al., 1993; Perini and Ricci, 1995; Leake, 1996; Kolodner, 1993; Watson, 1997). One of the key factors in ensuring this success is CBR's ability to allow users to easily define their experiences incrementally and to utilize their defined case knowledge when a relatively small core of cases is available in a case base.

However, defining the case knowledge is just the first step in the long life cycle of a knowledge-based application. In today's industrial environment, new cases are entered at a very fast rate, making it necessary to reorganize a case base from time to time. The relative importance of the cases are also changing, partly due to the uneven and changing distribution of the inherent problem space, and also partly due to the changing interest of users. How to evolve a case base continuously is an important issue in the knowledge base industry.

In this paper we present our research result in a focused area of case base maintenance, dealing primarily with the issue of how to maintain the importance measures of different

features. What we wish to have is a feature weight maintenance system that would learn its end users' desired weight values in a relatively short time. We also wish to have a CBR system evolve with its target user group so that when it ranks the different cases in response to a user input, the user's most current preferences and needs are strongly reflected.

Introspective learning has been proposed in the past for learning and adjusting the feature weighting (Fox and Leake, 1995). Introspective learning has a process to detect deviations that show when the learning is needed as well as what the learning needs (Leake et al., 1995; Ram and Cox, 1993). A main theme of this learning type adopts *qualitative* introspective learning, whereby the feature weights are adjusted based on a rough estimate of the *direction* for a change: if the weights are too high, then adjust them so that they become lower, and vice versa. But how much has to be changed quantitatively is not sufficiently determined. In this work, we extend qualitative introspective learning to quantitative introspective learning within CBR. Using quantitative learning, we can adjust the weights not only in the right *direction*, but also in the right *amount*. Such an extension provides a sound and promising continual introspective learning method for feature weighting in CBR. Our approach integrates a learning network with a CBR system. The architecture in our approach resembles that of a neural network, a feature which enables us to utilize the well-known back-propagation algorithms for accomplishing our maintenance tasks.

In the next section we present a brief introduction to case-based reasoning (CBR), focusing on the issues regarding feature weighting. Section 3 presents a new architecture of a case base and its related maintenance algorithms. In Section 4 we survey the literature for related work on case base maintenance, especially on feature weighting, and compare our approach with them. In Section 5 we discuss the experimental results for evaluating the performance of our system. In Section 6 we present an extension from the two-level architecture of the learning system to a three-level one. We conclude our discussion in Section 7, where we will also explore our future work.

2. Background

2.1. Case retrieval in case-based reasoning

The first step in developing a case-based reasoner is to build a case base. In general, there are three major components in a case (Kolodner, 1993).

1. *Problem description*: The status of the situation where this case occurred, and if appropriate, what problem was being solved at that time;
2. *Solution*: The explicitly stated or implicitly derived solution to the problem described in *problem description*;
3. *Outcome*: The resulting status 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 Kolodner (1993), in situations with many unknown or undetermined factors, severe inaccuracies might arise when the cases

Table 1. First example of case representation.

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

Table 2. Second example of case representation.

Question 1	Is the subscriber in pay status? Answer: Y
Question 2	What type of problem is being experienced? Answer: Picture
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.

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 less and less accurate and efficient. A third part, outcome, is often used to allow a reasoner to record and analyze the feedbacks from the outside environment. It records 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 the reason for such a success or failure. The execution result is often stored in a log or transcript file.

Once the information contained in a case for a particular application has been decided, it is relatively easy to decide the general structure for the representation of this information. Tables 1 and 2 areexamples of two types of the representation of a case. The case in Table 1 is fairly refined, down to the detailed features and their values, while the case in Table 2 has only two major parts: problem description and problem solution.

After a case base has been constructed, the next essential task is feature indexing. Feature indexing is to determine which features of a case will be used to facilitate its retrieval and what the weights for these features will be in the retrieval process. In practical implementations a case is associated with a set of feature-value pairs.[1] These pairs are combinations of important descriptors of a case, which distinguish it from other cases. Using these associations, a case base could be viewed as a 2-layer architecture, as shown graphically in figure 1. In the figure, we assume that a user is at the bottom layer, providing inputs to the feature-value pairs. Using the weights assigned to the connections between the feature-value pair layer and the case layer, a CBR system determines a similarity ranking for the most relevant cases, which are then returned to the user for considerations. The feature-value pairs are often presented to users as question-answer pairs in many applications.

Up to now, CBR has found a wide range of applications in the real world. In Kolodner (1993) and Watson (1997), several case-based reasoners are described in detail.

CASEY (Koton, 1988) 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

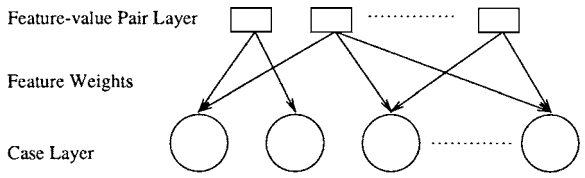

Figure 1. Two-layer architecture of a case base.

causal explanation of the disorders the patient might have. Another example is CLAVIER (Kolodner, 1993; Watson, 1997), one of the first commercial applications of CBR technology. This system configures the layout of composite airplane parts for curing in an autoclave.

One of the practical applications which are related to our work is the PROTOS system introduced in Bareiss (1988) and Kolodner (1993). It implements both case-based classification and case-based knowledge acquisition. When PROTOS misclassifies a situation or an object based on the given description, its user can intervene and notify it of its mistake and the knowledge it needs to classify the input correctly. Its knowledge acquisition is driven by failures in its classification process. When it is unable to correctly identify the category of an input, it engages in a conversation with an expert, which results in the addition of new knowledge and revision of its memory connection. It is reported in Kolodner (1993) that PROTOS has been applied to the recognition of a user's emotional state, which will changes from time to time. PROTOS has the same feature we have in that both of the models rely on the dynamic interactions from the end users, though we have a different learning purpose and employ a different learning mechanism.

For detailed background knowledge of CBR, such as CBR cycles, feature indexing and so on, see (Kolodner, 1993; Leake, 1996; Watson, 1997; Aamodt and Plaza, 1993).

2.2. *Introspective learning*

Leake and Ram (1993) summarize in a symposium report the goal-driven learning process from various aspects. They indicate that one of the three key properties of a goal-driven learner is its *introspectiveness*—the ability to notice the gaps in its knowledge and to reason about the information needed to fill in those gaps. They also pinpoint that introspective learning acquires problem-solving knowledge by monitoring its run-time performance, seeking chances in this process to learn by itself.

Fox and Leake (1995) describes experiences with introspective learning in CBR. The ROBBIE system described is an application of an introspective model to the task of refining indexes 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 with a declarative model which is used to describe the system's ideal reasoning process. Once a failure is detected, the model is used to create an explanation of the failure in terms of other failed assertions and to suggest a repair. The authors claim that even under knowledge-poor initial conditions, the introspective learning

of new feature indexes improves the success rate of the system. But they still indicate that there exists a problem with the ordering of the presentation of training cases to the system due to the inherent shortcoming of their learning mechanism.

Bonzano et al. (1997) propose introspective learning for feature weighting in CBR, demonstrating their system which combines introspective learning with CBR. They first pose the problem with their experience in constructing a CBR system for Air Traffic Control. The problem encountered 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** techniques 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 as shown in the following two formulae.

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

$$\text{decrease: } W_i(t+1) = W_i(t) - \Delta i \frac{F_c}{K_c} \tag{2}$$

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. We can observe that the timing of triggering the adjustment process is very important; when to trigger the adjustment of the weights using the above two formulae is a crucial issue yet to be further addressed in the work. This limitation makes it necessary to involve a human user in the learning process. In contrast, instead of relying on a domain-independent decaying factor, what we propose in this paper is a continual learning process in the *lifetime* of a case-based reasoner. This extension releases the human manager of the decision to explicitly trigger a learning process.

The second limitation of the work by Bonzano et al. (1997) is that it is qualitative in nature. While the direction of change in feature weights is indicated in the above two formulae, the amount of change is only influenced by the frequency of successes and failures and the decaying factor. A quantitative change would be needed to reflect the amount of adjustment in proportion to the error.

The third limitation, reported by the authors, is that the learning method does not work well for pivotal cases, as the redundancy in a case base is essential in such a learning process. A pivotal case is the one that provides coverage not provided by the other cases in a case base (Smyth and Keane, 1995). In contrast, the quantitative introspective learning paradigm that we will present in this paper will allow not only pairs of cases to be compared, but also any number of cases to participate in the learning process. This is achieved through a process in which a user can provide feedback at any time to all top-ranking cases, not just to a few selected. In Section 5, we will provide experimental comparisons between the quantitative and qualitative methods.

3. Designing a feature weight maintenance system

3.1. Problem statement

In a nutshell, the problem we attack is how to maintain dynamically feature weights in a case base in a changing and multi-user environment. Furthermore, the environment is complex in the sense that the same solution may serve to solve different problems under different contexts, and the same problem may be associated with different, alternative solutions. We wish to accomplish our goal by using semi-automatic learning methods, where we update our learned knowledge after user feedback.

Our assumptions for the research are as follows. We assume that our desired case base maintenance system is given a set of features where each feature has a set of potential values. Some subset of the features and values may be relevant to a particular case at hand at any given time, but there is no prior knowledge on which ones are actually useful to the reasoner currently. Users can provide feedbacks on the outcome of the solutions retrieved by our system through an interactive process. Our task is to update the feature weights as a user uses the system to solve problems.

The above task is directly motivated by our fielded application with a Cable-TV troubleshooting application, in which we have overseen the entire process of case base creation, the application of the CBR system for real-time problem diagnosis, and the critical problem of case base maintenance. In this domain, the creators of the case base are chosen as customer service representatives from the Cable-TV company. To assign feature weights to the case base, the creators have to manually change the weights through a case base editor. The maintenance process is so lengthy and tedious that it can potentially prohibit the end user from adopting the technology quickly and entirely. To make the problem more complex, the weights assigned to the initial case base are changing with time. For example, with the improvement in technology, the feature *VCR-recording problem* may become less important. Correspondingly its weight has to be decreased. Similarly, a feature's weight may be different dependent on different geographical regions; for example, a remote area may encounter one particular type of problem more often than an urban area.

The feature weights in a CBR system also encode its users' *preferences* or *interests*. The result of a case base retrieval process is a list of high-ranking cases. These cases represent the system's understanding of how a user prefers to solve a particular problem at hand at present time. The preference-encoding role of a case base can be best seen from a movie rental application, where the types of movies (*action*, *romance*, and so on) are reflected in the weight assignments (We will discuss more on this domain in the following.) Considering a CBR system in this way, it is natural to expect the weights to change with time and the weight maintenance system at large to cater to a user's preferences.

A case base can be conceptualized as a 2-layer architecture, where the feature-value pairs form one layer while the cases form the other, as shown in figure 1. The feature-value pair layer is connected to the case layer through a set of weights. The weight-maintenance process in our architecture is similar to that of a gradient-descent weight-learning neural network (Hinton and Anderson, 1981; Rumelhart and McClelland, 1986). An important difference between our maintenance process and a neural network is that our learning

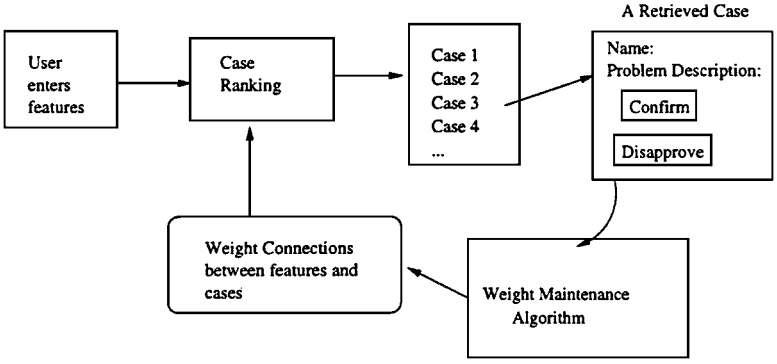

Figure 2. User's interaction model.

process is interactive rather than batch and fully automatic. In our model, there is no training data explicitly defined; the system is continuously being trained by its user throughout its lifetime. The user's interaction model is shown in figure 2.

3.2. Feature weight maintenance in the layered architecture

In this section, we will describe our learning network from a mathematical perspective. Our approach resembles a back-propagation neural network. The details on the mathematical foundations and applications of back-propagation neural networks can be found in Hinton and Anderson (1981) and Rumelhart and McClelland (1986).

Suppose that there are N features. For each feature F_i, there are m_i values, where $i = 1, 2, \ldots, N$. The case base contains J cases. The weights $W_{j,i}$ are attached to the connection between a case C_j and a feature-value pair FV_i if there is an association between them.

Given the feature-value pairs selected by a user, the corresponding nodes at the feature-value pair layer are turned on (set to one). A score is computed based on those selected feature-value pairs. For each case C_j, its score is computed using the following formula:

$$S_{C_j} = \frac{2}{1 + e^{-\lambda * \sum_{i=1}^{I} (W_{j,i} * X_i)}} - 1 \tag{3}$$

where $j = 1, 2, 3, \ldots, J$, S_{C_j} is the score of the case C_j, and X_i is 1 if there is a connection between case C_j and feature-value pair FV_i **and** FV_i is selected. Otherwise X_i is 0. This formula for computing case score is inspired by the similar formulas used in backpropagation techniques in neural network research. As in neural network research, this use of Sigmod function for scoring ensures that the influence of weight changes only affect the final score of a case in a smooth manner.

Cases will be presented to the user for his judgment after their retrieval. If the user thinks that a solution is the right one and has an appropriate score, he can confirm this by claiming

success. Otherwise, a failure can be registered by the system. In both situations, the user can have the option to either specify what the desired score of the solution is, or specify a desired rank for the case. This information is captured by the learning network, and will be used in the computation of the errors. In the situation that the user does not specify the desired score, he can also make confirmation or disapproval on a solution. In that case, a default adjustment value will be added to or deducted from the computed solution score to get the desired score.

The computation of a learning delta value is done after the scores for the cases are computered. We first compute the delta values for the solutions associated with the currently selected and confirmed case C_j. The following formula is identical to that in a neural network,

$$\delta_{C_j} = \frac{1}{2} * \left(D_{C_j} - S_{C_j}\right) * \left(1 - S_{C_j}^2\right) \tag{4}$$

where D_{C_j} is the desired score for C_j, and S_{C_j} is the computed score.

Once the user's feedback is received, weight learning is done in the general gradient-descent style. After computing the learning delta values for weight adjustments, we can adjust the weights as follows:

$$W_{j,i}^{new} = W_{j,i}^{old} + \eta * \delta_{C_j} * X_i \tag{5}$$

where $W_{j,i}{}^{new}$ is the new weight to be computed, and $W_{j,i}{}^{old}$ is the old weight attached to the connection between case C_j and feature-value pair FV_i. X_i is 1 if there is a connection between case C_j and feature-value pair FV_i and FV_i is selected by the user. Otherwise X_i is zero.

The learning rate η is the same for all the weight adjustments. This is in accordance with the computation of the case scores. The computation formulae for adjusting weights between the cases and the feature-value pairs are exactly the same as those employed in a back-propagation neural network.

In summary, the algorithm for learning feature weights can be sketched in figure 3.

Although the introspective learning process in our integrated model is similar to that of a back-propagation neural network, an important difference between them is that our learning process is interactive rather than batching and fully automatic.

3.3. System design and development

We have implemented the feature weight maintenance algorithm in the framework of the CaseAdvisor system (Racine and Yang, 1997). CaseAdvisor is a CBR system implemented using both C++ and Java by the Case-Based Reasoning Group at Simon Fraser University. This system is domain-independent and has been applied to many practical application domains, including an application in the Cable-TV domain.

Figure 4 shows the working process of the CaseAdvisor system. The system is divided into two separate modules, with the first one called Case Authoring Module and the second

```
Algorithm LearnFeatureWeights

Input: A case base D.
Output: A case base D with feature weights learned, a training query set Q.
Method:

Repeat
  For a query q from user,
        search for cases C' which have
        the highest ranking scores;
        ask the user to provide feedback on
        the score of each case in C';
        Adjust the relevant weights according
        to the feedbacks;
  End For
  Q := Q ⋃ {q};
End Repeat
```

Figure 3. Overall introspective learning algorithm.

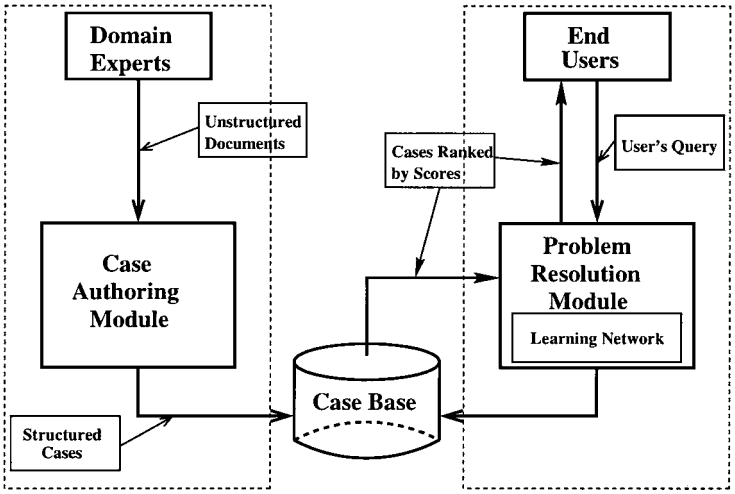

Figure 4. The CaseAdvisor system.

one Problem Resolution Module. The system has been used in realistic situations in for a cable-TV company's help desk applications.

4. Discussion

A key feature of our system is its ability to cater to a user's changing preferences. Therefore it is important for the system to selectively "forget" previous preferences and adopt new

ones. We note that traditional neural network systems suffer from the "over training" and "limited capacity" problems, whereby the new knowledge learned by a system can make the old knowledge be forgotten. This "forgetting" feature in fact is desirable for our purpose. With limited capacity, the system maintains a set of feature weights that represents the user's *most current* preferences. When new preferences arise, the system will gradually adapt itself to the new ones. In addition, we also observe that the poor-capacity problem of a neural network is partially addressed by a CBR framework because in a CBR framework the retrieval is based on similarity. Therefore the coverage of a case can be much larger than the case itself. This means that a relatively small case base can cover a potentially very large problem area.

In addition to the work of Bonzano et al. (1997), Yao and He (1994) discuss the possibility of introducing a back-propagation neural network into the retrieval process in CBR. Their network architecture include three layers involving a hidden layer, but the details of how to build such a neural network are missing. Furthermore, the lack of experimental results makes it hard for us to make convergence comparison with their work.

Recently, Conversational CBR (CCBR) has attracted substantial research (Aha and Breslow, 1997). CCBR, essentially interactive CBR, involves the refinement of diagnoses through interaction or conversation with the user, asking questions which are considered to have high information gain. These questions are based upon the unanswered attributes in the problem case which are relevant to the retrieved cases, and are ranked according to some heuristic such as the number of cases in which the attribute occurs. Popular in help-desk applications, commercial tools such as Inference Corporation's CBR Express and k-Commerce exemplify CCBR (Aha and Breslow, 1997).

5. Empirical tests

In this section, we wish to demonstrate that our proposed system conforms to our expectations. In particular, we wish to confirm the following that the feature weight maintenance system can learn the desired weights quickly after sufficient interaction between the user and the system. We expect that for applications where the frequently occurring problems are concentrated and thus the case base is small, the system can converge quickly to the desired feature weight sets.

We will test our algorithm on different application domains. The first is a Cable-TV domain. We also test the system on a case base from the Repository of Machine Learning Databases and Domain Theories at University of California at Irvine (Keogh et al., 1998).

In all experiments, we initialize all weights to be 0.5. We then simulate a process in which a user interacts with the system by repeatedly posing queries and giving feedbacks to system-generated results. In the practial application of our system, a user will continuously input queries and provide feedbacks for ranked cases. It is hoped that the weights converges to their desired scores quickly after the same query is seen over time. In the experiment, we simulate this process by repeating the same set of queries a number of times, and measure the errors as the difference between desired and computed scores. These error rates are an indication of how fast the system stablizes to the final set of scores.

5.1. Experiment with a cable-TV troubleshooting case base

5.1.1. Experiment setup. The case base which is being used in a local Cable-TV company is created using the Case Authoring Module in our CaseAdvisor system (discussed in Sections 3 and 3.2). It 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 are 30 question-answer pairs.

As an example for a troubleshooting session using the case base, 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 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 3 the questions posed by the technical representative and the answers returned 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 4 two cases with the highest scores produced under the query shown in Table 3.

In this experiment we apply our learning algorithm. We select 13 cases from the case base. These 13 cases are *frequently* used in the company. Accordingly we also create seven queries

Table 3. A user's query.

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 4. Case retrieval result.

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

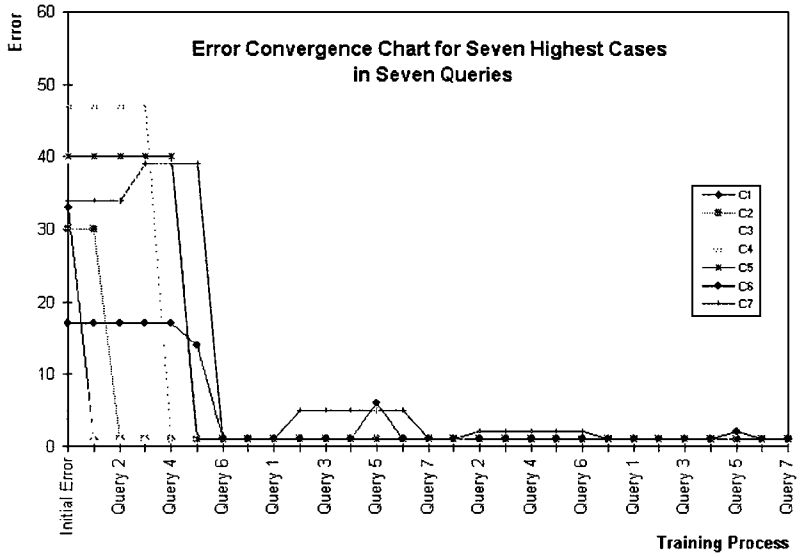

Figure 5. Error convergence chart for seven highest cases in seven queries.

of question-answer pairs. These cases are associated with the desired scores generated by the Cable-TV company experts, and are also associated with queries that used to retrieve them.

5.1.2. Experiment result. We define the error of a case produced by a query in the learning process as the absolute difference between its computed score and its desired score. The error convergence chart for these seven cases is graphed in figure 5. In the figure, the X-axis is the querying process represented as queries. The Y-axis represents the case score. We can find from the figure that all the errors converge to close to zero eventually.

5.2. An experiment with a case base from UCI

5.2.1. Experiment setup. In this section, our test again is based on the Dermatology Database at University of California at Irvine. The tests are conducted on a platform of SUN SparcStation 4 (SunOS 5.6) with 32 MB memory.

The Dermatology Database contains 366 instances and 34 attributes. In our experiment, we first convert these databases into the case bases that our algorithm can handle by converting all rows into cases and all columns into features. The values for a feature are contained under each column. In these tests, the score of a case or a solution is scaled to between 0.0 and 1.0.

Figure 6 shows the error convergence of the system as a function of queries. Queries are represented by the X-axis of the figure, and the error is shown on the Y-axis. It is easy to see that the error convergence shares the same trend as demonstrated in the previous two

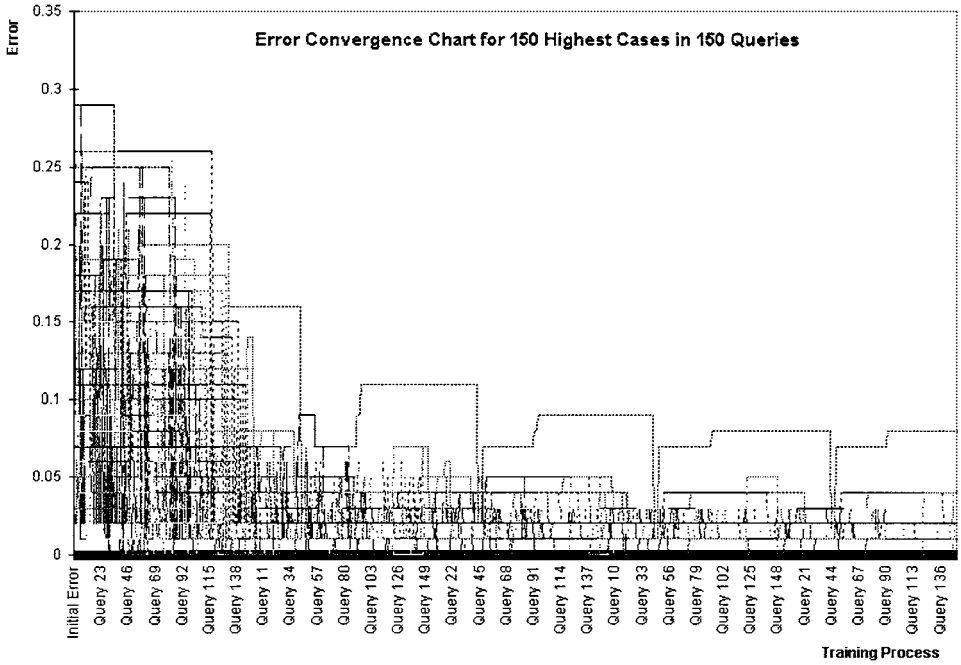

Figure 6. Error convergence chart for 150 highest cases.

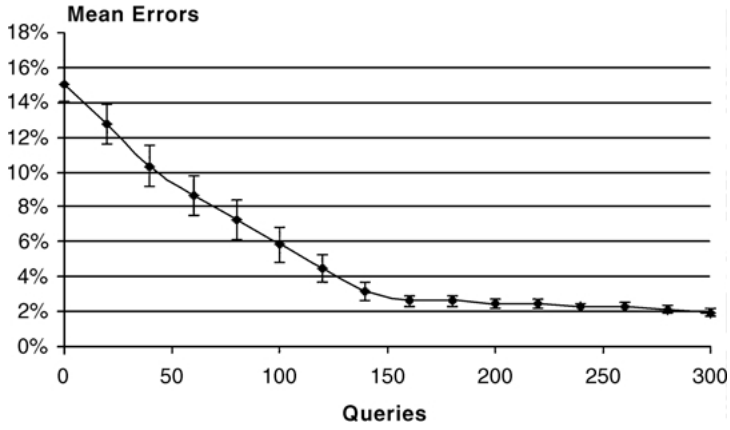

Figure 7. Plot of the mean error convergence and 95% confidence interval along the CBR process.

experiments. However, because of the interactions between different cases, not all the cases converge to their desired scores; in this experiment, seven out of 150 cases oscillate around their desired scores. The convergence of the system, as well as the reduction in interactions, can be seen from the mean-error chart in figure 7. In this figure, the 95% confidence interval

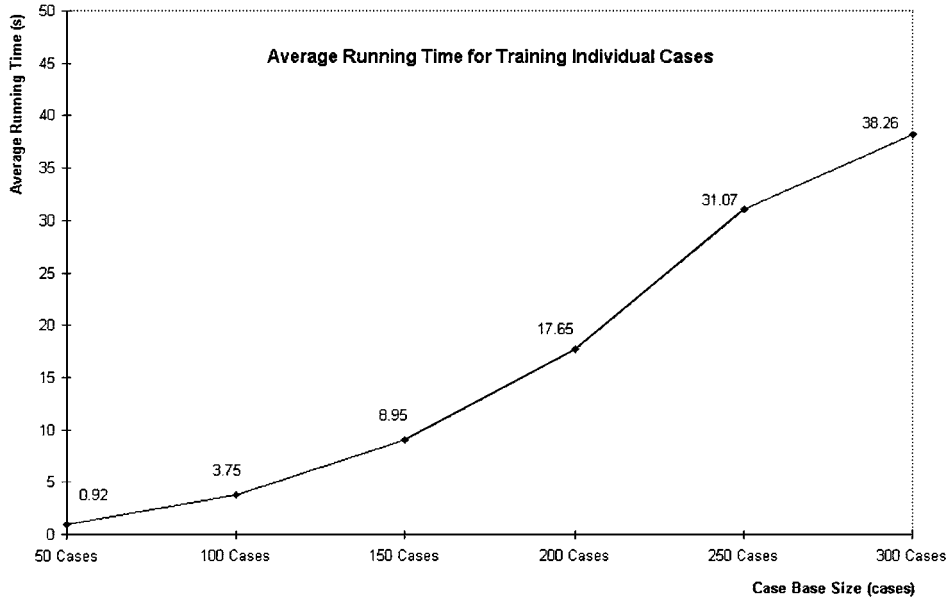

Figure 8. Average running time for learning spent on individual cases in a two-layer architecture.

is also shown on each datum point, where the size of the interval indicates the fluctuation around the mean values.

We also measure the average CPU time required for the adjustments for individual cases in each of these six case bases. The result is shown in figure 8. In the figure, the X-axis represents the six case bases with different sizes measured in cases, while the Y-axis represents the average running time for each case in CPU seconds. We can see that the increase of the running time is in proportion to the square of the number of cases in a case base. Considering the fact that in the practical applications the size of a case base is seldom very big, given that case bases represent only typical knowledge, we think that our algorithm is sufficiently fast enough to be used in practice.

5.3. A comparison with Bonzano et al.'s approach

As we discussed before, a closely-related work on maintaining feature weights is proposed by Bonzano et al. (1997), which contained sufficient details for us to make a comparison. We implemented their algorithm and compared the convergence result between the algorithms using the same Dermatology Database in UCI repository. Again the comparison is conducted on a platform of SUN SparcStation 4 (SunOS 5.6) with 32 MB memory. We choose the first 100 of the 366 cases for this test.

Figure 9 shows the comparison on the errors between these two algorithms. In the figure, the X-axis represents all test cases, while the Y-axis represents the errors of these cases at the end of training. From the figure, we can easily see that among the 50 test cases, our model produces smaller errors on 43 cases.

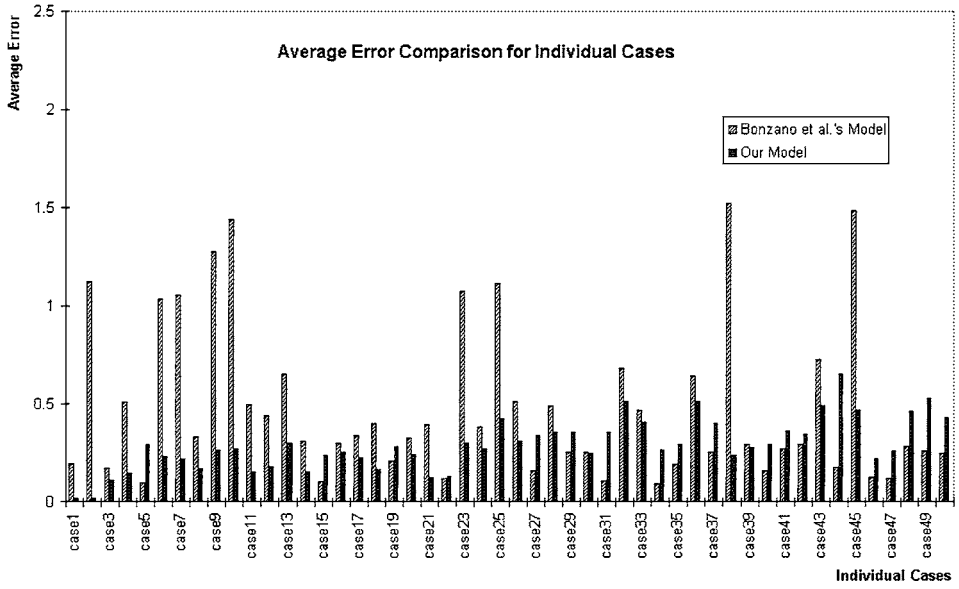

Figure 9. Comparisons between Bonzano et al's model and our model (a).

We now analyze the learning and adjustment formulae 1 and 2 of Bonzano et al.'s model. These formulae give an estimation of what should be done when a retrieval success or failure is encountered. However, such an estimation is not precise enough. For instance, if an undesired case has a lower score than expected, this means that the weights for its matching feature-value pairs have to be increased in order to compensate for the gap. Another example is that if the desired case has a higher score than expected, the case is over ranked and we have to reduce the weights associated with its feature-value pairs in order for it to be properly ranked. In the two formulae, there is no quantitative estimate associated with these information. In contrast, our adjustment strategy not only decides when to do the adjustments, but also takes into account at a more detailed level the quantitative gap between the current score and the target score, thus resulting in better learning quality. From this viewpoint, our algorithm is more quantitative while theirs is more qualitative.

A drawback of our model is that it might take longer to converge. On average, our model takes about four CPU seconds while the Bonzano et al.'s model uses approximately 1.8 CPU seconds to complete an individual learning task.

6. Extension to multi-level networks

While we have presented our architecture in terms of a two-level network, our architecture can be extended to three or multi-level networks in a straightforward manner. Consider each case as presented as a triple $\langle F, P, S \rangle$, where F corresponds to the feature values, P are the problem descriptions and S are the solutions. We can split this representation into three levels: a feature level corresponding to feature values F, a problem description

level corresponding to P and a solution level corresponding to S. With this model, a user enters feature-values at the first level. Then, the system ranks problem descriptions for the user at the middle layer. The user selects one intended problem description. Finally the system ranks the solutions corresponding to the selected problem description. A new system architecture is shown in figure 10, and a new user interaction model is shown in figure 11.

An important motivation for this separation in a case is to reduce the redundancy in a case base. Given N cases and M solutions, a case base of size $N \times M$ is now reduced to the one of size $N + M$, which eases the scale-up problem and helps make the case base maintenance task easier. A solution shared by several cases will only be revised once if a need arises.

In order to make this change possible, we introduce a second set of weights which will be attached to the connections between problems and their possible solutions. This second set of weights represents how important a case's solution is in this case, for this particular problem.

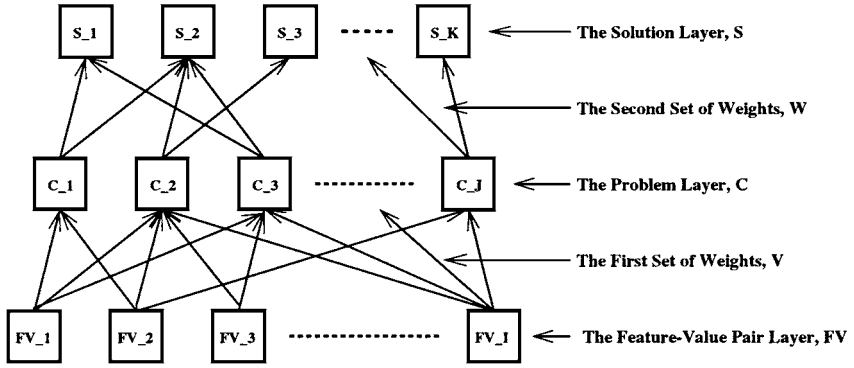

Figure 10. New architecture of a case base.

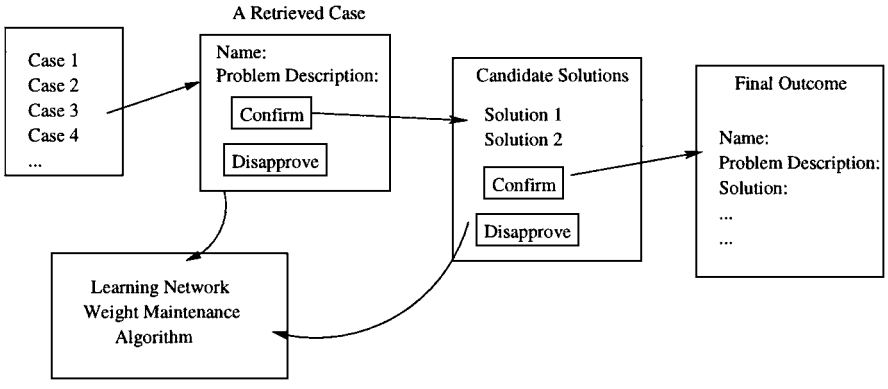

Figure 11. User's interaction model.

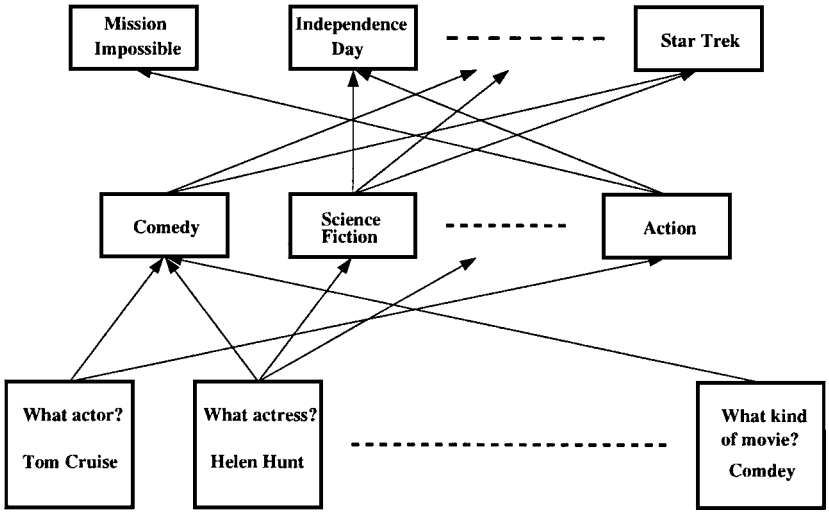

Figure 12. A movie rental example using new architecture.

In order to show the 3-layer architecture for a case base, we use a movie rental domain as an example. The movie domain is created using a CBR system and simulates an intelligent agent in a movie rental store, which helps its customers choose movies that match their tastes. traditional case base architecture of the domain has two layers. The feature-value layer is composed of a set of questions and their associated answers. The questions might be *What is your favorite actor?*, *What is your favorite actress?*, *What kind of movie do you like?*, etc. Originally the top layer is composed of a set of movie types, such as *action movie, science fiction movie*, *comedy movie*, and so on. In each movie type, there may have several specific movies for further consideration. For example, in science fiction movie, there are movies such as *Independence Day*, *Star Trek*, etc. In order to get a desired movie, a user first has to answer some questions based on his preference. The system will, according to these answers, retrieve a list of potential movie types for the user to consider. In our example domain, we collected 25 question-answer pairs and ten movie types.

Now we apply the new architecture to this domain, which is shown in figure 12. The domain right now has three layers. The first layer that is closest to the users is still for question-answer pairs. The third layer is for the specific movies from the movie types in the domain. The movie types are placed at the middle layer. As an example, the movie *Independence Day* belongs to the movie types *Science Fiction* and *Action* simultaneously, which is not possible in the previous architecture.

In order to get a desired movie, a user still needs to answer a set of questions. Then the system will retrieve a set of movie types. The user selects one movie type, and the movies belonging to that type will be displayed for further consideration. Thus, the whole problem solving procedure involves two selections. The first selection is made at the middle layer, while the second at the top layer. Each selection can get the feedbacks from the user if necessary. For instance, the user, based on the current answers to the questions, is not satisfied with the retrieved movie types. The user provides to the system information such

as *This movie type is not the one I like*, or *This movie type should get a higher score.* This feedback information will be captured by the system, and used as a valuable source for continuous learning. The same scenario applies to the second selection, which is made at the solution layer.

From the above example, we can see that the whole problem solving process in this new 3-layer architecture is a repetition composed of feedback, selection, and learning, until the retrieval result satisfies the user.

In addition to scaling-up and redundancy advantages, an added advantage of this architecture is that we can now represent a context sensitive case base. In this way, the second layer, which consists of problem descriptions, can be used to represent both problem and context layer, the latter representing different contexts in which problems occur. Under such conceptual representation, the third layer now contains the actual cases. A user can enter a problem's description in the form of feature-value pairs and then select the desired context in which to solve the problem. The second set of weights in turn can help rank the right case or cases for solving the problem. A set of features can simultaneously influence the contexts and the cases at the same time.

6.1. Rank computation and feedback learning

In this section, we extend our two layer network architecture to three layers. Our algorithms are again motivated by the back-propagation neural network computation (Hinton and Anderson, 1981; Rumelhart and McClelland, 1986). We introduce notations for different entities in figure 10. There are two sets of weights, similar to the weights in a 3-layer back-propagation neural network. Suppose that there are N features. For each feature F_i, there are m_i values, where $i = 1, 2, \ldots, N$. The case base contains J problems and K solutions. For the architecture shown in figure 10, 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 $i = 1, 2, 3, \ldots, I$. In the problem layer, we use C_j to represent each problem, where $j = 1, 2, 3, \ldots, J$. In the solution layer, we use S_k to represent each solution, where $k = 1, 2, 3, \ldots, K$.

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

6.2. Computation of a problem's score

Given the feature-value pairs selected by a user, the corresponding nodes at the feature-value pair layer are turned on (set to one). A problem's score is computed based on those selected feature-value pairs. For each problem C_j, its score is computed using the following formula:

$$S_{C_j} = \frac{2}{1 + e^{-\lambda * \sum_{i=1}^{I} (V_{j,i} * X_i)}} - 1 \tag{6}$$

where $j = 1, 2, 3, \ldots, J$, S_{C_j} is the score of the problem C_j, and X_i is 1 if there is a connection between problem C_j and feature-value pair FV_i **and** FV_i is selected. Otherwise X_i is 0. This formula for computing problem score (and subsequent formula for solution score) is inspired by backpropagation techniques in neural network research.

6.3. Computation of a solution's score

After the problem scores are computed, the problems and their scores will be presented to the user for selection and confirmation. For the current **selected confirmed** problem, the user might select its corresponding solutions. The computation of a solution's score is again similar to the computation of an output in a back-propagation neural network,

$$S_{S_k} = \frac{2}{1 + e^{-\lambda * \sum_{j=1}^{J} (W_{k,j} * S_{C_j} * \alpha)}} - 1 \tag{7}$$

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

In the above formula, α is a new parameter we introduce into our learning network. We call it the *bias factor*. The reason for us to introduce a bias factor is that in the architecture shown in figure 10, a user should first select which problem at the problem layer is the most desired one based on her/his current preference. This information needs to be reflected in the subsequent computation of the solution scores. We expect the selected problem to have a higher bias factor than the unselected ones, contributing more in the final solution scores. Thus the solutions of the selected problems *might* have relatively higher scores.

6.4. Delta learning rule for solutions and problems

The computation of the learning delta value is first done at the solution layer. We only compute the delta values for the solutions associated with the current **selected and confirmed** problem. The following formula is employed:

$$\delta_{S_k} = \frac{1}{2} * \left(D_{S_k} - S_{S_k}\right) * \left(1 - S_{S_k}^2\right) \tag{8}$$

where δ_{S_k} is the learning delta value for solution S_k, and D_{S_k} is the desired score for S_k.

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

$$\delta_{C_j} = \frac{1}{2} * \left(1 - S_{C_j}^2\right) * \sum_{k=1}^{K} \left(\delta_{S_k} * W_{k,j}\right) \tag{9}$$

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

6.5. *Weight adjustments*

After computing the learning delta values for weight adjustments, 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. They will be adjusted using the learning delta values and the problem scores. The formula for this adjustment is

$$W_{k,j}^{new} = W_{k,j}^{old} + \eta * \delta_{S_k} * S_{C_j} \tag{10}$$

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 C_j.

The weights attached to connections between the problems and the feature-value pairs will be adjusted next using the learning delta values as follows:

$$V_{j,i}^{new} = V_{j,i}^{old} + \eta * \delta_{C_j} * X_i \tag{11}$$

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 C_j and feature-value pair FV_i. X_i is 1 if there is a connection between problem C_j and feature-value pair FV_i and FV_i is selected by the user. Otherwise X_i is zero.

To fully evaluate this new architecture we have to conduct more experiments. Preliminarily, we expect that the network will consume more computational resources while have higher accuracy. This is because the middle layer, similar to the hidden layer in a neural network, can represent more elaborate classification hyperplanes.

7. Conclusions and future work

Our work aims to achieve the goal of continual maintenance of feature weights in the case retrieval process in CBR. The integration of a learning network into a CBR system makes continual maintenance possible. The weights can be learned through a process similar to neural networks.

The system has a number of areas to be improved. One obvious extension is to consider a three-layer rather than a two layer network, by splitting the features from cases, and splitting a case into problems and solutions. We are still in the process of evaluating the pros and cons of this new architecture. However, we expect that there will be performance degradation while accuracy improvements.

Also, although in our experimental tests nearly all the cases converge to their desired scores, we actually encounter divergence several times due to the interactions among different cases and among different features. The effect of such interaction can be reduced by introducing stronger bias factors into the system. Another important part is to study the effect of changing the order of queries presented to the system, on the eventual convergence

of the system. Our intuition is that a different order on the queries will result in different time efficiency, but such differences will not change the convergence behavior.

Acknowledgments

The authors would like to thank our funding partners: NSERC, BC ASI, Rogers Cablesystems Ltd., Canadian Cable Labs Fund, IRIS/PRECARN and Simon Fraser University.

Note

1. Here we assume discrete values for features. We assume that continuous feature values can be discretized according to the ranges of the values.

References

Aamodt, A. and Plaza, E. (1993). Foundational Issues, Methodological Variations, and System Approaches, *Artificial Intelligence Communications*, 7(1), 39–59.

Aha, D.W. and Breslow, L. (1997). Refining Conversational Case Libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning*, *ICCBR-97* (pp. 267–276). Providence RI, USA.

Bareiss, R. (1988). Protos: A Unified Approach to Concept Representation, Classfication and Learning, Ph.D. Dissertation, Technical Report, Department of Computer Science, University of Texas at Austin.

Bonzano, A., Cunningham, P., and Smyth, B. (1997). Using Introspective Learning to Improve Retrieval in CBR: A Case Study in Air Traffic Control. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97* (pp. 291–302). Providence RI, USA.

Czerwinski, M., Nguyen, T., and Lee, D. (1993). Compaq Quicksource—Providing the Consumer with the Power of AI. *AI Magazine*, 14, 50–60.

Fox, S. and Leake, D.B. (1995). Learning to Refine Indexing by Introspective Reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 430–440). Montreal, Canada.

Hinton, G. and Anderson, J. (1981). *Parallel Models of Associative Memory*. Potomac, MD: Lawrence Erlbaum.

Keogh, E., Blake, C., and Merz, C.J. (1998). UCI Repository of Machine Learning Databases.

Kolodner, J.L. (1993). *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann.

Koton, P. (1988). Using Experience in Learning and Problem Solving. Technical Report, Massachusetts Institute of Technology.

Leake, D.B. (1996). CBR in Context: The Present and Future. In David B. Leake (Ed.), *Case-Based Reasoning, Experiences, Lessons and Future Directions* (pp. 1–30). Menlo Park CA: AAAI Press/The MIT Press.

Leake, D.B., Kinley, A., and Wilson, D. (1995). Learning to Improve Case Adaptation by Introspective Reasoning and CBR. In *Proceedings of the First International Conference on Case-Based Reasoning*, Sesimbra, Portugal (pp. 229–240), Berlin: Springer-Verlag.

Leake, D.B. and Ram, A. (1993). Goal-Driven Learning: Fundamental Issues (A Symposium Report). *AI Magazine*, 14(4), 67–72.

Perini, A. and Ricci, F. (1995). An Interactive Planning Architecture: The Forest Fire Fighting Case. In Malik Ghallab (Ed.), *Proceedings of the 3rd European Workshop on Planning*, Assissi, Italy (pp. 292–302), September ISO Publishers.

Racine, K. and Yang, Q. (1997). Maintaining Unstructured Case Bases. In *Proceedings of the Second International Conference on Case-Based Reasoning, ICCBR-97*, Providence RI (pp. 553–564).

Ram, A. and Cox, M. (1993). Introspective Reasoning Using Meta-explanations for Multistrategy Learning. In R. Michalski and G. Tecuci (Eds.), *Machine Learning: A Multistategy Approach*. San Mateo: Morgan Kaufmann, IV, 349–377.

Rumelhart, D.E. and McClelland, J.L. (Eds.). (1986). *Parallel Distributed Processing*. Cambridge, MA: MIT Press.

Smyth, B. and Keane, M.T. (1995). Remembering to Forget. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence* (pp. 377–382), Montreal, Canada.

Watson, I. (1997). *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. San Mateo: Morgan Kaufmann.

Yao, B. and He, Y. (1994). A Hybrid System for Case-Based Reasoning. In *World Congress on Neural Networks (Volume IV). 1994 International Neural Network Society Annual Meeting*, San Diego (pp. 442–446).