Towards Lifetime Maintenance of Case Base Indexes for Continual Case Based Reasoning

Zhong Zhang[1] and Qiang Yang[2]

[1] Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
zhangz@cs.rpi.edu
[2] School of Computing Science
Simon Fraser University Burnaby
B.C., V5A 1S6, Canada
qyang@cs.sfu.ca
http://www.cs.sfu.ca/cbr
Tel: 604-291-5415
Fax: 604-291-3045

Abstract. One of the key areas of case based reasoning is how to maintain the domain knowledge in the face of a changing environment. During case retrieval, a key process of CBR, feature-value pairs attached to the cases are used to rank the cases for the user. Different feature-value pairs may have different importance measures in this process, often represented by feature weights attached to the cases. How to maintain the weights so that they are up to date and current is one of the key factors determining the success of CBR. Our focus in this paper is on the lifetime maintenance of the feature-weights in a case base. Our task is to design a CBR maintenance system that not only learns a user's preference in the selection of cases but also tracks the user's evolving preferences in the cases. Our approach is to maintain feature weighting in a dynamic context through an integration with a learning system inspired by a back-propagation neural network. In this paper we explain the new system architecture and reasoning algorithms, contrasting our approach with the previous ones. The effectiveness of the system is demonstrated through experiments in a real world application domain.

1 Introduction

Case based reasoning (CBR) has enjoyed tremendous success as a technique for solving problems related to knowledge reuse, but the maintenance issue of case bases has remained an open problem. In fact, defining the case knowledge is just the first step in the long life-cycle of a knowledge base application. In today's industrial environment, new cases are entered at a very fast rate. 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

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

	Problem Desc.				Solution
	Loan Status	Monthly Income	Job Status	Monthly Repayment	Bank Decision
Case 1	good	42 units	salaried	2 units	extend loan
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	...

interest of their users. How to evolve a case base continually in an automated manner is becoming an urgent task of the knowledge base industry.

Our research is directly motivated by our work in a Cable-TV troubleshooting domain. In this domain, typical equipment troubleshooting procedures are captured as cases and stored in a case base. After authoring an initial case base, we found that new equipment and services arrive at a very rapid rate, necessitating the case base manager in the Cable-TV company to constantly adjust the case base. In the adjustment processes new cases arrive in order to account for new equipment and services, and at the same time the weighting values attached to case features must be adjusted as well to conform to the ever changing trend in customer service.

Consider the elements of a case base. How to retrieve knowledge using cases depends on how the case features are selected and weighted. Feature indexing involves determining which features of a case will be used to facilitate its retrieval. The features associated with a case are combinations of its important descriptors, which distinguish one case from the others. In practice, besides using column names to designate features (see Table 1), features and values are sometimes presented to the user in the form of questions and answers.

Upon receiving an input problem, weights attached to the features and their values are used to compute a distance measure between cases. One of the most popular methods is the *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. For a case C and a query Q, the following formula is used to compute the distance between them[WA95].

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

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 similarity between an input problem and cases can be then assessed using this expression as a basis.

2 Previous Work

In a nutshell, the problem we attack is how to maintain the features and weights for a case base in a multi-user, changing and complex environment.

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 problem or case at hand at any given time, but there is no prior knowledge on which ones is actually useful to the reasoner currently. For any given set of weights attached to the feature-value-case combinations, the users of the system can provide feedback on the outcome of the solutions provided through a feedback process. Our tasks are: to acquire the feature-weights after a user has used the system for a certain period of time; to adapt the feature-weights to a user's preferences with time, and to allow different users to have different weights; to continuously track a user's changing preferences for the cases in the case base and to update the weights correspondingly to reflect the change; to allow the influence a feature-value has on the selection of a case to be dependent to the values of other features in the case base; In other words, the feature weights are context dependent.

Researchers in the past have focused on various aspects of the problem. Aha in [Aha91] introduces a series of case based learning (CBL) algorithms. They are aimed at situations where the feature weights vary greatly among the predictor features, for instance, the irrelevant features exist. Feature weights 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. It was pointed out, however, that CBL4 lacks of ability to address the context sensitive case information. It is also limited to the application of goal-feature prediction from other sets of features and values. The feature-weight update procedures point to a system similar to a neural network type of computation, but the author did not pursue this direction further.

Another closely related approach is introspective learning, which is also based on monitoring the run-time process of a reasoner[FL95,DLW95,ROS95,ABS97]. Recent work by Bonazno et al. in [ABS97] demonstrates such a 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 method uses 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. The authors reported good result based on their empirical tests. It is claimed that failure-oriented rather than the success-oriented learning contributes most

to this result. They also state that their 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 one that provides coverage not provided by other cases in a case base[SK95].

These two papers inspired us to look deeper in the issue of case base maintenance. Both systems allow the system to acquire new feature-weights from feedback from the CBR result. We wish to move in this direction one step further, by providing a general architecture as well as well-founded algorithms to address the lifetime maintenance problems.

3 The Maintenance System

3.1 A Three-Layer Architecture

The structure of a case base can be conceptualized as a two-layer structure, where the feature-values form one layer and the cases another. The feature-value layer is connected to the case layer through a set of weights to be maintained.

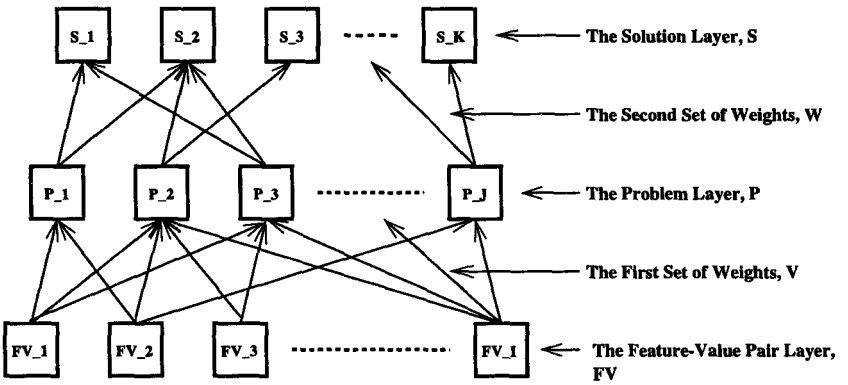

Fig. 1. New Structure of A Case Base

We now extend the original two-layer structure of a case base into a three-layer structure, taking the two-layer architecture as a special case. In the case layer, we extract the solutions from each case, and put them onto a third layer. This makes it possible for different problems to share a solution, and for a problem to have access to alternative solutions. An important motivation for this separation of a structure of a case is to reduce the redundancy in the case base. Given N problems and M solutions, a case base of size $N \times M$ is now reduced to one with size $N + M$. This approach eases the scale-up problem and helps make the case base maintenance problem easier, since when the need arises, each problem and solution need be revised only once.

In order to make this change possible, we introduce a second set of weights, 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. Initially, there is a weighted connection from every feature-value pair to every problem in the problem layer, and from every problem to every solution in the third layer.

In addition to scaling-up and redundancy, an added advantage of this structure is that we can now represent a context sensitive case base using the three-layer structure. Using this method, the second layer which consists of problem descriptions can be used to represent both the problem and a context layer, the latter representing different contexts in which problems occur. A user can then enter the problem descriptions 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 dealing with the problems. A set of features can simultaneously influence the contexts and the problems at the same time.

The architecture we have presented, and the learning algorithms that follow, can be seen as a balance in between the neural and Bayesian models[RN95],[Nea96]. In CBR, if we consider that a user's behavior is encoded in the weights assigned with the feature-value pairs of all the cases in a case base, 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. In addition, the cases, in the form of problem descriptions and contexts and their solutions, represent experiential knowledge rather than encoded probability variables.

3.2 Weight Maintenance

We introduce notations for different entities in Figure 1. There are two sets of weights, similar to the weights in a three-layer back-propagation neural network. Suppose that there are N features. For each feature F_i, there are m_i values, where $i = 1, 2, \cdots, N$. The case base contains J problems and K solutions. For the structure shown in Figure 1, 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 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 P_j if S_k is a potential solution to P_j.

Computation of A Problem's Score Given the input feature-value pairs, the corresponding first layer nodes are considered turned-on. A problem's score

is computed first based on those feature-value pairs as follows. For each problem P_j, its score is computed using the following formula:

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

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.

Formula 2 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 Section 2.

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

As soon as the score of a solution is computed, it will be presented to a user for his judgment. If the user thinks that this solution is the right one and it has an appropriate score, he can confirm by claiming success. Otherwise, a failure can be registered by the system. 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 algorithm, 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. In that case, 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 be 71, 73, or some other values as long as the specified value is less than 80.

The computation of the learning delta value is first done at the solution layer. These delta values are used to update the weights in that layer. In the computation of solution scores, we only compute the learning delta values for the solutions associated with the current **selected and confirmed** problem. The following formula is employed.

$$delta_{S_k} = \frac{1}{2} * (D_{S_k} - S_{S_k}) * (1 - S_{S_k}^2) \tag{3}$$

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 learning delta values are then propagated back to the problem layer. The computation of the learning delta value at this layer is similar to Formula 3.

$$delta_{P_j} = \frac{1}{2} * (1 - S_{P_j}^2) * \sum_{k=1}^{K} (delta_{S_k} * W_{k,j}) \tag{4}$$

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 $\sum_{k=1}^{K}(delta_{S_k} * W_{k,j})$.

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 3, and the problem scores computed in Formula 2. The formula for this adjustment is:

$$W_{k,j}^{new} = W_{k,j}^{old} + \eta * delta_{S_k} * S_{P_j} \tag{5}$$

where $W_{k,j}^{new}$ is the new weight to be computed, $W_{k,j}^{old}$ is the old weight attached to the connection between solution S_k and problem P_j, and η is the learning rate. The weights attached to the problem layer is computed in a similar manner.

Our system also gives the user an opportunity to judge whether a problem has the desired score after it computed the problem scores. For this situation, the system uses the following formula to compute the new weights.

$$V_{j,i}^{new} = V_{j,i}^{old} + \frac{1}{2} * \eta * (DS_{P_j} - S_{P_j}) * (1 - S_{P_j}^2) * X_i \tag{6}$$

where each symbol represents the same as above. 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 adjustments are local only to a selected problem.

A Simple Example We consider the small case base shown in Table 1 which is adapted from [Wat97]. Each row of the table represents the loan information about a person. We select columns 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 2, 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 cases are ranked, suppose that a user is not satisfied with the score of Case 2. Instead, the user sets its score to 0.8. After receiving this response, the system will 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 before all cases reflect the user's preferences in case ranking.

3.3 Learning Process

Contrasting our case base maintenance algorithm with that of a back-propagation neural network, we see that a traditional neural network processes the sample datum one by one, in which process the weights connecting adjacent layers are adjusted. This process will have to repeat several times in order to get the final satisfactory goal. In our case base maintenance framework, however, we assume that there is no explicit training 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 long run, it is expected to converge to a user's preferences. The interactions between the feature-value pairs often make it impossible to precisely model a user's preference in one pass. Therefore, the adjustment for the user's preferences requires an iterative process.

4 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 hypotheses:

1. the feature-weight maintenance system can learn a user's preferences after sufficient interaction between the user and the system;
2. the feature-weight maintenance system can be affected by the interactions between the different input vectors. In particular, the fewer the interactions there are the shorter it will take for the system to converge to the user's behavior;
3. for real-world applications, where the frequently occurring problems are concentrated, the system can converge quickly to the desired feature-weight sets provided by domain experts.

The case base that is being used in a local Cable-TV Company is created using a CBR system which integrates the feature-weight maintenance functions in its problem resolution components. 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 $i = 1, 2, \cdots, 30$. The weights assigned initially to the individual question-answer pairs by the domain experts from the company. Table 2 shows one case in this case base.

In this domain, 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.

Table 2. Example of Case Representation in Cable-TV Domain

Question 1	Is the subscriber in pay status? (y)
Question 2	What type of problem is being experienced? (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.

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 of 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; these are the feature-value pairs that are not mutually exclusive by domain knowledge. 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.

In order to test the effectiveness of the learning ability of the feature-weight maintenance system, we execute the two copies of case base reasoning system side by side, 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, and use that to provide feedback to the weight learning modules. 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 approximate the ones produced by the first copy.

In our experiment, the learning process took 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 as 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

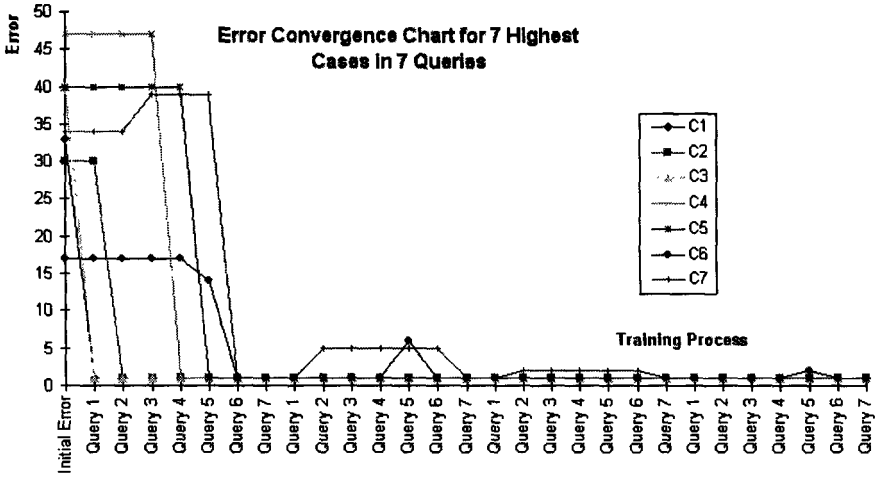

Fig. 2. Error convergence chart for the seven highest cases in the Cable-TV domain

computed scores produced 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 2. In the figure, the X-axis represents the time line of the training process as the queries are entered. The Y-axis represents the errors in case score, where an error is defined as the absolute difference between the obtained and desired case scores. In the experiment, 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 provide evidence to our claim made there.

We also would like 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' scores will still converge to their desired ones. Maybe the user will change his input distribution. Or a different user might come. All these will destroy the previous convergence state and trigger another learning process.

5 Conclusions and Future Work

Our work aims to achieve the lifetime maintenance of the feature weights in the case retrieval process in CBR. The need from practical applications of CBR systems motivated us to explore their dynamic nature. The research is motivated by our desire for a CBR system to be 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. The integration with a back-propagation neural network

not only makes the maintenance possible, but also exhibits important scaling up characteristics. This is because of the separation of problems and solutions, where each problem and solution need be represented only once, making the maintenance issues much easier to deal with. In addition, the middle layer can effectively be used to represent contextual information, solving a long-standing problem in CBR.

We also note that our system has its own limitations. 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 and different feature sets. The effect of such interaction can be reduced by introducing stronger bias factors into the system. In addition, one of the assumptions of our learning model is that the user of our system should be one person. If a different user comes to use the system, she might not satisfy the previous optimal case retrieval result. In the future, we hope to address these problems by introducing more effective learning and feedback control algorithms and architectures.

6 Acknowledgment

The authors are supported by grants from: Natural Sciences and Engineering Research Council of Canada (NSERC), an Ebco/Epic NSERC Industrial Chair Fund, BC Advanced Systems Institute and Canadian Cable Labs Fund.

References

[ABS97] P. Cunningham A. Bonzano and B. Smyth. 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*, pages 291–302, Providence RI, USA, 1997.

[Aha91] 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.

[DLW95] 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.

[FL95] 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.

[Nea96] R. M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag, 1996.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach.* Prentice Hall, Inc., 1995.

[ROS95] R. Edwards R. Ochlmann and D. Sleeman. Changing the viewpoint: Reindexing by introspective question. In *Proceedings of the 16th Annual Conference of the Cognitive Science Society*, pages 381–386. Lawrence-Erlbaum and Associates, 1995.

[SK95] 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.

[WA95] 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.

[Wat97] Ian Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann Publishers Inc., 1997.