CABOT: An Adaptive Approach to Case-Based Search

James P. Callan* Tom E. Fawcett+ Edwina L. Rissland++

Department of Computer and Information Science
University of Massachusetts • Amherst, Massachusetts 01003 • USA
callan@cs.umass.edu, fawcett@cs.umass.edu, rissland@cs.umass.edu

Abstract

This paper describes CABOT, a case-based system that is able to adjust its retrieval and adaptation metrics, in addition to storing cases. It has been applied to the game of OTHELLO. Experiments show that CABOT saves about half as many cases as similar systems that do not adjust their retrieval and adaptation mechanisms. It also consistently beats these systems. These results suggest that existing case-based systems could save fewer cases without reducing their current levels of performance. They also demonstrate that it is beneficial to distinguish failures due to missing information, faulty retrieval, and faulty adaptation.

1 Introduction

Many case-based reasoning (CBR) systems are designed to solve new problems by adapting solutions to similar, previously solved, problems. The proposed solution to a new problem can be inappropriate for a variety of reasons: I) the system lacks a similar past case, 2) the "wrong" case was retrieved, or 3) the retrieved case was not adapted properly to the new problem. Most case-based and instance-based systems are designed to handle just one or two of these errors. However, it is rare for a CBR system to handle all three.

It is important to distinguish between errors with differing causes because they require different solutions. It may be possible to compensate for an error in the retrieval mechanism by adding more information or altering the adaptation mechanism, but the resulting system might do more work or save more information than would otherwise be necessary. For example, the CHEF system [Hammond, 1986] may expend effort adapting a retrieved case when a different case could be adapted with less effort. Compensating for an error, rather than dealing with its causes, may also hamper future performance.

*Supported by a grant from Digital Equipment Corp.
+ Supported by a grant from GTE Laboratories Inc.
++ Supported by grants from the Air Force Office of Sponsored Research under contract AFOSR-90-0359; the National Science Foundation, contract IRI-890841; and GTE Laboratories Incorporated.

The work presented in this paper is based upon two hypotheses. The first is that it is useful for a case- or instance-based system to distinguish among errors due to a lack of information, a faulty retrieval mechanism, and a faulty adaptation mechanism. The second hypothesis is that a problem-solving system can profitably use *feedback* from its environment to distinguish among the three types of errors. Once this distinction can be made, it is assumed that error correction can be done using a standard machine learning technique for supervised learning.

These hypotheses have been tested in CABOT, a hybrid case-based reasoning/machine learning system that both reasons from cases and uses an inductive learning algorithm to adjust its retrieval and adaptation mechanisms. CABOT has been tested against a non-learning system, two case-based systems with fixed retrieval and adaptation mechanisms, and two inductive learning systems. Results from these tests suggest that existing case-based and instance-based systems could save fewer cases without reducing their current levels of performance.

2 Case-Based Reasoning

Although case-based reasoning systems vary significantly, those that are used for problem-solving share a common approach. When a new problem is encountered, the system first *retrieves* one or more cases that are similar to the new problem. Typically no case matches the new problem exactly, so the system must *adapt* one of the retrieved solutions to the new problem. Finally, the system may *add* the new problem and its solution to the case base, and it may adjust its indices.

Instance-based learning algorithms also share this approach to problem-solving, although they are sometimes considered distinct from case-based systems. When a new problem is encountered, one or more similar instances are retrieved. If their classifications differ, the differences must be resolved to provide a classification for the new instance. Finally, the system may add a new instance to its instance base. The primary differences between case-based systems and instance-based learning algorithms are the tasks to which they are applied, and the complexity of the information that they store; cases are typically more complex than instances.

The simplest approach to error-correction is to save *all* information, whether or not an error was made. The

1. Retrieve the stored problem state most like the current problem state, according to the Retrieval metric.
2. Select from the available successor states the one most like the retrieved successor, according to the Selection metric.
3. Implement the selected action.
4. Receive feedback from the Oracle about which successor was best.
5. If CABOT did not select the best successor, perform error-correction:
   (a) Try to adjust the Retrieval metric. If successful, go to 1.
   (b) Try to adjust the Selection metric. If successful, go to 1.
   (c) Add the current problem state and the best successor to the case base. Go to 1.

Figure 1: CABOT's problem-solving cycle.

GINA program [De Jong & Schultz, 1988] for playing OTHELLO adopts this approach, saving every board that it sees. One might expect GINA eventually to become swamped with cases. Instead, the number of unique boards encountered by GINA in its play with opponents eventually stabilized at a small fraction of the number of possible boards.

Aha & Kibler (1989) have experimented with a family of *instance filtering* algorithms. The *Proximity* algorithm saves all instances, while the *NTGrowth* algorithm discards instances that would be classified correctly or that appear to be noisy *(i.e.,* their classifications conflict with the rest of the data). In their tests on noisy data, the two algorithms were roughly equal; thus, equal performance was gained from fewer instances.

Both CYRUS [Kolodner, 1983] and PROTOS [Bareiss & Porter, 1987] dynamicaUy adjust the structure of their case bases, and hence their retrieval mechanisms. CYRUS tries to maintain an appropriate organization of cases based upon an internal metric, expressed in terms of the number of adherents and exceptions to a given memory organization packet (MOP). In contrast, PROTOS is given feedback by a benevolent teacher. The teacher provides the correct answer and may explain the relevance of individual features. The teacher also approves or rejects changes that PROTOS proposes, which prevents PROTOS from making serious mistakes. This feedback also enables PROTOS to prune its case-base by merging cases.

EACH [Salzberg, 1988] also stores exemplars (generalized, representative instances) to perform classification. When EACH encounters a new instance, the distance to each stored exemplar is measured. The closest exemplar determines the classification that EACH predicts. EACH generalizes and specializes its exemplars in response to classification successes and failures. In addition, EACH uses a weighted distance function, similar to CABOT's, which is adjusted after every classification error. Thus EACH tunes its retrieval mechanism in response to limited feedback from its environment.

CHEF [Hammond, 1986] adjusts its adaptation mechanism and also its retrieval mechanism by changing the way cases are indexed. CHEF receives very detailed feedback from a simulator, which it can analyze to identify the reason that an adapted case fails to meet its goals. After it repairs the adapted case, CHEF constructs demons that prevent it from making similar adaptation mistakes in the future. CHEF also indexes the repaired case according to the failures that the case avoids, as well as the goals the case satisfies.

Adding a case, adjusting the retrieval mechanism, and adjusting the adaptation mechanism are all ways of coping with failure. Few of the systems above make clear distinctions among causes of failure, and therefore it is rarely clear which mechanism should be adjusted.

Most systems that can adjust their retrieval or adaptation strategies depend upon detailed feedback from the problem-solving environment. Requiring detailed feedback limits the environments in which those systems can operate. In contrast, CYRUS optimizes an internal metric that does not use feedback from the environment. However, the inability to use feedback prevents a system from adjusting to the environment in which it operates. Therefore, an important research question is how a system can adjust both its retrieval and adaptation mechanisms using only limited feedback from the environment. The following sections present a system that addresses this question.

3 CABOT

The CABOT program was developed to investigate the problem of using limited feedback from a problem-solving environment to distinguish different types of errors in case-based reasoning. CABOT is designed for state-space search, where states are represented as feature vectors and problem-solving consists of repeatedly selecting and then executing one of a set of actions. Figure 1 outlines CABOT's problem-solving cycle. Cases consist of pairs of states $(\vec{S_p}, \vec{S_c})$, where $\vec{S_c}$ (the *child)* is the desired successor to $\vec{S_p}$ (the *parent).*

State-space search is performed by starting at an initial state and repeatedly selecting successor states until a goal state is reached. The object of CABOT's reasoning is solely to determine, at each state in the search, the best successor state. CABOT does this by performing a restricted form of adaptation called *selection:* when a case is retrieved, it is used to select one from a set of successor states. By improving the quality of decisions at each step in the search, CABOT improves its problem-solving performance.

The feedback available to CABOT is qualitative. A domain-dependent Oracle identifies the best successor state, but it does not explain its reasoning, nor does it score or order the rest of the available successors. CABOT's task is to use this limited feedback to improve its problem-solving performance. When CABOT's selection differs from that of the Oracle, its decision is considered a failure that it must correct. CABOT first tries to adjust its retrieval metric. If the retrieval metric cannot or should not be changed, CABOT tries to adjust its selection metric. Changes to the retrieval and selection metrics are both subject to an *assured consistency* condition that only allows changes if the two metrics remain consistent with each other. CABOT checks for assured consistency by testing any changes on a random sample of problems that it has previously encountered; if a change to one of the metrics decreases accuracy, the change is discarded. If neither metric can be changed, CABOT adds a new case.

The next three sections discuss each step in detail.

3.1 Retrieval

Retrieval consists of identifying the case $R = (\vec{R_p}, \vec{R_c})$ whose $\vec{R_p}$ is closest to the current problem state $\vec{S_p}$. Distance is determined by a weighted distance function d_R (the *retrieval metric)* whose weight vector $\vec{W_R}$ can be adjusted during error correction. No similarity threshold is used; retrieval always returns a case.

3.2 Selection

It is rare for the retrieved successor state to exactly match any of the available successors of the current problem state. However, the exact pair of states is less important than their relationship; the pair of states encodes a transformation of the problem that presumably reduces the distance to a goal. Selection consists of identifying the successor state whose transformation of the current problem is most similar to the transformation represented by the retrieved case.

The transformation of the problem from one state to another is represented by the difference between their feature vectors. The vector $\vec{\Delta R_c} = \vec{R_p} - \vec{R_c}$ represents the *change* in feature values that should ideally occur between the current problem state and the selected successor state. The change that actually occurs between the current problem state and each successor state $\vec{S_c}$ is represented as $\vec{\Delta S_c} = \vec{S_p} - \vec{S_c}$. These relationships are illustrated in Figure 2.

Selection consists of identifying the successor state whose $\vec{\Delta S_c}$ is closest to $\vec{\Delta R_c}$. Distance is determined by a weighted, signed distance function *ds* (the *selection* metric), whose weight vector *Ws* can be adjusted during error correction, *ds* is defined as follows:

$$d_S(\vec{X}, \vec{Y}) = \sum W_{S_i} \cdot \mathbf{sign}(X_i - Y_i) \cdot (X_i - Y_i)^2$$

Ties are broken arbitrarily. No similarity threshold is used; selection always selects a successor state.

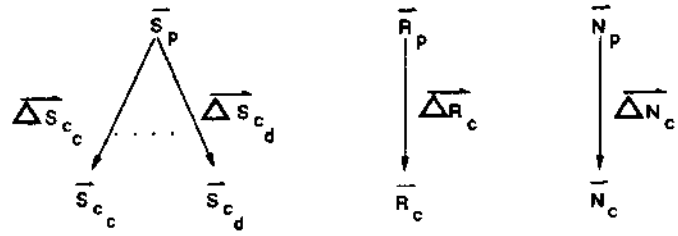

Figure 2: The relationships among cases in retrieval and selection. $\vec{S_{c_c}}$ is the successor chosen by CABOT. $\vec{S_{c_d}}$ is the successor chosen by the Oracle.

3.3 Error Correction

After CABOT selects a successor state and carries out the associated action, it asks the Oracle to identify the desired action. If the Oracle agrees with CABOT's choice, no error correction is performed. Otherwise, CABOT considers its choice to be wrong and tries to correct itself. Error correction consists of the following:

1. CABOT checks to see if it retrieved the wrong case. It does so by conducting retrieval again with the added constraint that the retrieved case must cause it to select the desired successor state. The resulting case $N = (\vec{N_p}, \vec{N_c})$ is called a *near miss.* If no such case is found, CABOT proceeds to step 2.
   If a near miss is found, CABOT tries to adjust the weighted Euclidean distance function d_R so that the near miss state is closer than the retrieved state to the current problem state. The desired distance relationship is:

$$d_R(\vec{S_p}, \vec{N_p}) < d_R(\vec{S_p}, \vec{R_p})$$

   This relationship may be expressed in terms of the retrieval weight vector $\vec{W_R}$:

$$\vec{W_R} \cdot \overrightarrow{((S_{p_i} - N_{p_i})^2)} < \vec{W_R} \cdot \overrightarrow{((S_{p_i} - R_{p_i})^2)}$$

$$\vec{W_R} \cdot \overrightarrow{((S_{p_i} - N_{p_i})^2 - (S_{p_i} - R_{p_i})^2)} < 0 \qquad (1)$$

   CABOT adjusts $\vec{W_R}$ by applying the absolute correction rule [Nilsson, 1965] to Equation 1.
   Finally, CABOT verifies that the retrieval and selection metrics remain consistent, as discussed earlier. If they do, error correction terminates. Otherwise, the adjustments to the retrieval weight vector are discarded, and CABOT proceeds to step 2.

2. CABOT assumes that it retrieved the right case, but that the selection metric is wrong. It tries to adjust the selection weight vector, in a manner similar to that used for the retrieval weight vector. Adjustment is based upon the desired relationship:

$$d_S(\vec{\Delta R_c}, \vec{\Delta S_{c_d}}) < d_S(\vec{\Delta R_c}, \vec{\Delta S_{c_c}})$$

   $\vec{S_{c_c}}$ is the successor state chosen by CABOT. $\vec{S_{c_d}}$ is the desired successor state.

CABOT verifies that the retrieval and selection metrics remain consistent, as discussed earlier. If they do, error correction terminates. Otherwise, the adjustments to the selection weight vector are discarded, and CABOT proceeds to step 3.

3. CABOT assumes that its error was due to missing information, and adds a new case. The new case consists of the current problem state and the desired successor state.

Weights used in the retrieval and selection metrics ($\overrightarrow{W_R}$ and $\overrightarrow{W_S}$) are initially set to 1.0 and are adjusted by the error correction process. CABOT attempts to adjust its retrieval metric (step 1) before attempting to adjust its selection metric (step 2) because retrieval determines the cases that are used in selection. This ordering is designed to make the retrieval metric converge before the selection metric.

4 An Experimental Domain: OTHELLO

CABOT has been been tested on OTHELLO, a two-player board game. Players alternate moves, and there are usually 60 moves in a game. Although the rules of OTHELLO are simple, the search space contains approximately 10^{50} nodes, and the strategies can be quite complex. OTHELLO was selected as a domain for CABOT because of its large search space, because it has been used as a domain by researchers in artificial intelligence [Rosenbloom, 1982; Lee & Mahajan, 1988; De Jong & Schultz, 1988], and because of the availability of immediate feedback after every move. Problem-solving in OTHELLO consists of deciding which move to make next; on each cycle, CABOT selects a move, makes it, receives feedback, and possibly performs error correction.

The OTHELLO Oracle makes decisions by conducting a minimax search, using alpha-beta pruning, to a depth of 3-ply and then evaluating search states with a polynomial evaluation function. The Oracle shifts to exhaustive search for the last 8 moves of the game.

Raw board positions are often considered too specific a representation for OTHELLO game states, so it is common to represent them with vectors of more abstract features. A feature is a numeric function that measures some important characteristic of a board position. CABOT and its opponents use a set of features common to OTHELLO-playing programs: *mobility, potential mobility, corner squares, X squares,* etc. [Rosenbloom, 1982] Hereafter, "board" will be used to denote this feature vector, rather than the actual raw board configuration. The OTHELLO Oracle uses a larger and more comprehensive set of features than is used by CABOT.

4.1 LATCBR: A Pure Case-Based Opponent

One of CABOT's opponents is a case-based reasoning system that uses a very simple version of a *claim-lattice* [Rissland & Ashley, 1987] to determine the best move. This opponent, called LATCBR, orders the cases into a lattice in which the cases closest to the root are those with maximal subsets of features exactly matching the current problem state. LATCBR contains no built-in indices to prune this set, nor does it have any way of judging which of the nodes at a given level are best for the player. It therefore treats all nodes occurring closest to the root as equal, and discards the rest. The cases contained in these nodes are all considered *retrieved*.

The claim-lattice was designed for comparing and contrasting competing alternatives. It does not offer a straightforward mechanism for selecting a single alternative, nor is there any *a priori* reason for selecting one alternative as best. The approach adopted for LATCBR is to adapt each case to the current situation by identifying the move that it matches most closely, and then make the choice that is recommended by the majority of the cases, with ties broken randomly. The degree of match between a case and a given move is determined by the number of features on which they exactly match.

LATCBR does not attempt to correct either its retrieval function or its adaptation function when it makes a mistake. When the Oracle disagrees with its chosen move, LATCBR adds a new case.

4.2 PIL: A Pure Inductive Learning Opponent

The PIL opponent uses an inductive learning algorithm to improve a heuristic evaluation function *h* over search states. PIL selects the successor to the current problem state for which *h* is greatest; ties are broken arbitrarily.

After each selection, PIL asks the Oracle to identify the desired successor. If the desired successor differs from PIL's selection, PIL considers its choice to be wrong and tries to correct *h*. It does so by adjusting *h* so that the desired successor state is rated more highly than all other successor states.

PIL's evaluation function *h* is a weighted sum of feature values. Weights in the weight vector $\overrightarrow{W_{PIL}}$ are initially set to 1.0. Qualitative feedback can be used correct the weights, as shown below:

$$\begin{aligned} h(\overrightarrow{S_{c_d}}) &> h(\overrightarrow{S_{c_c}}) \\ \overrightarrow{W_{PIL}} \cdot \overrightarrow{S_{c_d}} &> \overrightarrow{W_{PIL}} \cdot \overrightarrow{S_{c_d}} \\ \overrightarrow{W_{PIL}} \cdot \left(\overrightarrow{S_{c_d}} - \overrightarrow{S_{c_c}} \right) &> 0 \end{aligned} \qquad (2)$$

PIL adjusts the weight vector by applying the absolute correction rule [Nilsson, 1965] to Equation 2.

5 Experiments and Results

Four experiments were run to determine the relative effectiveness of the pure case-based (LATCBR), adaptive case-based (CABOT), and pure inductive learning (PIL) move selection strategies. The experiments were designed to examine three characteristics of each strategy:

- Effectiveness: How effective was it for playing OTHELLO? Effectiveness was measured by the win/loss ratio, and by the average discs taken per game.
- Decision Accuracy: How often did it agree with the Oracle's choice?
- Growth of Case Base: For the case-based systems, how quickly did the case base grow?

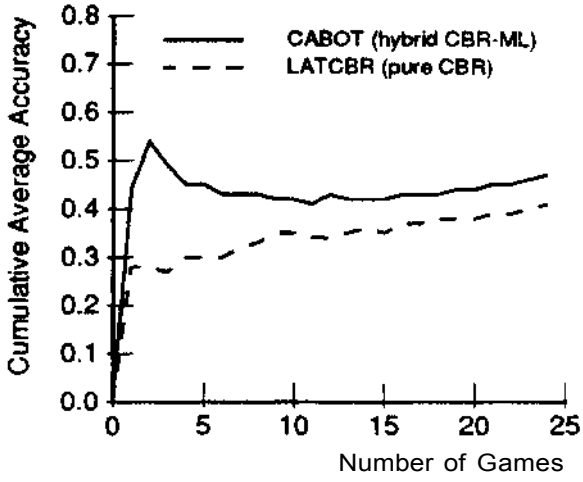

Figure 3: Accuracy of decision-making.

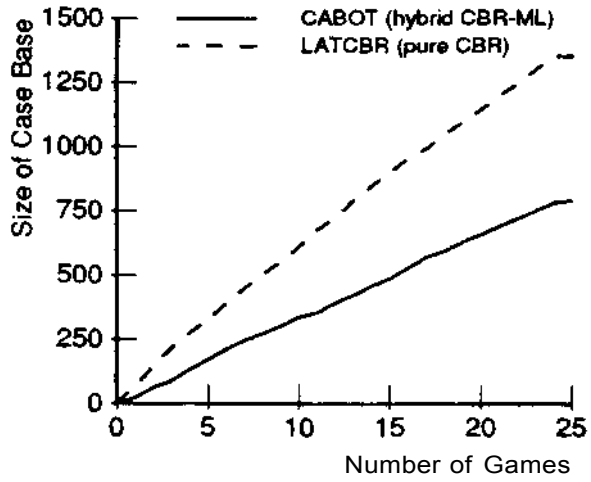

Figure 4: Growth of case base during game playing.

Table 1: Games won and lost by each player during the four tournaments.

Opponents	1st player	2nd player	tied
CABOT vs PIL:	58	39	3
CABOT vs LATCBR:	14	10	0
CABOT vs Oracle:	1	24	1
PIL vs LATCBR:	66	32	2

Table 2: Average discs per game won by each player during the four tournaments.

Opponents	1st player	2nd player
CABOT vs PIL:	34.1	26.6
CABOT vs LATCBR:	33.1	29.2
CABOT vs Oracle:	4.9	55.7
PIL vs LATCBR:	38.1	25.9

Each tournament began with empty case bases and unweighted linear threshold units (i.e., all weights set to 1.0). At the beginning of each game, one player was randomly selected to make the first move. Each player was permitted to learn from both its own moves and its opponent's moves throughout the tournament. By training on both sets of moves, a learning program receives balanced training because it can train on better game states than it might achieve on its own.

Games were played until either 100 games had been played, or until the Oracle's best game had been encountered. The latter stopping condition is important for case-based competitors. If a case- or instance-based game-playing program is permitted to observe an opponent that follows a fixed strategy, it will eventually memorize that opponent's best game. Once the opponent's best game is memorized, the case-based program cannot lose more than 50% of the subsequent games. When *both* opponents are case-based and both are learning from a fixed Oracle, they will eventually begin playing a single game repeatedly. All of the tournaments reported below were ended if this condition occurred. The results from the four tournaments are summarized in Tables 1 and 2.

Table 1 shows the number of games won, lost and tied by each playeT during the four tournaments. It demonstrates that CABOT's performance is superior to that of the pure case-based system (LATCBR) and the pure inductive learning system (PIL) in playing OTHELLO. It also shows that the performance of PIL is superior to the performance of LATCBR. This result is unexpected because the training data are not linearly separable. We expected PIL to perform more poorly, and are unable to explain why it did not. The performance of CABOT against the Oracle is not surprising. However, it is interesting to note that it took CABOT just 26 games to discover the Oracle's best game. After that point, the two opponents began playing the same game repeatedly, each losing exactly 50% of the games.

OTHELLO players are rated both on how often they win and on the magnitude of the score. Table 2 reports the average number of discs won by each player in each tournament. The figures confirm that the hybrid system is better at selecting moves than either of its pure opponents.

Figure 3 shows the decision accuracy of CABOT and LATCBR. Decision accuracy is the percentage of time that a player's chosen move is judged to be the best by the Oracle. This graph shows a running average of decision accuracy through 25 games, and confirms that CABOT's performance is better than LATCBR's because its decisions tend to be more accurate. The training data for both systems are identical, so the only factor that can account for this difference is CABOT's improved retrieval and adaptation strategies. The improvement in CABOT's retrieval and adaptation strategies occurs quickly, usually within the first game or two. In contrast, LATCBR is initially error-prone but its accuracy continues to improve as its case base grows. Figure 3 shows that the difference between the two accuracies decreases as the number of games increases. This confirms the intuition that having enough cases is asymptotically as good as having a smaller number of cases with better retrieval and adaptation strategies.

Superior performance is one result of adjusting retrieval and adaptation strategies. Another result is the difference in the growth of the case base. Figure 4 shows

this growth during the CABOT vs LATCBR tournament. The CABOT case base tends to be about half the size of the LATCBR case base. Neither case base shows any sign of stabilizing. Continued growth may be due to the small number of games that have been played. The search space for OTHELLO is large (about 10^{50} legal boards), so it may be unrealistic to expect either case base to stabilize within 24 games.

Results vary from tournament to tournament, because of the effects of learning, and because the Oracle makes random choices when it has two or more equally good moves. We have observed variation in the specific numbers of games won and lost by each move selection strategy. However, the relative performance of each strategy is consistent. CABOT is superior both to LATCBR and P1L, and PIL is superior to LATCBR.

We have also tested CABOT against opponents that use other inductive learning techniques to guide move selection. One of these opponents used the ID3 algorithm for building decision trees [Quinlan, 1986]. One problem with using ID3 for this task is that the OTHELLO boards are described by numeric features, whereas ID3 is intended for symbolic attributes. We experimented with several methods of converting numeric data to symbolic attributes, including Quinlan's method (1986), Vapnik's method (1982), and our own manual method. None of these methods for creating symbolic attributes was both computationally feasible *and* more effective than the weighted sum evaluation function of the PIL system.

6 Conclusions

CABOT was developed to investigate the hypothesis that case-based reasoning systems would benefit from the ability to dynamically adjust their retrieval and adaptation mechanisms. CABOT reasons from cases for the purpose of guiding state-space search. State-space search is different than traditional CBR tasks because the problem-solver's task is to repeatedly identify the successor state to explore next. As a result, CABOT performs a restricted form of adaptation, in which the retrieved case is used to select one of a set of known alternatives.

One of the assumptions of this work is that feedback from the environment is necessary for making proper adjustments to retrieval and selection mechanisms. The feedback provided to CABOT and its opponents consists of the best move as identified by the Oracle. However, no information is provided that might suggest *why* one move was better than another. This feedback is quite weak when compared with the detailed information available to systems like CHEF [Hammond, 1986] and PROTOS [Bareiss & Porter, 1987].

The empirical results reported in this paper confirm several hypotheses. First, they demonstrate that the ability to tune retrieval and selection mechanisms leads to both a smaller case base *and* better performance. They suggest that traditional CBR systems may be saving more cases than they really need. Second, they demonstrate that detailed feedback from the environment, while useful, is not always necessary to make these adjustments. Sometimes it is enough just to know the desired result. Finally, the performance of CABOT when played against pure inductive and pure case-based learning opponents illustrates that a hybrid architecture can overcome the weaknesses of opponents of each type.

Acknowledgements

Support for this work was provided by the Office of Naval Research through a University Research Initiative Program, under contracts N00014-86-K-0764 and N00014-87-K-0238. Comments by David Aha and Bernard Silver were especially valuable. We thank Jeff Clouse for providing the Oracle program.

References

Aha, D. W., & Kibler, D. (1989). Noise-tolerant instance-based learning algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 794-799). Detroit, Michigan: Morgan Kaufmann.

Bareiss, R., & Porter, B. W. (1987). Protos: An exemplar-based learning apprentice. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 12-23). Irvine, CA: Morgan Kaufmann.

De Jong, K. A., & Schultz, A. C. (1988). Using experience-based learning in game playing. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 284-290). Ann Arbor, MI: Morgan Kaufman.

Hammond, Kristian (1986). Learning to anticipate and avoid planning problems through the explanation of failures. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 556-560). Philadelpha, PA: Morgan Kaufmann.

Kolodner, J. L. (1983). Maintaining organization in a dynamic long-term memory. *Cognitive Science, 7.*

Lee, K. F., & Mahajan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence, 36,* 1-25.

Nilsson, N. J. (1965). *Learning machines.* New York: McGraw-Hill.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning,* i, 81-106.

Rissland, E.L., & Ashley, K.D. (1987). A Case-Based System for Trade Secrets Law. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 60-65). Milan, Italy: Morgan Kaufmann.

Rosenbloom, P. (1982). A world-championship-level othello program. *Artificial Intelligence, 19,* 279-320.

Salzberg, S. (1988). *Exemplar-based learning: Theory and implementation* (TR-10-88). Cambridge, MA 02138: Harvard University, Center for Research in Computing Technology.

Vapnik, V. N. (1982). *Estimation of dependencies based upon empirical data.* New York: Springer Verlag.