Adaptive Case-Based Reasoning

James P. Callan¹
Tom E. Fawcett²
Edwina L. Rissland³
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003 USA

Abstract

This paper describes CABOT, a case-based system for the game of Othello that is able to adjust its retrieval and adaptation mechanisms, in addition to storing cases. Experiments with CABOT show that it saves about half as many cases as similar systems that do not adjust their retrieval and adaptation mechanisms. It also consistently beats these systems. CABOT has been tested against, and beats, an opponent that uses an inductive learning technique to select moves. These results suggest that existing case-based and instance-based systems could save fewer cases without reducing their current levels of performance. They also demonstrate that it is beneficial to distinguish three different types of errors: those due to missing information, those due to faulty retrieval, and those due to faulty adaptation.

1 Introduction

When a case-based reasoning system makes a mistake, the error may be due to missing information, a faulty retrieval mechanism, or a faulty adaptation mechanism. Most case-based and instance-based systems are designed to handle just one of these errors, and a few are designed to handle two of them. However, none of the published systems are capable of handling all three types of errors.

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 significant effort adapting a retrieved case when a different case could be adapted with less effort. Compensating for an error, rather than fixing it, may also hamper future performance on unseen problems.

The work presented in this paper is based upon two hypotheses. The first is that it is useful for a case-based or instance-based system to distinguish among errors due to

¹Supported by a grant from the Digital Equipment Corporation
²Supported by a grant from GTE Laboratories Incorporated
³Supported by grants from the Air Force Office of Sponsored Research under contract AFOSR-90-0559; the National Science Foundation, contract IRI-890841; and GTE Laboratories Incorporated.
• a lack of information,
• a faulty retrieval mechanism, and
• a faulty adaptation mechanism.

The second hypothesis is that a problem-solving system must 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 an othello-playing program called CABOT. CABOT is a hybrid case-based reasoning/machine learning (CBR-ML) 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 opponent, two case-based opponents with fixed retrieval and adaptation mechanisms, and two inductive learning opponents. Results from these tests suggest that existing case-based and instance-based systems could save fewer cases without reducing their current levels of performance.

Section 2 reviews some of the case- and instance-based systems that adjust retrieval or adaptation mechanisms. Section 3 describes CABOT and its opponents, with particular emphasis on how CABOT distinguishes between the three types of errors. Section 4 presents results from tournaments between the various programs. Section 5 summarizes our observations and conclusions from this research.

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. In situations where no case matches the new problem exactly, the system must adapt one of the retrieved solutions to the new problem. Finally, the system may add a new case to its case base and possibly 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 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.
CYRUS [Kolodner, 1983] and PROTOS [Bareiss & Porter, 1987] are two systems that can adjust their retrieval mechanisms. CYRUS does not receive feedback from a problem-solving environment. Instead, it tries to maintain an appropriate organization of cases based upon a 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. Using this type of feedback, PROTOS is also able to prune its case-base by merging cases.

EACH [Salzburg, 1988] also stores exemplars (generalized, representative instances) to perform classification. When a new instance is presented to EACH, its distance to each exemplar is measured, and 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] is a system that primarily adjusts its adaptation mechanism, although it also has the ability to change 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. It also indexes the repaired case according to the failures that the case avoids.

Adding a case, adjusting the retrieval mechanism, and adjusting the adaptation mechanism are all ways of coping with failure. None of the systems discussed above can do all three. In particular, none can adjust both the retrieval and adaptation mechanisms. Managing both of these abilities presents special problems for a system, because 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. CYRUS adopts an alternate approach: optimize 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 limited feedback from the environment. The following sections present a system that addresses this question.

3 CABOT and Its Opponents

The CABOT program was developed to investigate the problem of using limited feedback from a problem-solving environment to distinguish between different types of errors in case-based reasoning. CABOT has been tested on the game of Othello, although nothing in its architecture is specifically designed for Othello. Othello was selected because of its large search space and the availability of immediate feedback on problem-solving performance.

The feedback available to CABOT is qualitative. After each move, an Oracle program tells CABOT which of the available moves was actually best. It does not explain why the move was

---

4The Oracle makes its 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 uses a larger and more comprehensive set of features than is used by CABOT. The Oracle shifts to exhaustive search near the end of the game.
best, nor does it score or order the rest of the available moves. CABOT's task is to use this limited feedback to improve its game-playing ability.

When CABOT's move differs from the move chosen by the Oracle, its decision is considered a failure that it must correct. CABOT first checks its retrieval function and, if possible, adjusts it. If the retrieval function cannot or should not be changed, CABOT tries to adjust its adaptation function. Changes to the retrieval and adaptation functions are both subject to an assured consistency condition that only allows changes if the two functions remain consistent with each other. CABOT checks for assured consistency by testing any changes on a random sample of game states that it has previously encountered; if a change to one of the functions decreases accuracy, the change is discarded. If neither function can be changed, CABOT adds a new case.

The following section describes the game of OTHELLO, and how it is represented by the various game-playing programs. Section 3.2 describes in detail CABOT's retrieval and adaptation functions, and how CABOT adjusts them. Section 3.3 describes one of CABOT's case-based opponents, while Section 3.4 describes one of its inductive learning opponents.

### 3.1 Othello and its Representation

**Othello** is a two-player game played on an $8 \times 8$ board. The players alternate placing pieces on the board until all of the squares are occupied by pieces, or until neither player can place a piece. Although OTHELLO is a simple game to learn, it is difficult to master; the space of legal boards is approximately $10^{50}$. It has often been used as a domain by researchers in Artificial Intelligence [Rosenbloom, 1982; Lee & Mahajan, 1988; De Jong & Schultz, 1988].

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 numerical function that measures some important aspect 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.⁵ Hereafter, "board" will be used to denote this feature vector, rather than the actual raw board configuration.

CABOT and its case-based opponents all define a *case* to be a pair of boards $R = (R_p, R_c)$, with $R_p$ designated the *parent* and $R_c$ the *child*. A case $R$ represents the Oracle's judgement that the child board is the best result that can be obtained by making a legal move on the parent board.

### 3.2 Move Selection and Error Correction

CABOT plays OTHELLO one move at a time. Its cases consist only of a pair linking a board with the best successor board. The object of its case-based reasoning is solely to determine the best next move. This approach is similar to that taken by traditional game-playing programs which search for the best next move, then take that move and throw away the information gained from search. CABOT does no planning of future moves, and its "search" is strictly limited to the immediate descendants of the current board.

The next three sections discuss how CABOT retrieves cases, adapts cases to determine the best next move, and corrects errors.

---

⁵See [Mitchell, 1984] for a concise definition and thorough discussion of OTHELLO features.
3.2.1 The retrieval step

Given a game board (called the game parent \( G_p \)), the retrieval process identifies the case \( R = (R_p, R_c) \), such that \( R_p \) is closest to \( G_p \), according to a weighted Euclidean distance. During error correction, the weight vector is adjusted by a standard preference predicate training rule, as discussed in Section 3.4.1.

The result of the retrieval step is a single best case \( R \). Note that no similarity threshold is used; the retrieval function always returns a case.

3.2.2 The adaptation step

Recall that \( R_p \) is the board that most closely matches \( G_p \), and \( R_c \) is its child. Since the cases in the case base were provided by an expert, \( R_c \) is the recommended move for the board \( R_p \). The next step is to adapt the retrieved case to the current situation, by determining which of the legal moves (the game children of \( G_p \)) is most like the retrieved child \( R_c \). Figure 1 depicts the relationships between game states involved in retrieval and adaptation.

What is important in a case is not the boards themselves, but the relationship between the parent and the child board. Therefore, the adaptation matching is done between feature differences between the parent and the child. Let \( \Delta R_c \) be the delta vector \( R_p - R_c \). This delta vector represents the change in feature values that should ideally occur between the parent \( G_p \) and child \( G_c \). For every child \( G_{ci} \) of the current game board, we can compute a delta vector \( \Delta G_{ci} = G_p - G_{ci} \). The adaptation function then uses a preference predicate, separate from the one used for retrieval, to select the most preferred delta vector.

The adaptation process identifies the game child \( G_{ci} \) that is closest to the retrieved child \( R_c \), according to the following weighted distance function:

\[
W \times \text{sign}(\Delta R_c - \Delta G_{ci}) \times (\Delta R_c - \Delta G_{ci})^2
\]

During error correction, the weight vector is adjusted with a standard preference predicate training rule, as discussed in Section 3.4.1.

The result of the adaptation step is a single best child, which becomes the game child chosen \( (G_{ce}) \). This determines the move made by CABOT.
3.2.3 The error correction step

After CABOT chooses and makes its move, it asks the Oracle for the best move in the same situation. Call the child corresponding to this move the game child desired \( G_{cd} \). If CABOT’s choice agrees with the Oracle’s choice, no error correction is performed.

If CABOT’s choice does not agree with the Oracle’s choice, then CABOT considers its choice to be wrong and tries to correct itself. As mentioned earlier, CABOT will only make a change if it can assure that its retrieval and adaptation functions will remain consistent with each other.

CABOT goes through three steps in making a correction:

1. CABOT determines whether it could fix its retrieval function. It attempts this first because the retrieval function is the more critical of the two distance functions: since the retrieval function determines the cases that will be used in adaptation, changing it may radically affect the performance of the adaptation process as well. CABOT looks first for cases that, if retrieved, would have been properly adapted; the nearest of these to the game parent \( G_p \) is called a near miss case \( N \). Specifically, the near miss case must satisfy two criteria:

   (a) Its delta vector must match the desired child delta vector \( G_p - G_{cd} \) more closely than that any of the other game children.
   
   (b) Its parent must be closer to \( G_p \) than that of any other case (besides \( R_p \)) satisfying criterion 1.

The first criterion ensures that if the near miss case had been retrieved, its adaptation would have resulted in \( G_{cd} \) being the chosen move. The second criterion ensures that the smallest possible retrieval change will be considered.

CABOT attempts to make this change to the retrieval function, and makes its consistency check. If it cannot maintain accuracy, it discards the change.

2. If CABOT could not change its retrieval function, or if no near-miss case was available, it tries to change its adaptation function. The goal of the adaptation change is for the proper game-child \( G_{cd} \) to be chosen if the current state ever arises again. This is done by training the adaptation function linear threshold unit (LTU) [Nilsson, 1965] so that the weighted distance between \( \Delta R_c \) and \( \Delta G_{cd} \) is less than the weighted distance between \( \Delta R_c \) and every other child \( \Delta G_c \).

   If CABOT cannot make such a change to the adaptation LTU without compromising accuracy, it discards the change.

3. If neither the retrieval nor the adaptation functions could be changed, CABOT adds a new case. The new case consists of the game parent \( G_p \) linked with the game child desired \( G_{cd} \).

The ordering of the first two steps — attempting to fix retrieval before adaptation — is based on retrieval being the more critical of the two processes. This ordering is designed to make the retrieval LTU converge more quickly than the adaptation LTU, and in the implementation this does happen.

CABOT plays an entire game, regardless of its performance during the game. CABOT keeps many statistics regarding the accuracy of its decisions, the number of cases added, and the performance of its retrieval and adaptation functions.
3.3 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 version, 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 game parent. 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 constrasting 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; ties are 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. Its only choice is to add a new case. LATCBR does not add a new case if it selects the same move that the Oracle selects.

3.4 PP: A Pure Inductive Learning Opponent

The PP component of CABOT is a simple inductive learning approach to playing Othello that uses qualitative feedback from an Oracle. PP uses no cases, but chooses its moves using a preference predicate [Utgoff & Saxena, 1987] based on the features. A preference predicate is used for the evaluation function because it can be learned from qualitative information. It needs only the information that the evaluation of one state should be greater than another, rather than the exact evaluations of the two states.

A linear threshold unit (LTU) [Nilsson, 1965] is used to implement the preference predicate for PP. When choosing the best next move during game-playing, PP first enumerates all successor states of the current state. The preference predicate learned by the LTU may not be consistent; therefore, a simple counting process determines the most preferred successor state, and this state determines the move chosen by PP. The Oracle's best move is then used to generate preference pairs on which the LTU is trained.

The next section explains preference predicates in more detail, and how they are implemented using an LTU.

3.4.1 Preference predicates

A preference predicate \( p \) is a generalization of an evaluation function, such that \( p(S_1, S_2) \) is true iff state \( S_1 \) is better than \( S_2 \) for the performance system. The Oracle provides this qualitative information. After each move, PP asks the Oracle for its preferred move, and if it differs from PP's move, PP generates preference pairs from it.

Preference pairs are tuples \((S_1, S_2)\) such that state \( S_1 \) is better for the player than \( S_2 \). If the
Oracle recommended a move resulting in $S_w$, then for every alternate move resulting in a state $S_a$ a preference pair $(S_w, S_a)$ is generated. From these pairs, the preference predicate $p$ is learned as follows. Given features $f_1, f_2, \ldots, f_n$, then from the preference pair $(S_w, S_a)$ the feature vectors

$$F_w = \langle f_1(S_w), f_2(S_w), \ldots, f_n(S_w) \rangle$$
$$F_a = \langle f_1(S_a), f_2(S_a), \ldots, f_n(S_a) \rangle$$

may be generated. These vectors are subtracted giving the feature delta vector:

$$F_D = \langle f_1(S_w) - f_1(S_a), f_2(S_w) - f_2(S_a), \ldots, f_n(S_w) - f_n(S_a) \rangle$$

This delta vector $F_D$ is then a positive example for $p$, and its negation $-F_D$ is a negative example for $p$. Both examples are used to train the LTU, which is then used by PP to choose its move at each turn.

The justification for using the delta vector is as follows\(^6\). Given that $S_w$ is preferred to $S_a$, a linear evaluation function would satisfy the inequality $W \cdot F_w > W \cdot F_a$, where $W$ is the set of weights on the features. This equation can be rewritten as $W \cdot (F_w - F_a) > 0$, and

$$W \cdot F_D > 0$$
$$W \cdot -F_D < 0$$

The LTU is used to find the weight vector $W$ that satisfies these equations.

4 Experiments and Results

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

- **Effectiveness**: How effective was it at playing the game of OTHELLO. Effectiveness was measured by the win/loss ratio, and by the average discs taken per game.

- **Decision Accuracy**: How often it agreed with the Oracle’s choice.

- **Growth of Case Base**: For the case-based systems, how quickly the case-base grows, and whether or not growth appears to level off at some point.

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 go first. Each player was permitted to learn from both its moves and its opponent’s moves throughout the entire tournament. By training on both its own moves and its opponent’s 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

\(^6\)A more detailed discussion may be found in [Utgoff, Saxena, Callan & Fawcett, 1989].
Table 1: Games won and lost by each player during the four tournaments.

<table>
<thead>
<tr>
<th>Opponents</th>
<th>1st player</th>
<th>2nd player</th>
<th>tied</th>
</tr>
</thead>
<tbody>
<tr>
<td>CABOT vs PP:</td>
<td>58</td>
<td>39</td>
<td>3</td>
</tr>
<tr>
<td>CABOT vs LATCBR:</td>
<td>14</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>CABOT vs Oracle:</td>
<td>1</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>PP vs LATCBR:</td>
<td>66</td>
<td>32</td>
<td>2</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Opponents</th>
<th>1st player</th>
<th>2nd player</th>
</tr>
</thead>
<tbody>
<tr>
<td>CABOT vs PP:</td>
<td>34.1</td>
<td>26.6</td>
</tr>
<tr>
<td>CABOT vs LATCBR:</td>
<td>33.1</td>
<td>29.2</td>
</tr>
<tr>
<td>CABOT vs Oracle:</td>
<td>4.9</td>
<td>55.7</td>
</tr>
<tr>
<td>PP vs LATCBR:</td>
<td>38.1</td>
<td>25.9</td>
</tr>
</tbody>
</table>

case-based or instance-based game-playing program is permitted to observe an opponent following a fixed strategy for a sufficiently long time, 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 the following tables and graphs.

Table 1 shows the number of games won, lost and tied by each player during the four tournaments. It demonstrates that CABOT’s performance is superior to the performance of the pure case-based system in playing OTHELLO. It also shows that the performance of the preference predicate is superior to the performance of the pure CBR system. This result is unexpected, because the training data are not linearly separable. We expected the preference predicate 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 typically rated both on how often they win and by the magnitude of the score, so the average number of discs won by each player was also recorded. Table 2 reports the results for each tournament. These figures confirm that the hybrid system is better at selecting moves than both the pure case-based and the pure inductive learning systems.

Figure 2 confirms that CABOT’s performance is better than LATCBR’s performance because its decisions tend to be more accurate relative to the Oracle’s best moves. 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, normally within the first game or two. In contrast, the pure CBR system is initially error-prone, but its accuracy continues to improve as its case base grows. Figure 2 shows that the difference between the two accuracies decreases as the number of games increases. This confirms the intuition that at some point having enough perfect cases is almost as good as having
a smaller number of prototypical 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 3 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 fairly consistent. CABOT is superior both to LATCBR and PP, and PP is superior to LATCBR.

We have also run tournaments between CABOT and opponents using other inductive learning techniques to guide move selection. One of these opponents used the ID3 algorithm for building decision trees [Quinlan, 1986]. One problem in using ID3 for this task is that the Othello boards are described by numeric features, while ID3 is designed 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 finally our own manual method. None were both computationally feasible and more effective than the preference predicate.

5 Conclusion

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. One of the central assumptions in its development was that feedback from the environment is crucial to making the necessary adjustments. The CABOT environment was designed to provide qualitative
feedback to each move selection strategy; that is, it identified a better move if one was available. However, it did not provide any information 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 the retrieval and adaptation mechanisms leads to both a smaller case base and better performance. They suggest that traditional CBR systems might 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. However, it is necessary to ensure that an adjustment will not cause the retrieval and adaptation mechanisms to become inconsistent. CABOT tests this empirically by sampling randomly from a set of problems and ensuring that problem-solving accuracy is not adversely affected. Finally, the performance of CABOT when played against pure generalization and pure case-based learning opponents illustrates that a hybrid architecture can overcome the weaknesses of its components.

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 on an early draft were especially valuable. We thank Jeff Clouse for providing the Oracle program, and Bernard Silver for his assistance.

References

Michigan: Morgan Kaufmann.


