From: AAAI-90 Proceedings. Copyright ©1990, AAAI (www.aaai.org). All rights reserved.

Explaining Temporal Differences to Create Useful Concepts for Evaluating States

Richard C. Yee Sharad Saxena Paul E. Utgoff Andrew G. Barto
Department of Computer and Information Science
University of Massachusetts, Amherst, MA 01003
yee@cs.umass.edu, saxena@cs.umass.edu
utgoff@cs.umass.edu, barto@cs.umass.edu

Abstract

We describe a technique for improving problem-solving performance by creating concepts that allow problem states to be evaluated through an efficient recognition process. A *temporal-difference* (TD) method is used to bootstrap a collection of useful concepts by backing up evaluations from recognized states to their predecessors. This procedure is combined with *explanation-based generalization* (EBG) and *goal regression* to use knowledge of the problem domain to help generalize the new concept definitions. This maintains the efficiency of using the concepts and accelerates the learning process in comparison to knowledge-free approaches. Also, because the learned definitions may describe negative conditions, it becomes possible to use EBG to explain why some instance is *not* an example of a concept. The learning technique has been elaborated for minimax game-playing and tested on a Tic-Tac-Toe system, T2. Given only concepts defining the end-game states and constrained to a two-ply search bound, experiments show that T2 learns concepts for achieving near-perfect play. T2's total searching time, including concept recognition, is within acceptable performance limits while perfect play without the concepts requires searches taking well over 100 times longer than T2's.

1 Introduction

The use of concepts holds the potential for improving both the speed and accuracy of a problem-solving agent. Concepts define sets over the space in which input problem instances are represented. They represent classes of inputs that are significant with regard to achieving the goals of the agent. Concepts are only useful, however, if their definitions support a recognition process that is faster than other means available to the agent for computing the same information. For example, if an agent can determine that property P is true of a problem instance as quickly through search as through concept recognition, then there is no benefit in forming and storing a concept for recognizing P. On the other hand, using concepts that are efficient sources of valuable information allows an agent to respond more quickly to problems or to spend extra time computing higher-quality responses.

This paper focuses on a method for enabling an agent to identify and define concepts that improve its performance in a task. The method is implemented in a system called T2, which operates within the domain of minimax game-playing. T2's learning process is closely related to the one used by Samuel's checkers-playing program, which recursively improved its ability to evaluate board positions based on a bounded-depth search of its current position [Samuel, 1959]. The general approach of learning through recursively caching state evaluations was also demonstrated in the pole-balancing system of Barto, Sutton and Anderson [1983]. Recent related work may be found in the methods of *temporal differences* (TD) discussed by Sutton [1988] and in the learning architecture of Sutton's Dyna system [1990]. Also, Barto, Sutton and Watkins [1990a; 1990b] discuss TD methods from the perspective of the general framework of *dynamic programming* as suggested by Werbos [1977].

One of the mechanisms Samuel used for caching board payoffs was a rote memory that contained complete descriptions of individual boards and their payoff values. In contrast, T2 takes advantage of its knowledge of the problem domain to generalize the descriptions of states before they are cached. The generalizations are achieved through *explanation-based generalization* (EBG) [Mitchell *et al.*, 1986] followed by a slightly modified form of *goal regression* [Waldinger, 1976]. Consequently, T2's memory contains generalized specifications of states rather than descriptions of individual boards.

The efficiency of using concepts learned through

This material is based upon work supported by the National Science Foundation under Grants IRI-8619107 and ECS-8912623, by the Air Force Office of Scientific Research, Bolling AFB, under Grant AFOSR-89-0526 and by the Office of Naval Research through the University Research Initiative program, contract N00014-86-K-0764.

explanation-based approaches has been studied by a number of researchers. Minton [1988] demonstrated that a system employing EBG to learn search control knowledge may become ineffective if no attempt is made to identify and eliminate low utility concepts whose expected benefits are outweighed by their average matching costs. Tambe and Newell [1988] showed that the Soar learning architecture [Laird *et al.*, 1986] is susceptible to forming such low utility concepts, so-called "expensive chunks". In such cases, overall performance after learning may be worse than in the initial system state. Tambe and Rosenbloom [1989] propose restricting the expressiveness of the language used to define concepts as a way of addressing the expensive chunks problem. In a similar spirit, concepts in T2 possess restrictive definitions that are fast to evaluate. Concept recognition in T2 tends to be "perceptual" rather than analytic. The motivation behind the T2 learning mechanism has been specifically to maintain the efficiency of new concept definitions while recursively improving the significance of the concepts being formed.

2 Task Performance and Learning

Given a game state in which it must move, T2 chooses its response by performing a bounded-depth minimax search to try to identify the best child state. Ultimately, the value of any state depends upon a *payoff function* that maps states into real values in the interval $[-1.0, +1.0]$ where $+1.0$ is identified with wins for T2, -1.0 with losses, and 0.0 with draws. In T2, the backed-up minimax payoff is discounted by a uniform cost assigned to moves. This is done so that states with non-zero payoffs can be distinguished on the basis of the minimum number of steps to a win. We chose an arbitrary positive discount factor of 0.9.

Since T2's search is never deeper than a predetermined number of levels, many states at interior nodes of the complete game tree will appear as leaves of T2's truncated search tree. To approximate the complete search therefore requires a payoff function that can evaluate states representing intermediate stages of play. However, T2 starts out with a payoff function only capable of accurately evaluating completed games: wins, losses and draws. All intermediate states evaluate to 0.0, the same value as a draw. Thus, the learning problem in T2 is as follows:

> Given a payoff function sufficient for evaluating the leaves of a complete game tree, develop an enhanced payoff function that can also provide useful evaluations of states at intermediate stages of play.

The usefulness of the payoff function's evaluations is determined by how well the truncated search identifies perfect lines of play. The rationale behind this approach is that using the enhanced payoff function should be a much more tractable computation than determining payoffs via deeper search. Hence, a primary constraint on the learning process is that the application of the new payoff function be efficient.

3 Caching Boards in Concepts: An Overview of T2

T2 uses a collection of concepts to associate states with payoff values. Each concept is uniquely associated with a discounted payoff value found in the minimax game tree, and each has a definition that, ultimately, covers all the states corresponding to its payoff. To ensure efficiency in concept recognition, definitions are not allowed to specify properties of a state requiring information derived from combinatoric searches. In particular, definitions may only refer to directly observable features of the given state representation, which we call the structural state description.

Identifying concepts with payoff values has limitations. A more general approach would view the mapping from input instances to concepts as an intermediate or supplemental step in the process of mapping from inputs to evaluations. For example, the mapping from inputs to concepts could be composed with a nontrivial mapping from concepts to payoff values. Such an extension of the current approach could be useful for problems with a large number of payoff values.

A general picture of the concept learning process in T2 consists of first detecting that an unknown state belongs to a particular payoff group and then caching that result in the definition of the proper concept. Rather than simply caching the entire state as an instance of the payoff group, generalizing the state before caching it provides significant advantages both in the efficiency of the new concept definition and in the speed of the learning process because a set of states is learned in a single step. The new concept then provides a basis for further concept learning. Hence, the entire collection of concepts is built up in a bootstrapping fashion.

3.1 A Simple Test Domain: Tic-Tac-Toe

Our concept learning method was developed from considering the game of Tic-Tac-Toe. One advantage of this domain is that its simplicity allows attention to be focused on the problem of integrating EBG into the process of learning efficient definitions for concepts that enable perfect play. Also, the method has been designed with particular regard to the class of minimax problems, and therefore it is potentially applicable to more domains amenable to minimax search.

Initially, T2 is given exactly three concepts for recognizing any legal Tic-Tac-Toe board b: $null(b)$, $win(X, b)$ and $win(O, b)$; assuming T2 plays X, these represent the payoffs: 0.0, $+1.0$ and -1.0, respectively. The null concept covers every board not covered by any other concept in the collection. Hence, it is always correct for the boards that do not possess a guaranteed winning combination for either player. The concepts

for O need not be given explicitly since they can be obtained by using the X concepts on inverted boards.

Clearly, performing a complete minimax search grounded in T2's initial collection of concepts is sufficient for producing perfect play. However, given T2's performance constraints—a search depth bounded at two-ply—this initial collection yields playing behavior only slightly better than random. All boards leading to forced wins or losses occurring beyond T2's search horizon are simply recognized as null-payoff boards.

To learn concept definitions that correct this situation, one must first be able to identify a misclassified board and its proper payoff value. Misclassifications are identified by temporal differences: the difference between the predicted value of a board and the value backed up either from search or from actual state transitions experienced during problem-solving. It is expected that the backed-up payoff values are more accurate than those derived directly from the payoff function. This will eventually become true even if the backed-up values are themselves ultimately derived from the same payoff function because the backed-up values are based on more accurate information concerning the future consequences of potential actions.

To play a game using minimax search, the process of using concepts to determine payoff values need only occur for boards at the leaves of the search tree. For concept learning, however, concepts are also used to determine payoffs for boards at interior nodes of the tree, including at the root. As in Samuel's system, for a given board b, the payoff determined from memory is a prediction of what b's backed-up search payoff will be. A violated prediction indicates that the current collection of concepts is inadequate. Thus, concept learning in T2 is triggered in the following situation:

> If board b's membership in concept C yields a payoff that is different from the backed up minimax payoff, then C has a definition that is overly general. Its definition needs to be restricted to exclude (at least) the board b.

In T2's initial collection of concepts, the given definitions for $win(X, b)$ and $win(O, b)$ are correct, but the null definition, which covers every board, overgeneralizes in covering boards with intermediate payoff values. For example, a board b leading to a win in one step will be recognized only as a null board, but it will be found through minimax to have a payoff of +0.9. Therefore, it is necessary to prevent $null(b)$ from recognizing b. Since $null(b)$ covers all boards, it is "pre-empted" by creating a definition for a non-null concept that will cover b. The proper non-null concept to create or modify is the one representing b's backed-up search payoff, +0.9. This concept will be created if it does not already exist in the current collection; otherwise, its definition will be modified to cover b, thereby excluding b from $null(b)$.

A useful perspective is that all of the non-null concepts define groups of exceptions to the null concept.

Input: A board b at a node of a minimax search tree.

Output: The collection of concepts that has been modified as appropriate to better predict b's true payoff in the complete minimax game tree.

Method:
1. Compute b's payoff from a concept C, where $b \in C$.
2. Compute b's backed-up search payoff, p.
3. If *Payoff-of* $(C) \neq p$ then
   (a) Identify the relevant children of b, $\{b_i\}$, and their corresponding concepts $\{D_i\}$.
   (b) Form a generalization of b based on C, $\{b_i\}$ and $\{D_i\}$.
   (c) Use the generalization to restrict C.

Table 1: An overview of the concept learning algorithm

This view characterizes the learning process in T2: only overly general concept predictions are detected and subsequently corrected through the learning of exceptions. Under-generalizations of non-null concepts are only detected when such failings lead to incorrect predictions of the null payoff. The learning of exceptions to the null concept translates into the learning of positive examples for the non-null concepts. This learning process produces concepts possessing a logical structure similar to Vere's *multilevel counterfactuals* [1980]. One consequence of this counterfactual structure is that it becomes possible, in certain cases, to use EBG to explain "why not", i.e., to explain why an instance is not an example of a concept.

3.2 Generalizing Boards Before Caching

The algorithm for learning concepts is summarized in Table 1. This section gives a brief description of the steps involved in generalizing a board b for which a payoff prediction from a concept C has been invalidated—steps (3.a–c) of the algorithm. Further details are provided in the sections indicated below, and a summary of the algorithm for step (3.b) is given in Table 2.

To generalize the structure of board b, more information is needed than simply the board and its backed-up payoff. Because the children of b determine its payoff, information about their structures is needed as well. It is necessary to determine which of b's children participated in the prediction violation and which concepts gave rise to *their* payoffs. Determining which children are relevant for correcting the parent's misclassification depends upon whether the backed-up payoff was better or worse than predicted. When the backed-up payoff is better than predicted for the parent, it is because at least one child had a better payoff than predicted for the children. If there is more than one such child, one is selected arbitrarily. When the backed-up search payoff is worse for the parent, it is because *all* of the children have worse payoffs than predicted for them. These two cases yield the relevant children, $\{b_i\}$, indicated in step (3.a). For each such child, it may be

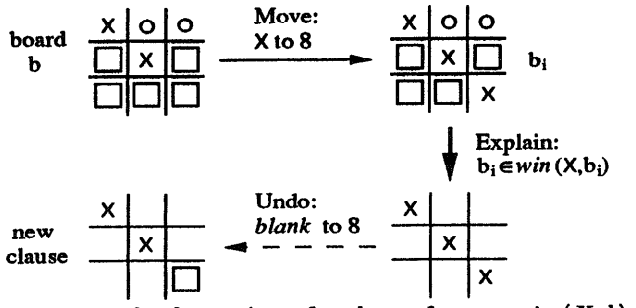

Figure 1: The formation of a clause for *pre-win* (X, b)

necessary to know the structure on which its own payoff is based. This structure can be extracted from the definition of the concept D_i used to assign the child its payoff.

In step (3.b), the key to generalizing the parent b is to use EBG appropriately to generalize the set of relevant children $\{b_i\}$. Given a child b_i, we rely on EBG to identify the child's relevant features, where relevance is determined with respect to some concept Γ. Typically, EBG is used only when an example is known to be a member of a concept, i.e., when $b_i \in \Gamma$. However, because concept definitions in T2 may explicitly represent exceptions, it also becomes possible to produce useful generalizations of an example with respect to its *non*-membership in a concept: $b_i \notin \Gamma$. Section 5.1 describes the explanation process used in T2, and in particular, points out how the case for explaining non-membership arises. In either case, EBG extracts from a child board b_i a general board structure that is sufficient for explaining its relationship to Γ.

After obtaining the structures of the relevant children, we are in a position to back up this information to generalize the parent. A structure at the parent's level is produced from a child's structure by undoing the move that led to the child. This is accomplished by applying a slightly modified version of goal regression, which is a general technique for recovering the pre-image of a set by passing the set backwards through an operator. In T2, each particular move is an operator. The corresponding backwards operator that we use produces a set that *contains* the pre-image of the child's structure, possibly as a proper subset. This process is further described in Section 5.2.

Undoing the moves in each child's structure yields structures at the parent's level of description. The regressed structures are then conjoined into a single specification that describes a general structure that occurs in the parent b. Finally, this new specification is itself conjoined as an exception within the definition of the concept C that incorrectly predicted the payoff of b—step (3.c) of the algorithm.

An Example

Consider the formation of a clause for the concept representing the payoff of +0.9, indicating that X can achieve a win in one step. Call this concept *pre-win*. Suppose that a board b of the form in Figure 1 is evaluated. Board b does not contain a win, so its payoff is initially 0.0, given by $null(b)$. However, one of its children, b_i, satisfies $win(X, b_i)$ yielding the payoff +1.0. Therefore, the payoff of b should have been +0.9. This identifies a learning situation. For convenience, label the nine board squares from 0 to 8 in order from left-to-right, top-to-bottom. The portion of the definition of win (X, b_i) that is the reason for $b_i \in win$ (X, b_i) is found through EBG to be: $((X \text{ at } 0) \wedge (X \text{ at } 4) \wedge (X \text{ at } 8))$. Undoing the move that led to this child yields b's structural generalization: $((X \text{ at } 0) \wedge (X \text{ at } 4) \wedge (blank \text{ at } 8))$. This structure has been found to be the reason that b has the payoff +0.9. Hence, it is cached in the definition of the concept *pre-win* (X, b).

4 Concept Definitions

To achieve an efficient recognition process, concepts are required to have definitions that eliminate or bound the amount of search that may be done to match an instance with a concept. By expressing definitions solely in terms of the immediately observable features used to represent board structures, concepts are not allowed to refer to functional properties requiring search through the state space. A further restriction is that definitions may not have variables at nested levels because these lead to combinatorial searches for bindings on a given board's structure.

In T2 the representation of boards consists of a list of the nine squares, in which each square takes on one of the three values: X, O or *blank*. The values of specific squares are the only terms used in concept definitions. The only bindings allowed occur at the level of entire boards, i.e., the eight board symmetries provided by rotations and reflections are used in matching boards with concepts. This gives a fixed number of possible bindings across all concept definitions.

Generalized board structures are represented as regular boards in which the squares may also take on the value "*don't care*". Such specifications shall be called *gen-boards*. A gen-board is used as a definition for the set of boards matching the specified structure. Because one cannot determine from a gen-board which player is on-move, it is necessary to record this information explicitly in the concept definitions.

A concept definition also has a disjunctive expression in which each clause specifies a subset of legal Tic-Tac-Toe boards. Each clause is a recursive expression, which is represented as a tree in which the nodes are gen-boards. The significance of the root gen-board of a clause is that its structure was sufficient for explaining why a particular board achieved the payoff value represented by the concept possessing the clause. The significance of any child gen-boards of the root is that they describe structures that occur in exceptions—boards containing the root structure yet not yielding the associated payoff. This gives rise to a recursive logical

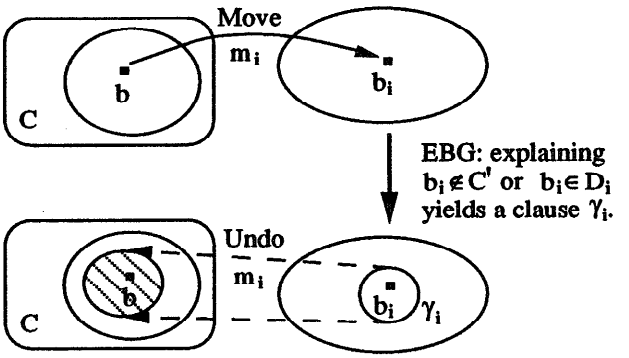

Figure 2: Deriving a clause for an exception

Input: A concept C where $b \in C$, b's relevant children $\{b_i\}$ $(i = 1, \ldots, n)$, their corresponding concepts $\{D_i\}$.

Output: A clause generalizing b.

Method:
1. Let C' be the concept predicted for the children, based on $b \in C$.
2. For each child b_i
   If $\gamma_i \leftarrow$ *Explain* $(b_i \notin C')$ then
   Return: *Undo-Move-Clause* (γ_i)
3. For each child b_i
(a) $\gamma_i \leftarrow$ *Explain* $(b_i \in D_i)$
(b) $\gamma_i \leftarrow$ *Undo-Move-Clause* (γ_i)
4. Return: *Conjoin-Clauses* $(\gamma_1, \gamma_2, \ldots, \gamma_n)$

Table 2: The generalization algorithm: (3.b) of Table 1

structure in clauses: there can be exceptions to exceptions, to any level of nesting. Hence, a clause is either a gen-board or a gen-board conjoined with a conjunction of negated clauses. For a given board to satisfy a clause, it must match the root gen-board while failing to match each child clause. Henceforth, one entire clause (tree) of a concept definition will be called a *concept-clause* when it is necessary to distinguish it from child clauses.

Recall that non-null concepts can be considered as exceptions to the null concept. This indicates that all of the concept-clauses for the non-null concepts can be treated as children of the null gen-board. Thus, the entire set of concepts forms a single tree with the null gen-board at the root and all of the concept-clauses at the first-level of children. The given concept $win(X, b)$ (hence $win(O, b)$) is defined by three concept-clauses each of which is a single gen-board. There is a gen-board for an edge row: $((X \text{ at } 0) \wedge (X \text{ at } 1) \wedge (X \text{ at } 2))$, and, similarly, one for a middle row and one for a diagonal row. Using the board symmetries, these are sufficient for recognizing any win.

5 Learning New Clauses

This section describes how the generalization of a board, b, is derived using EBG and a modified version of goal regression. Figure 2 illustrates the process in the case of a single relevant child. The total information required for creating a new clause is: the concept C that incorrectly predicted b's payoff, the children $\{b_i\}$ that produced the prediction violation for b, and each such child's corresponding concept, D_i.

5.1 Explaining "Why" and "Why Not"

The membership of board b in the concept C, predicts that the best payoff among all of b's children will be given by the concept C', where the payoff of C' is the payoff of C divided by the discount factor. Therefore, in order to generalize b as an exception to C, we would like to know why the relevant children were not in C'. In this case, EBG is sometimes able to identify features of a child b_i that are relevant for non-membership in C'. This type of explanation is attempted first because, when it succeeds, it appears to yield a more precise generalization of the parent b than does explaining $b_i \in D_i$. Explaining a child's membership in its own concept, D_i, provides a reliable back-up strategy. Whichever approach is used, the resulting explanation is a clause specifying a set of boards that includes b_i.

T2's back-up strategy employs the standard EBG approach to explain why $b_i \in D_i$. The explanation is a clause that is a portion of D_i's definition that matched b_i and was sufficient for concluding concept membership. Specifically, it is a concept-clause of D_i that is satisfied by b_i. The concept-clause is a generalized structural specification that covers b_i along with a group of boards sharing the relevant structures and, therefore, sharing the payoff represented by D_i.

Explaining non-membership in a concept is possible because the concept-clauses may recursively specify alternating levels of positive and negative conditions. A non-trivial generalization of a non-member example can be obtained by explaining the example's membership in one of the child clauses of a concept-clause. Suppose we wish to explain $b_i \notin C'$. Clearly, one possible reason could be that b_i does not match the root gen-board of any of C''s concept-clauses, but this is a trivial explanation since the default assumption for all boards is that they are null. Such an explanation cannot be used to improve the system's overall knowledge. The potentially interesting case occurs when b_i satisfies the root gen-board of a concept-clause for C' yet also satisfies at least one of the root's child clauses which specify exceptions. In this case, there is *prima facie* evidence that b_i belongs in the concept, yet it is an exception. The explanation of b_i's non-membership in C' is a matching child clause of a concept-clause whose root also matches b_i.

5.2 Undoing Moves

In T2 each move m_i is an operator yielding b_i from b. We wish to regress each generalized child clause back through its respective move operator to recover a

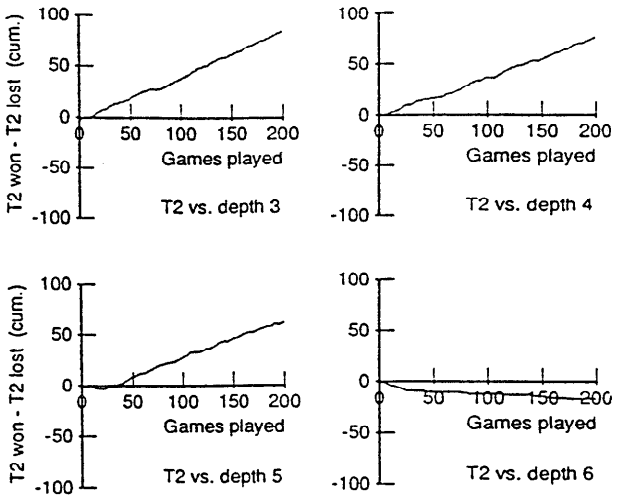

Figure 3: The success of T2 against standard minimax opponents of varying depths

clause for a pre-image containing b. The conjunction of the regressed clauses specify the components of b that make it an exception to the overly general concept C.

Each move consists of a player's symbol and a board square. Undoing a move in a clause is accomplished by recursively traversing the clause's tree structure, and undoing the move in the gen-board at each node. Undoing a move in a gen-board is illustrated by the following example. Suppose that in gen-board G the move "X to square 3" is to be undone. If 3 contains an X replace it with a *blank*. If 3 is a *don't care*, then we use the heuristic of returning G unchanged. Strictly speaking, in such a case G should be returned with the added specification "*blank* at 3". Using the heuristic rule is an attempt to produce a useful generalization of b. In many cases, specifying the additional blank squares introduces unnecessary constraints that can be expected to increase the number of concept-clauses and to slow significantly the speed of learning. Therefore, we are willing to tolerate the possibility that G may be slightly overly general since the learning mechanism can produce corrections if necessary. While we do not yet have a proof that perfect concepts will eventually be formed, experiments demonstrate that the concepts come to support near-perfect play.

After undoing the moves in the clauses, they are conjoined into a single clause by conjoining their root gen-boards into a single gen-board. All child clauses become children of this new root—step (4) in Table 2.

6 Experiments

To evaluate our approach, T2 was played against opponents using standard minimax searches of different fixed depths. T2 always performed a two-ply search. The only concepts known to the opponents were: $null(b)$, $win(X, b)$ and $win(O, b)$. In both T2 and the opponents, if more than one child of a board returned the best payoff value, then a child was randomly selected from among these children. The goal of these experiments was to determine: (a) whether T2 can improve its performance sufficiently to match or surpass any opponent, and (b) whether T2 always uses an acceptable amount of resources, especially time.

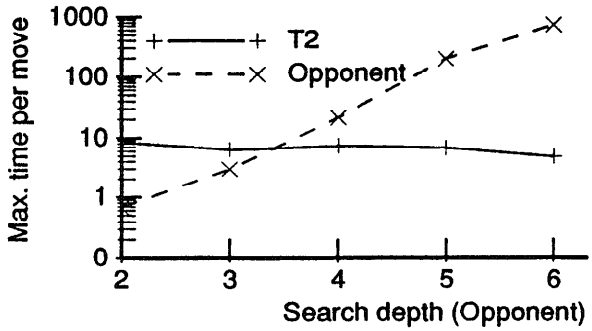

Figure 4: The maximum time used by the players to make a move. Time is shown on a logarithmic scale.

Figure 3 shows the performance of T2 against four opponents. Their searches are bounded from depths three to six; a six-ply search is sufficient for perfect play in Tic-Tac-Toe. The abscissa is the number of games played, and the ordinate is the cumulative difference between the number of games won by T2 and the number it lost. The performance of T2 may be judged by the average slope of the line, e.g., a positive slope indicates a dominant winning trend. The graphs indicate that the learned concepts enable T2 to win 35–40% of the time against all opponents that search to a depth of five or less. Against the depth-six opponent the slope of the line showing T2's performance approaches zero indicating that T2 is approaching the perfect play of the opponent.

Figure 4 uses a logarithmic scale, to show the maximum time (in seconds) used for making a move by T2 and each successive opponent. The time reported is for compiled functions written in Common-Lisp and run on a SUN 3/60. Each value is the maximum for any move occurring in ten additional games that were played after the 200 shown in Figure 3. Figure 4 shows that the time required by T2 to make a move is nearly constant regardless of the opponent. As one would expect, the opponents' times show exponential growth with increasing search depth.

Figure 3 shows that T2 performs as well or better than its opponents while Figure 4 shows that T2 is achieving its results faster than any opponent using a search of depth four or more. In particular, T2 can approach the level of perfect play. Using the learned concepts for this level of play is well over 100 times faster than using standard search alone.

7 Conclusions

We have described a technique for combining a *temporal-difference* learning method with *explanation-based generalization* and a slightly modified form of *goal regression* to enable an agent to learn concepts that improve its problem-solving performance. The learned concepts are generalized memories of problem-solving experiences, and they can be used to evaluate quickly similar problem states encountered in the future. The information for forming concepts can be derived from either local search employing a model (planning) or from direct environmental feedback. We have been interested in situations in which the agent is able to integrate concept learning with actual task performance. Consequently, neither the learning process nor the subsequent process of recalling information can be allowed to interfere seriously with meeting the time constraints of performance.

The T2 system implements the technique in the domain of minimax game-playing and has been tested on Tic-Tac-Toe. The value of the approach can be understood by following Minton's analysis of the benefits versus the costs of using learned concepts. The TD process of backing up board evaluations ensures that application of the concepts will yield significant benefits, which are measured in terms of the depth of the searches necessary for computing the same payoff information. The generalizations and use of board symmetries help ensure that each definition covers a relatively large number of instances resulting in wide applicability of the concepts. Also, there is a corresponding increase in the speed of learning. In considering the costs of the concepts, it is seen that the generalizations also help ensure that there will be a small number of gen-boards to match in the definitions. Restricting the expressiveness of the definitions strictly controls the binding problem during matching as Tambe and Rosenbloom have also demonstrated. Finally, the logical structure of the concepts probably also contributes to the efficiency of recognition. Since concepts are recognized by applying a level-by-level series of general positive and negative tests, it may be expected that, for most concepts, most boards will either pass or fail early in the process. Moreover, in certain cases, this logical structure allows us to use EBG to explain why a given instance is not a member of some concept.

Exploiting domain knowledge within the context of problem solving using local search appears to be an effective method of learning. Our approach efficiently defines concepts whose significance lies in the fact that they distinguish only those regions of the input representation space that are relevant to to the goals of the agent. This addresses fundamental computational issues in producing goal-directed behavior, and we expect that further research will produce more general formulations of these principles.

Acknowledgements

We are grateful to Steven Bradtke, Carla Brodley, Jamie Callan, Margie Connell, and Tom Fawcett for many valuable comments on a draft of this paper.

References

[Barto *et al.*, 1983] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13:835–846, 1983.

[Barto *et al.*, 1990a] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Learning and Sequential Decision Making. In M. Gabriel and J. W. Moore, (eds.), *Learning and Computational Neuroscience*, MIT Press, Cambridge, MA, Forthcoming.

[Barto *et al.*, 1990b] A. G. Barto, R. S. Sutton, and C. J. C. H. Watkins. Sequential decision problems and neural networks. In D. S. Touretzky, (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann, San Mateo, CA, Forthcoming.

[Laird *et al.*, 1986] J. E. Laird, P. S. Rosenbloom, and A. Newell. Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1:11–46, 1986.

[Minton, 1988] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 564–569, Morgan Kaufmann, San Mateo, CA, 1988.

[Mitchell *et al.*, 1986] T. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1:47–80, 1986.

[Samuel, 1959] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, July 1959.

[Sutton, 1988] R. S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44, 1988.

[Sutton, 1990] R. S. Sutton. Integrated architectures for learning, planning and reacting based on approximating dynamic programming. Submitted to the 1990 International Machine Learning Conference, 1990.

[Tambe and Newell, 1988] M. Tambe and A. Newell. Some chunks are expensive. In *Proceedings of the Fifth Conference on Machine Learning*, Morgan Kaufmann, San Mateo, CA, 1988.

[Tambe and Rosenbloom, 1989] M. Tambe and P. Rosenbloom. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 731–737, Morgan Kaufmann, San Mateo, CA, 1989.

[Vere, 1980] S. A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:138–164, 1980.

[Waldinger, 1976] R. Waldinger. Achieving several goals simultaneously. In E. W. Elcock and D. Michie, (eds.), *Machine Intelligence*, Wiley and Sons, New York, 1976.

[Werbos, 1977] P. J. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22:25–38, 1977.