Adaptive Pattern-Oriented Chess

Robert Levinson and **Richard Snyder***
Department of Computer and Information Sciences
University of California Santa Cruz
Santa Cruz, CA 95064
levinson@cis.ucsc.edu and snyder@cis.ucsc.edu

Abstract

Psychological evidence indicates that human chess players base their assessments of chess positions on structural/perceptual patterns learned through experience. *Morph* is a computer chess program that has been developed to be more consistent with the cognitive models. The learning mechanism used by Morph combines weight-updating, genetic, explanation-based and temporal-difference learning to create, delete, generalize and evaluate chess positions. An associative pattern retrieval system organizes the database for efficient processing.

The main objectives of the project are to demonstrate capacity of the system to learn, to deepen our understanding of the interaction of knowledge and search, and to build bridges in this area between AI and cognitive science.

To strengthen connections with the cognitive literature limitations have been place on the system, such as restrictions to 1-ply search, to little domain knowledge, and to no supervised training.

Introduction to the Problem and Its Solution

> Although it is apparently effective to discover tactical issues by searches, isn't it dull to "forget" them immediately after use instead of "learning" something for a later re-use? There is no serious attempt known to make a chess program really learn from its experience for future games itself. (Kaindl, 1989)

Despite the recognition of the criticality of search and the high-costs that are paid to achieve it, little effort has been applied to getting chess systems to utilize previous search experiences in future searches. Thus, excluding random factors from the system (or human intervention), one can expect a chess system to play exactly the same way against the same series of moves, whether it has won or lost, and take the same amount of time to do so! There do now exist some systems that recall positions that they found promising, but from which they later lost material (Scherzer *et al.*, 1990; Slate, 1987). This is certainly a step in the right direction, but much more important than dealing with exact replication of positions is to handle situations that are *analogous, but not identical*, to previous situations.

The main characteristic of the current model of chess programming is the use of brute-force alpha-beta minimax search with selective extensions for special situations such as forcing variations. This has been further enhanced by special purpose hardware. This model has been so successful that little else has been tried.

Alternative AI approaches to chess have not fared well due to the expense in applying the "knowledge" that had been supplied to the system. Those times in recent years that chess has been applied as a testbed (Flann and Dietterich, 1989; Quinlan, 1983; Michalski and Negri, 1977; Niblett and Shapiro, 1981; O'Rorke, 1981; Shapiro, 1987; Tadepalli, 1989; Pitrat, 1976; Minton, 1984) only a small sub-domain of the game was used.

However, we feel that there is a third approach that neither relies on search or the symbolic computation approach of knowledge-oriented AI: this we shall call the "pattern-oriented approach." In this approach configurations of interaction between squares and pieces are stored along with their significance. A uniform (and hence efficient) method is used to combine the significances in a given position to reach a final evaluation for that position.

Morph[1] is a system developed over the past 3 years that implements the pattern oriented approach (Levinson, 1989b; Levinson, 1989a). It is not conceivable that the detailed knowledge required to evaluate positions in this way could be supplied directly to the system, thus learning is required. A learning mechanism has been developed that combines weight-updating, genetic, and temporal-difference learning modules to create, delete, generalize and evaluate graph patterns. An associative pattern retrieval system organizes the database for efficient processing.

To strengthen the connections with the cognitive literature the system's knowledge is to come from its own

*Both authors supported in part by NSF Grant IRI-8921291.

[1]The name "Morph" comes from the Greek *morph* meaning form and the chess great, Paul Morphy.

playing experience, no sets of pre-classified examples are given and beyond its chess pattern representation scheme little chess knowledge such as the fact that having pieces is valuable (leave alone their values) has been provided to the system. Further, the system is limited to using only 1-ply of search.[2]

System Design

Morph makes a move by generating all legal successors of the current position, evaluating each position using the current pattern database and choosing the position that is considered least favorable to the opponent. The system is designed so that after each game patterns are created, deleted and generalized and weights are changed to make its evaluations more accurate based on the outcome of the game.

Patterns and their Representation

The basic unit of knowledge to be stored is a pattern. Patterns may represent an entire position or represent a boolean feature that has occurred in one or more positions and is expected to occur again in the future. Along with each pattern is stored a weight in [0,1] that is an estimate of the expected true minimax evaluation of states that satisfy the pattern. In Morph, patterns come in two different types: *graph patterns* and *material patterns*. Material patterns and graph patterns are processed identically by the system (in separate databases) except that the matching and generalization operations are simpler for material patterns.

The graph patterns in Morph represent positions as unique directed graphs in which both nodes and edges are labeled. Nodes are created for all pieces that occur in a position and for all squares that are immediately adjacent to the kings. The nodes are labeled with the type and color of piece (or square) they represent and for kings and pawns (and pieces of *arity* 0) the exact rank and file on the board in which they occur. Edges represent attacks and defends relationships: Direct attack, indirect attack, or discovered attack. Graphs are always oriented with white to move. Patterns come from subgraphs of position graphs (see Figures 1a and 1b) and may generalize rank and file square designations. Many chess positions that are on face value dissimilar, have a large subgraph (see Figure 1) in common in this representation.

A similar representation scheme has successfully been used to represent chess generalizations (Zobrist and Carlson, 1973) and to produce a similarity metric for chess positions(Levinson, 1989b; Botvinnik, 1984).

Morph's material patterns are vectors that give the relative material difference between the players, e.g. "up 2 pawns and down 1 rook," "even material," etc. The material patterns provide useful subgoals or signposts that are used to enhance the temporal-difference learning process since they can occur at any point in a game sequence. Morph tends to learn proper values for these early on in its training (see Table 1), thus providing a sound basis for learning the significance of graph patterns.

[2]Though nothing in the method except perhaps efficiency, prevents deeper search.

Associative Pattern Database

The pattern database expedites associative retrieval by storing the patterns in a partially-ordered hierarchy based on the relation "subgraph-of" ("more-general-than") (Levinson, 1991). Only the immediate predecessors and immediate successors of each pattern are stored in the partial ordering. At one end of the hierarchy are simple and general patterns such as "white bishop can take black pawn" and at the other end are complex and specific patterns such as those associated with full positions.

Empirically, it has been shown that on typical databases only a small fraction of the patterns in the database need be considered to answer a given query. For example, on a database of 600 patterns rarely are more than 10 to 20 comparisons required (Levinson, 1991).

Evaluation Function

The evaluation function takes a position and returns a value in [0,1] that represents an estimate of the expected outcome of the game from that position (0=loss, .5=draw, 1.0=win). The value returned by the evaluation function is to be a function of the patterns that are immediate predecessors of the state in the database and their weights. The contribution of other predecessors is in principle reflected in the weights of the immediate predecessors since they are most-specific.

Determining the exact method by which the pattern values should be combined is a difficult and critical problem and is a major issue being explored. Some progress has been made:

The "extremeness" of pattern probabilities induces the prioritization structure. Specifically, let $w_1, \ldots, w_n$ be the weights of patterns (material or graph) that apply to a position, P, and $ex(w)$ be the "extremeness" of that weight. Clearly we want to "listen" to the more extreme patterns because they are an indication that something important is happening. To be more formal, the extremeness value, $ex(w)$, of a pattern w is defined as follows: $ex(w) \equiv |w - 0.5|$. The evaluation function for P is:

$$eval(P) = \frac{\sum_{i=1}^{n}(w_i \cdot ex(w_i)^\beta)}{\sum_{i=1}^{n}(ex(w_i)^\beta)}$$

If $\sum_{i=1}^{n} ex(w_i)^\beta = 0$, $eval(P) = 0.5$. β is a tunable parameter that indicates how much we want our evaluation function to be skewed by extreme patterns.

Learning System

Positional Credit Assignment and Weight-Updating Each game provides feedback to the system about the accuracy of its evaluations. The first step is to use the outcome of the game to improve the evaluations assigned to positions during the game. The method used by the system to assign new evaluations to states is temporal-difference (TD) learning (Sutton,

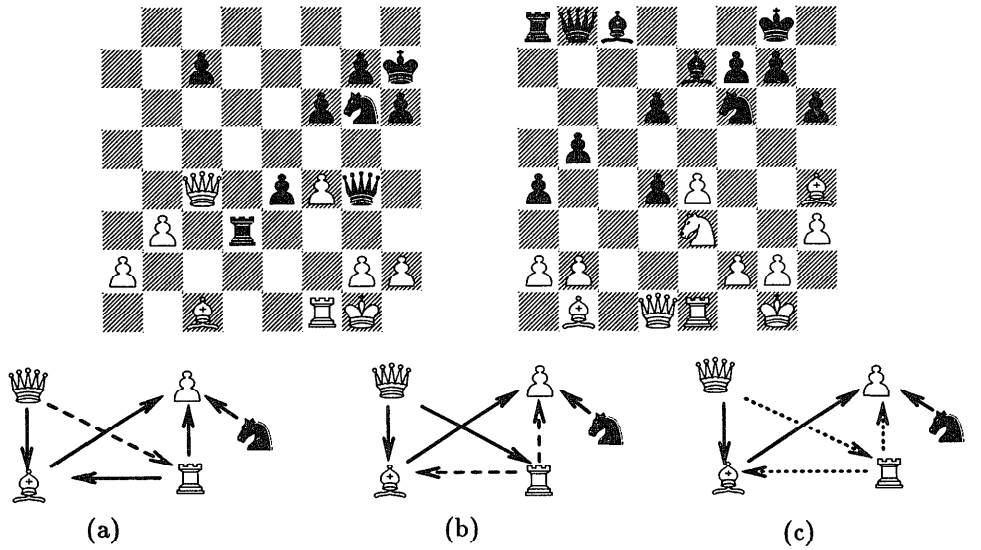

Figure 1: A generalization derived from two different chess position. (a) is the subgraph derived from the board on the left and (b) is the subgraph from the board on the right. The solid edges correspond to direct edges between pieces and the dashed edges correspond to indirect edges. Graph (c) is the generalized graph derived from (a) and (b) in which the dotted edges have been generalized to generic "attacks."

1988). In TD learning, the idea is to update weights so that one position's evaluation corresponds more closely to the evaluation of the following position (as opposed to the evaluation of the final or goal state). Temporal difference learning makes the feedback loop more local.

Specifically, the system assigns the final position its true value (0, 0.5, or 1.0) and then iteratively assigns each preceding position P (working backwards) the value $(1 - \alpha)$(current value of P) + $\alpha(1 -$ new value of succeeding position). α (now 1/2) is a parameter in [0,1] that can be adjusted to reflect how much importance should be given to new versus old information.

Once new positions have been given evaluations, the weights of patterns that were used to evaluate the positions are moved in the direction of the desired or "target" evaluation.

Pattern Creation The system must have a method for developing and storing new patterns. During weight updating, for each pair of successive positions s_i and s_{i+1} the system constructs two new patterns P_{before} and P_{after} through a form of explanation-based learning (EBG) (Mitchell *et al.*, 1986). These new patterns are to represent the changes between two positions that occur with a single move and hence must contribute to the difference in their evaluations. A domain-dependent rule adds to the patterns a context in which these changes may occur. With graph patterns, P_{before} is the smallest connected subgraph made up of those edges in s_i that are not in s_{i+1} and P_{after} is the smallest connected subgraph made up of those edges in s_{i+1} that are not in s_i. P_{before} and P_{after} are then augmented (for context) by adding to them all edges adjacent to their initial set of nodes.

Pattern Deletion As in genetic algorithms (Goldberg, 1989), there must be a mechanism for insignificant, incorrect or redundant patterns to be deleted (forgotten) by the system. A pattern should contribute to making the evaluations of positions it is part of more accurate. The utility of a pattern can be measured as a function of many factors including age, number of updates, uses, size, extremeness and variance. Using a utility function (Minton, 1984), patterns below a certain level of utility can be deleted.

Performance Results

Morph's main trainer and opponent is GnuChess Level One, which is rated at least 1600 (better than 63% of tournament players). For a 1-ply program, starting from scratch and beating such an opponent, would demonstrate a very significant amount of learning. To date Morph has been unable to defeat GnuChess even once, though it has obtained several draws when GnuChess has inadvertently stalemated it. The main difficulty seems to be that due to insufficient training and/or imprecision in the weight-updating and evaluation mechanisms, removing all "bugs" from Morph's database has not been possible. In chess, one bad move can ruin an otherwise reasonable game, especially against a computer. Morph's main trouble is "overgeneralization" attributing too much significance to small features of the board. To achieve an even record against GnuChess is a primary objective of the current project. The longest game has been 48 moves (for each player) with an average game length after 50 games of training of 26 moves.

Despite the lack of success against GnuChess there have been many encouraging signs in the three months since Morph was fully implemented:

- After 30 or more games of training Morph's material patterns are consistent and credible (see Table 1 for a database after 106 games), even though no information about the relative values of the pieces or that pieces are valuable have been given the system. The weights correspond very well to the traditional values assigned to those patterns (except the queen is undervalued). These results reconfirm Korf and Christensen's efforts (Christensen and Korf, 1986) and perhaps go beyond by providing a finer grain size for material.
- After 50 games of training, Morph begins to play reasonable sequences of opening moves and even the beginnings of book variations despite that no information about development, center control and king safety have been directly given the system and that neither Morph or GnuChess uses an opening book.
- Morph's database contains many patterns that are recognizable by human players and has given most reasonable values. The patterns include mating patterns, mates-in-one, castled king and related defenses and attacks on this position, pawn structures in the center, doubled rooks, developed knights, attacked and/or defended pieces and more.
- Morph's games against GnuChess improve with training. Since Morph has not yet been victorious, we use another performance metric besides win-loss record: the total amount of opponent's material captured by Morph, using the traditional piece values (queen=9, rook=5, bishop=3, knight=3, pawn=1). Of course, Morph is "unaware" of this metric. Figure 2 gives a typical plot of Morph's performance over time (with β=4.5). We have discovered that adding patterns too frequently, "overloads" Morph so it can't learn proper weights. In the experiment graphed in Figure 3, weights were updated after every game but patterns were added only every 7 games. This allowed the system to display a steady learning curve as depicted.

Relationship to Other Approaches

Clearly, the chess system combines threads of a variety of machine-learning techniques that have been successful in other settings. It is this combination and exactly what is done to achieve it that is the basis for Morph's contributions. The learning-method areas and their involvement in Morph include genetic algorithms (Goldberg, 1989; Grefenstette, 1987; Holland, 1987), neural nets (weight updating) (Rumelhart and McClelland, 1986), temporal-difference learning, explanation-based generalization (EBG), and similarity-based learning. To combine these methods some design constraints usually associated with these methods are relaxed. With genetic algorithms, structured patterns rather than bit strings are used. In contrast to neural networks the nodes in Morph's hierarchy are assigned particular semantic/structural values. Temporal-difference learning is usually applied to methods with fixed evaluation function forms (in which the features are known but not their weights) but here the features change and the hierarchical database organization produce atypical discontinuities in the function.

We feel the integration of these diverse techniques would not be possible without the uniform, syntactic processing provided by the pattern-weight formulation of search knowledge. To appreciate this, it is useful to understand the similarities and differences between Morph and other systems for learning control or problem-solving knowledge. For example, consider Minton's explanation-based Prodigy system (Minton, 1984). The use of explanation-based learning is one similarity: Morph's pattern creation specifically create patterns that are "responsible" for future favorable or unfavorable patterns being created. Also similar is the use of "utility" by Morph's deletion routine to determine if it is worthwhile to continue to store a pattern, basing the decision on accuracy and significance of the pattern versus matching or retrieval costs. A major difference between the two approaches is the simplicity and uniformity of Morph's control structure: no "meta-level control" rules are constructed or used nor are goals or subgoals explicitly reasoned about. Another difference is that actions are never explicitly mentioned in the system. Yee et al.(Yee *et al.*, 1990) have combined explanation-based learning and temporal-difference learning in a manner similar to Morph and other APS systems, applying the technique to Tic-Tac-Toe.

Now let's compare Morph to other adaptive-game playing systems. In earlier systems, the system is given a set of features and asked to determine the weights that go with them. These weights are usually learned through some form of TD learning (Tesauro and Sejnowski, 1989). Morph extends the TD approaches by exploring and selecting from a very large set of possible features in a manner similar to genetic algorithms. Lee and Mahajan (Lee and Mahajan, 1988) also improve on these approaches by using Bayesian learning to determine inter-feature correlation.

A modicum of AI and machine learning techniques in addition to heuristic search have been applied directly to chess. The inductive-learning endgame systems (Michie and Bratko, 1987; Muggleton, 1988) have relied on pre-classified sets of examples or examples that could be classified by a complete game-tree search from the given position (Thompson and Roycroft, 1983). The symbolic learning work by Flann (Flann and Dietterich, 1989) has occurred on only a very small subdomain of chess. The concepts capable of being learned by this system are graphs of two or three nodes in Morph. Such concepts are learned naturally by Morph's generalization mechanism.

Tadepalli's work (Tadepalli, 1989) on hierarchical goal structures for chess is promising. We suspect that such high-level strategic understanding may be necessary in the long run to bring Morph beyond an intermediate level (the goal of the current project) to an expert or master level. Minton (Minton, 1984), building on Pitrat's work (Pitrat, 1976), applied constraint-based generalization to learning forced mating plans. This method can be viewed as a special case of our

Material Pattern					Statistics				Trad.
Pawn	Knight	Bishop	Rook	Queen	Weight	Updates	Variance	Age	Value
0	0	0	0	0	0.455	2485	240.2	106	0
0	0	−1	0	0	0.094	556	7.53	86	−3
0	0	+1	0	0	0.912	653	11.19	88	+3
0	+1	0	0	0	0.910	679	23.59	101	+3
0	−1	0	0	0	0.102	588	17.96	101	−3
0	0	0	−1	0	0.078	667	3.56	103	−5
0	0	0	+1	0	0.916	754	5.74	103	+5
+1	0	0	0	0	0.731	969	22.96	105	+1
−1	0	0	0	0	0.259	861	13.84	105	−1
0	0	0	0	+1	0.903	743	5.68	105	+9
0	0	0	0	−1	0.085	642	3.12	105	−9
0	0	−1	+1	0	0.894	10	0.03	55	+2
0	0	−2	0	0	0.078	146	0.53	71	−6
+1	0	−1	0	0	0.248	26	2.35	73	−2
0	+1	−1	0	0	0.417	81	4.48	82	0
0	−1	+1	0	0	0.478	84	5.72	82	0
0	0	+2	0	0	0.924	168	0.66	91	+6

Table 1: A portion of an actual Morph material database after 106 games.

The columns headed by pieces denote relative quantity. The weight column is the learned weight of the pattern in [0,1]. Updates is the number of times that this weight has been changed. Variance is the sum of the weight changes. Age is how many games this pattern has been in the database. Value is the traditional value assigned to this pattern. Note that a weight of 0.5 corresponds to a traditional value of 0. The entire database contained 575 patterns.

pattern creation system. Perhaps the most successful application of AI to chess was Wilkin's Paradise (PAttern Recognition Applied to DIrecting Search) system (Wilkins, 1980), which, also building on Pitrat's work, used pattern knowledge to guide search in tactical situations. Paradise was able to find combinations as deep as 19-ply. It made liberal use of planning knowledge in the form of a rich set of primitives for reasoning and thus can be characterized as a "semantic approach." This difference plus the use of search to check plans and the restriction to tactical positions distinguish it from Morph. Also, Paradise is not a learning program: patterns and planning knowledge are supplied by the programmer. Epstein's Hoyle system (Epstein, 1990) also applies a semantic approach but to multiple simultaneous game domains.

Conclusions

To our knowledge, this is the first major project to apply recent learning results to computer chess. The early results with the chess system are highly encouraging. We are excited about this project's relevance to some growing debates on fundamental issues in artificial intelligence and cognitive science research.

In addition to Morph's unique combination of methods, what distinguishes it are:

- A uniform representation of search knowledge.
- A syntactic approach to playing and learning.
- An attempt to play a complete game of chess rather than a small subdomain.
- Rejection of a learning-by-examples framework for an experiential framework that is more cognitively-inspired.
- Responsibility for feature discovery given to the system.
- Non-reliance on search (though at some point a small amount of guided search may be incorporated, bringing us even closer to the cognitive model).

References

(Botvinnik, 1984) M. Botvinnik. *Computers in Chess.* Springer-Verlag, 1984.

(Christensen and Korf, 1986) J. Christensen and R. Korf. A unified theory of heuristic evaluation. In *AAAI-86*, 1986.

(Epstein, 1990) S. L. Epstein. Learning plans for competitive domains. In *Proceedings of the Seventh International Conference on Machine Learning*, June 1990.

(Flann and Dietterich, 1989) N. S. Flann and T. G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

(Goldberg, 1989) D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Welsley, Reading, MA, 1989.

(Grefenstette, 1987) J. J. Grefenstette, editor. *Genetic algorithms and their applications*, Hillsdale, NJ, 1987. L. Erlbaum and Associates.

(Holland, 1987) J. H. Holland. Genetic algorithms and classifier systems: Foundations and future directions. In J. J. Grefenstette, editor, *Second International Conference on Genetic Algorithms*, Hillsdale, NJ, 1987. L. Erlbaum and Associates.

(Kaindl, 1989) H. Kaindl. Towards a theory of knowledge. In *Advances in Computer Chess 5*, pages 159–185. Pergammon, 1989.

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

(Levinson, 1989a) R. Levinson. Pattern formation, associative recall and search: A proposal. Technical Report UCSC-CRL-89-22, University of California at Santa Cruz, 1989.

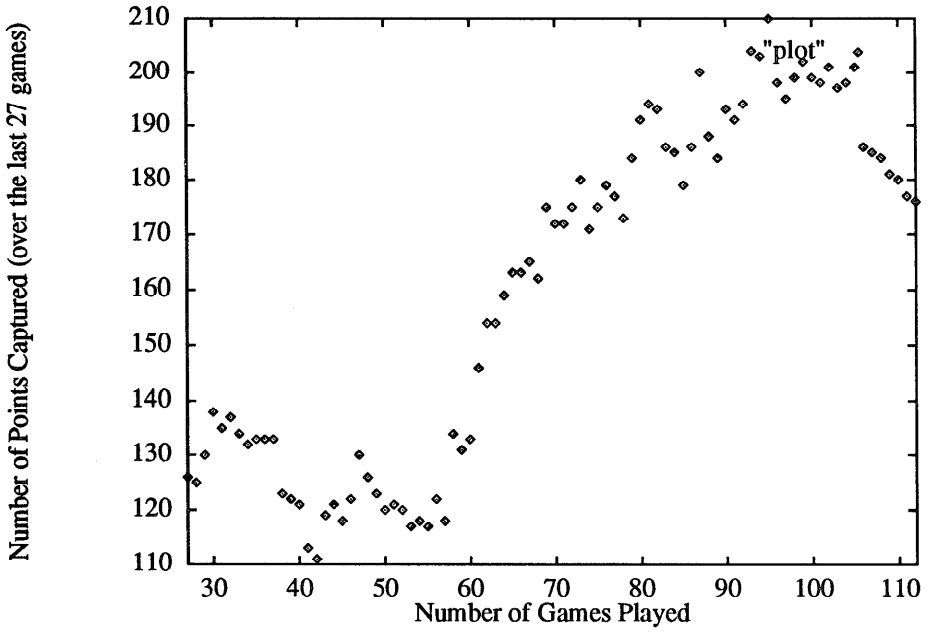

Figure 2: Graph plotting Morph's progress versus GnuChess.

(Levinson, 1989b) R. Levinson. A self-learning, pattern-oriented chess program. *International Computer Chess Association Journal*, 12(4):207–215, December 1989.

(Levinson, 1991) R. Levinson. A self-organizing pattern retrieval system and its applications. *Internation Journal of Intelligent Systems*, 1991. To appear.

(Michalski and Negri, 1977) R. S. Michalski and P. Negri. An experiment on inductive learning in chess end games. In E. W. Elock and D. Michie, editors, *Machine Representation of Knowledge, Machine Intelligence*, volume 8, pages 175–192. Ellis Horwood, 1977.

(Michie and Bratko, 1987) D. Michie and I. Bratko. Ideas on knowledge synthesis stemming from the KBBKN endgame. *Journal of the International Computer Chess Association*, 10(1):3–13, 1987.

(Minton, 1984) S. Minton. Constraint based generalization- learning game playing plans from single examples. In *Proceedings of AAAI*. AAAI, 1984.

(Mitchell *et al.*, 1986) T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation based generalization: A unifying view. In *Machine Learning 1*, pages 47–80. Kluwer Academic Publishers, Boston, 1986.

(Muggleton, 1988) S. H. Muggleton. Inductive acquisition of chess strategies. In D. Michie J. E. Hayes and J. Richards, editors, *Machine Intelligence 11*, pages 375–389. Oxford University Press, Oxford, 1988.

(Niblett and Shapiro, 1981) T Niblett and A. Shapiro. Automatic induction of classification rules for chess endgames. Technical Report MIP-R-129, Machine Intelligence Research Unit, University of Edinburgh, 1981.

(O'Rorke, 1981) P. O'Rorke. A comparative study of inductive learning systems AQ11 and ID3. Technical Report Intelligent Systems Group Report No. 81-14, Department of computer Science, University of Illinois at Urbana-Champaign, 1981.

(Pitrat, 1976) J. Pitrat. A program for learning to play chess. In *Pattern Recognition and Artificial Intelligence*. Academic Press, 1976.

(Quinlan, 1983) J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1983.

(Rumelhart and McClelland, 1986) E. D. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, volume 1–2. MIT Press, 1986.

(Scherzer *et al.*, 1990) T. Scherzer, L. Scherzer, and D. Tjaden. Learning in Bebe. In T. A. Marsland and J. Schaeffer, editors, *Computer, Chess and Cognition*, chapter 12, pages 197–216. Springer-Verlag, 1990.

(Shapiro, 1987) A. D. Shapiro. *Structured Induction in Expert Systems*. Turing Institute Press with Addison-Wesley, 1987.

(Slate, 1987) D. J. Slate. A chess program that uses its transposition table to learn from experience. *Journal of the International Computer Chess Association*, 10(2):59–71, 1987.

(Sutton, 1988) R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

(Tadepalli, 1989) P. Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989. Morgan Kaufmann.

(Tesauro and Sejnowski, 1989) G. Tesauro and T. J. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39:357–390, 1989.

(Thompson and Roycroft, 1983) K. Thompson and A. J. Roycroft. A prophesy fulfilled. *EndGame*, 5(74):217–220, 1983.

(Wilkins, 1980) D. Wilkins. Using patterns and plans in chess. *Artificial Intelligence*, 14(2):165–203, 1980.

(Yee *et al.*, 1990) R. C. Yee, Sharad Saxena, Paul E. Utgoff, and Andrew G. Barto. Explaining temporal differences to create useful concepts for evaluating states. In *Proceedings of the Eighth National Conference on AI*, Menlo Park, 1990. American Association for Artificial Intelligence, AAAI Press/The MIT Press.

(Zobrist and Carlson, 1973) A. L. Zobrist and D. R. Carlson. An advice-taking chess computer. *Scientific American*, 228:92–105, June 1973.