ENCYCLOPEDIA OF COMPUTER SCIENCE AND TECHNOLOGY

EXECUTIVE EDITORS

Allen Kent *James G. Williams*

UNIVERSITY OF PITTSBURGH
PITTSBURGH, PENNSYLVANIA

ADMINISTRATIVE EDITOR

Carolyn M. Hall

ARLINGTON, TEXAS

VOLUME 44
SUPPLEMENT 29

MARCEL DEKKER, INC. NEW YORK • BASEL

COMPUTER CHESS AND INDEXING

OVERVIEW OF COMPUTER-CHESS HISTORY

Computer chess probably began in 1769 with an impressive chess automation called Turk (1) that had been devised and presented by Baron Wolfgang von Kempelen. This automation looked like a large desk with a chessboard and chess pieces on top. Before the beginning of a game, Turk's doors were opened and the audience saw a proliferation of gear wheels inside. Over a few decades, it participated in hundreds of exhibitions and won against most of its opponents, including Napoleon. In May 1827, however, two boys observed a small man coming out of Turk, revealing that there was a chessmaster who made his moves from a secret compartment.

In 1914, an electromechanical device was demonstrated by an engineer named Torres y Quevedo (2). In contrast to Turk, this was a real machine, capable of mating an endgame with a king and rook against a king. This kind of ending is rather simple, because a few rules can explain how to win in such a way. The application was quite a marvel for that period, however.

Thirty years later, John von Neumann and Oskar Morgenstern (3) presented the minmax algorithm and explained how it applied to chess. During the early 1950s, when computers became available to a few people, a significant contribution was made by two landmark articles. In his article, Claude Shannon (4) described the basic ideas concerning how a chess program could be constructed. Alan Turing (5) independently wrote his hand-simulated chess program and one of its games. Both scientists proposed the use of the minmax algorithm and a simple evaluation function.

In 1967, Mac Hack Six, developed by Richard Greenblatt at the Massachusetts Institute of Technology, became the first chess program to play and win in a tournament game. CHESS 3.0, the Northwestern University chess program programmed by David Slate and Larry Atkin, reached the level of a tournament player in 1970. In 1976, CHESS 4.0, programmed by the same team, reached the expert level. BELLE, developed by Ken Thompson and Joe Condon at Bell Labs, was the first chess program to achieve the American Master level. HIGHTECH, developed by Hans Berliner, Carl Ebeling, Murray Campbell, and Gordon Goetsch at Carnegie Mellon University, achieved the American Senior Master level in 1988. DEEP THOUGHT, developed by Thomas Anantharaman, Murray Campbell, Feng Hsu, Andreas Nowatzyk, and Mike Brown at Carnegie Mellon University, reached the level of Grandmaster in 1988. In 1997, an IBM team that included Murray Campbell, Feng Hsu, Jerry Brody, Joe Hoane, Joel Benjamin, and C. J. Tan developed DEEP BLUE, a massively parallel computer running that became the first chess program to beat a world champion in a regulation match. It won against Garry Kasparov in a six-game match by the score of 3.5 to 2.5.

A more comprehensive overview of computer chess in general and its history in

particular can be found in Refs. 6–10. An overview of the theory and practice of computer game playing, along with references on this subject, is found in Ref. 11.

OVERVIEW OF COMPUTER-CHESS CONCEPTIONS

With more people involved in developing chess-playing programs, two philosophical camps have emerged: the emulation camp and the engineering camp. The disagreement between these two camps is whether to emphasize simulating human play or searching.

The first camp claims that computers should simulate humans' playing by modeling their decision-making process. Chess masters decide on their moves in a "knowledge-intensive" mode, applying relatively small amounts of search. Therefore, their strategies of play and the knowledge they use should be simulated.

The second camp claims that computers should use their computational power by using special-purpose hardware and/or applying brute-force search [i.e., extensive tree searching on very large game trees using various Artificial Intelligence (AI) techniques and complex evaluation functions]. As is summarized in Ref. 12: "The further a process can look ahead, the less detailed knowledge it needs." It appears now that the second camp has won. The most high-level playing programs belong to this camp; for example, HITECH (13) and DEEP BLUE (14).

Even the second camp, however, uses knowledge, especially in the following three domains: (1) the opening stage (using theoretical opening books), (2) the middlegame and endgame stages (transposition tables; see the section Overview of Using Chess Precedents), and (3) the endgame stage (endgame databases based on theoretical endgames and endgames constructed by special programs).

Computer-chess scientists have invested more in problem-solving heuristics than in retrieval from past matches, however. In the next subsection, we discuss things that have done in the domain of using knowledge in general and previous cases in particular.

Complexity of Chess

Shannon (4) estimated the number of different legal chess positions to be about 10^{43}. Shannon reached this estimation by the following calculation $64!/[32! \times (8!)^2 \times (2!)^6]$, which is, more exactly, about 4.63×10^{42}. The explanation for this combinatorial calculation is as follows: There are 64 squares on the chessboard, 32 different pieces, 8 white pawns, 8 black pawns, 6 groups of 2 identical pieces (2 rooks, 2 bishops, and 2 knights for both sides), and 4 groups of 1 piece each (king and queen for both sides) which do not imply on the denominator.

This calculation takes into consideration only possible arrangements of the pieces on the board. There are positions that are not legal, however (the two kings are in near squares, both kings are checked, etc.); therefore, chess problemists and mathematicians estimate the number of different legal chess positions to be 10^{40} (15).

In addition, the number of nodes in the chess tree that can be developed during a search is estimated to be at least 10^{100} nodes (12, p. 173). Such a number of positions is impractical to store in memory, thus chess programs store only important positions in three kinds of databases. (See the section Overview of Using Chess Precedents.) In addition, there is, of course, a need to compress stored positions and played moves. These subjects are discussed below.

REPRESENTATION OF THE CHESSBOARD AND THE MOVES

Shannon's Mailbox Method

Shannon (4) proposed that the chessboard be presented by 64 computer words in the computer's memory. Each word (like a mailbox) would contain information about one square of the chessboard. The information can be 1 of 13 possible integer values between −6 and 6: 0 for an empty square, 1 for a white pawn, 2 for a white knight, 3 for a white bishop, 4 for a white rook, 5 for a white queen, 6 for a white king, −1 for a black pawn, −2 for a black knight, and so on; that is, a word needs 4 bits to enable 13 possibilities. The 8×8 chessboard is numbered according to the coordinate system presented in position 1. A representation of a whole position needs 256 bits.

Additional variables are needed in order to know who has to move (White or Black) and what the privileges are concerning castling and en passant captures. A move (except of promotion and castling) can be presented by giving the source square and the target square. For example, a white pawn on 10 that proceeds two squares will be presented as 1,0; 3,0. Such a move can be stored in the memory in 12 bits because each square can be 1 of 64 squares (i.e., 2^6 squares needs 6 bits).

70	71	72	73	74	75	76	77
60	61	62	63	64	65	66	67
50	51	52	53	54	55	56	57
40	41	42	43	44	45	46	47
30	31	32	33	34	35	36	37
20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17
00	01	02	03	04	05	06	07

Position 1

More advanced programs have also used this procedure. Instead of an 8×8 board, however, they used a 10×12 board with a special value (e.g., 7) stored on all squares that are outside the board. In this way, they can easily detect the edges. In all these methods (including Shannon's), legal moves can be quite easily determined, simply by defining the offsets to its present square. For example, a white pawn can proceed +10 or +20 and a knight's address can be changed by one of the following offsets: +8, +19, +21, +12, −8, −19, −21, and −12. After calculating the move, the program must check the new address to see whether or not the move was legal.

A more modern method to represent the chessboard, called a bit-map presentation, was proposed in the late 1960s by several groups independently. The first group to do so was the Russian computer-chess group (16). Instead of using 256 bits for a whole chessboard and assigning 4 bits for each square, they used 12 (64-bit) words for a whole chessboard, assigning 1 bit for each square. The first word represents the places of the

white pawns. The second word represents the places of the black pawns. In this way, 10 other words represent all other kinds of chess pieces (knights, bishops, rooks, queens, and kings) for both sides. Moreover, in this way, we can define many other needed words (e.g., one word for all squares attacked by white pieces or one word for all squares for which a black knight can fork black pieces). Various chess relationships can be presented in these bit maps. Bit maps that are oriented toward relationships between pieces and generation of moves can support the program in working out plans and achieving improved results. An example for a program that makes extensive use of these bit maps is the advice-taking program developed by Zobrist and Carlsson (17). A more comprehensive overview on the representation of the chessboard and the moves can be obtained in *Chess Skills in Man and Machine*, edited by P. W. Frey (8), especially the third chapter.

Overview of Data Compressing in Encoding Chess Knowledge

Nievergelt found that any legal chess position of about 10^{40} can be determined by 136 bits (15). His explanation for this is that a question with n possible answers is about $\log_2 n$ bits. (If we make an accurate calculation, however, we will see that 133 bits suffice, because 2^{133} is about 1.09×10^{40} and 2^{136} is about 8.7×10^{40}.) Nievergelt conducted experiments in which a human was allowed to determine a master-level tournament position by asking multiple-choice (yes/no) questions. The average number of bits required to guess a position was about 70. The question of whether or not the storage of positions by computers can be done with similar efficiency was partially solved by Balkenhol (18), who wrote an algorithm encoding realistic positions by 1-bit answers to a sequence of yes or no questions. On average, his algorithm found that 79 bits are required for such a position to be encoded. Special questions about castling rights, en passant captures, or the player, however, will require five additional bits of encoding.

Compressing chess games is another important issue in storing chess games in databases. One simple method is the short correspondence chess notation method. In this method, every move is represented by the coordinates of its source and target squares. Each move needs 12 bits, as 3 bits (8 possibilities) are needed for each coordinate.

Althofer (19) proposed a method called compression by prediction, which compresses tournament games with the help of a fast deterministic chess program. Althofer's method saved about 75% of the storage needed by the short correspondence chess notation method (i.e., only about 3 bits for a move). The key idea of this method was that the move is encoded relative to its probability to be played by a tournament player.

OVERVIEW OF CHESS PRECEDENTS

Kinds of Chess Precedents

Chess precedents can be categorized into several main categories; for example, chess games, chess positions, and chess patterns (chunks). The kinds of precedents needed for a computer-chess program depend on the tasks the program should take care of (e.g., playing, solving problems, and analyzing).

Each kind of precedent usually requires additional information [e.g., evaluation and/or analysis in different levels, recommended moves to play, expected continuation, and expected results for each size (white or black)].

Chess Patterns and Their Analysis

A psychological study by de Groot (20) shows that chess players base their evaluations of chess positions on patterns (called also chunks) gained through experience. A pattern/chunk is part of a board position represented by a meaningful grouping of pieces. Each board can be further divided into several chunks. According to psychological evidence, human chess players perform chunking (21).

Two processes take place as a chess player improves his quality: (1) the number of stored chunks increases and (2) part of his chunks become richer and more complex. (They will contain more pieces and more relationships among them.) The player's chunks guide him in deciding which move to play or, rather, which strategy to choose in a given position. Simon (22) estimates that a master has an estimated repertoire of between 25,000 and 100,000 patterns.

Why use patterns in computer chess? Patterns encode important information. They contain various situations with related information (e.g., evaluation and move to play) and using them is considerably more powerful than brute-force search. Without working with patterns, many more additional searches would be needed to get similar results.

KINDS OF CHESS DATABASES

Computer-chess programs use four kinds of databases: opening databases, endgame databases, chunk databases, and transposition tables. Why use databases when we achieve such good results with brute-force search using heuristic evaluation functions? The answers will be presented in the descriptions of the three databases that computer chess programs tend to use.

Endgame Databases

High-level playing in the endgame stage requires huge amounts of detailed knowledge. Part of this knowledge is seldom used. Plans may contain variants of more than 40 plies; therefore, brute-force search will not be enough in the endgame stage.

Ken Thompson has constructed well-known endgame databases (which are available on CD-ROMs) that are widely used. These databases include solutions for all five-man endgames and several six-man endgames. Using his databases, Thompson (23) showed several previously unknown results in the endgame literature, including that a king and a queen win against a king and two bishops in general and that a king and a queen win against a king and two bishops (or two knights) in most positions.

Opening Databases

Brute-search functions using heuristic evaluation functions are relatively weak for choosing good moves in the opening stage. This is because the opening emphasizes optimal development of the pieces for the middlegame stage 20–30 plies later, which are beyond the machine look-ahead horizon. All reasonable computer-chess programs therefore use extensive databases of theoretic variants of different openings taken from a wide variety of opening literature.

Chunk Databases

A library of analyzed chunks in a limited domain (a special kind of king and pawn endgame) has been used successfully in order to evaluate whole positions in a playing program called CHUNKER (24). CHUNKER solved difficult problems using chunks during search.

Flinter and Keane (25) proposed a method for automatically generating a case library of chunks from master games. They took positions from 350 games of an ex-world champion, Mikhail Tal, and created a library with 4533 base chunks after reducing chunks with a too low a frequency (too rare) and a too high frequency (too primitive). Further work is needed to refine the generation of the chunks, however.

Transposition Tables

Transposition tables (26,27) are large direct-access tables that store positions that have already been evaluated in previous searches. They are used to prevent recalculations of the same positions and different kinds of symmetric positions, such as white and black symmetric, vertical symmetric, horizontal symmetric, and diagonal symmetric. The transposition tables save not only the results of positions previously evaluated but also the moves that achieved these results and the depth of the subtrees that have been searched.

OVERVIEW OF USING CHESS PRECEDENTS

Using Hash Tables

In practice, the memory required for all the positions in chess exceeds the available random-access memory of computers; therefore, transposition tables are usually implemented as hash tables (28). The most common kind of hash table used by chess programs was defined by Zobrist (29).

A hash table is one with a number that is sufficiently large, called the hash value. Using special hash functions, a given position is converted into a hash index (a number between 0 and the hash value). If, for example, the hash table contains 2^n entries, then n low-order bits of the hash value are used as a hash index. The remaining bits, called the key, are used to distinguish among different positions which were mapped into the same hash index. Using a hash table can cause two kinds of errors, which have been identified by Zobrist (29). A short summary of these errors and methods to cope with them can be found in Ref. 30.

A hash table is usually implemented as one table with one entry per position. According to Marsland (31), an entry should at least contain the following components: key, move, score, flag indicating whether the score is a true value or only a bound, and the relative depth in the subtree searched.

The information stored in the transposition table is used for each node in the search tree; that is, the resulting position for each checked move is looked up in the hash table. If the position is present and its search depth is not smaller than the depth still to be searched for the current position, then the information in the hash table is used.

Ebeling proposed a two-level hash table as an improvement (32). Schaeffer (33) implemented this idea with two table positions per entry. The last recently used position is stored in the first-level position and the other position is stored in the second-level position.

Learning Chess Precedents

Perhaps the most basic type of learning by a computer program is storing analyzed positions in the transposition table. Such a program was constructed by Slate (34). He implemented a simple method proposed by Samuel (35). This technique enables a tree-searching program to accumulate selected positions and information related to them during its play. The transfer of information constitutes a kind of learning from experience. A limited capability of generalization from specific positions to classes of similar positions is available. The learned knowledge may be used in further play via the transposition table. The constructors of BEBE (36), a full-playing program, developed a technique for updating its transposition program during play by using concepts of both short-term memory and long-term memory. Their experimental results showed that when playing against a single specific opponent in 100–200 games, an improvement of about 1 level in chess rank is possible.

Learning patterns is considered to be more complicated. Such learning was done in the program Patterns and Learning (PAL), constructed by Morales (37,38). This program plays simple endgames (e.g., a king and a rook against a king) while incrementally learning chess patterns expressed in a subset of first-order Horn clauses.

Grandmaster Nunn (39) used computer-generated endgame databases for analyzing endgames and discovering new and interesting endgames. Nunn describes different ways of using this software as follows: checking and correcting analyzed endgames, analyzing over-the-board endgames, exploring endgames to extend theory, discovering general rules that govern a certain type of ending, and forming key positions that a human player could memorize.

OVERVIEW OF REPRESENTATION OF KNOWLEDGE AND KNOWLEDGE-BASED PROGRAMS

Knowledge-Based Representation and Programs

The importance of the utility of expert knowledge is described in experiments made by Schaeffer and Marsland (40). They show that knowledge cannot be added in arbitrary order, however, because additional knowledge and interactions need to be taken into consideration to solve contradictions and improve evaluating.

Michie's (41) advice language applied AI techniques to solve simple endgames. A pattern-based representation of chess endgame knowledge was proposed by Bratko et al. (42). An elaboration of this representation was given in Bratko and Michie (43).

Pitrat's system (44) is a chess combination game program that analyzed a given middlegame position, generated some plans, and tried to execute them while considering natural branches of the search tree. Wilkins's system (45,46) is also a planning program for middlegame positions. This system is more complicated than Pitrat's system and has more kinds of plans. In addition, it was the first program that used knowledge to control tree searching.

Wilkins's program used a knowledge base containing about 200 production rules (that resemble if–then rules) to find the best move in various chess positions. Each production rule has a pattern (a complex interrelated set of features) as its condition. The action can be one or more concepts that can be used by the program in its planning process.

A more comprehensive model of chess knowledge and reasoning was proposed by

Seidel (47). This model elaborated the knowledge used in Pitrat's and Wilkins's programs. Seidel proposed a "generative grammar," which can be used to analyze positions and create more kinds of plan than those proposed in the programs mentioned.

Most computer-chess programs do not involve using knowledge in the sense of retrieving similar positions and adapting their solution (e.g., evaluation, move to play) to the position at hand. There are some computer-chess programs, however, that involve this kind of using knowledge, which is better known as case-based reasoning.

CASE-BASED REASONING

Case-based reasoning (CBR) means adapting solutions of previous cases to solve new cases. In computers, CBR has been successfully employed in several domains, such as law (48) and medicine (49). Another potentially exciting CBR domain is game playing in general and chess in particular, because human players use extensive knowledge in their playing. Little research has been done in these areas, however. In CBR research on non-chess-playing programs, we find treatment in the games of Eight-puzzle (50) and Othello (51).

HITECH (13) is a pattern-based program that can play a full game at a strong master level. It combines large and fast searches with pattern recognition. HITECH has production rules that define temporary goals for both sides and the patterns need to recognize achievement of these goals during the search. At the start of each search iteration, the program analyzes the position and selects the appropriate patterns. These patterns are compiled into a form that can be used effectively by the program.

A pattern-based evaluation process was proposed by Levinson and Snyder (52). Their system, called Morph, splits the given position into several chosen patterns, which have already been evaluated, and computes the evaluation value of the entire position based on the values of the chosen patterns. Morph is restricted to lower-level tactical or piece position patterns with only a limited ability to abstract from these, however. Morph II (53) has addressed these concerns by abstracting new and wider patterns from the rules of the domain. However, these patterns may not really coincide with the way humans would classify the position and, thus, only have limited explanatory use. Morph and Morph II do not supply any detailed evaluative comments about the given position.

Lazzeri and Heller (54) constructed an intelligent consultant system for chess. Their system ICONCHESS uses CBR and fuzzy logic techniques in order to supply the user with high-level advice, especially for middlegame positions. ICONCHESS analyzes a position, extracting its relevant features, and then evaluates the position and proposes possible strategies for play. Then, it retrieves similar positions by considering syntactical similarities (exact place of pieces) and semantical similarities (strategic goals/plans are similar) to supply additional advice. At the end, ICONCHESS presents advice by combining graphical and textual approaches. The cases that Lazzeri and Heller used are positions taken from games played by experts and masters and their analysis.

Another intelligent educational chess system constructed by Kerner (55) before ICONCHESS will be discussed here is a simple but a detailed example of computer chess and indexing. This system includes a case-based model that supplies a comprehensive positional evaluation for any given position.

The Evaluation Problem

The quality of the player's evaluation and analysis capabilities are the most important factors in determining his strength as a player. These capabilities allow a chess player to work out plans and to decide which specific variations to calculate. Evaluation combines many different factors, each with its own weight, depending on the factor's relative importance (56).

Most game-playing programs do not make the evaluation process explicit, however. Current systems do not supply any detailed evaluative comments about the internal content of the given positions; that is, there is a deficiency of evaluative comments concerning given positions. General evaluative comments concerning given positions have been supplied by several systems (e.g., Refs. 57–60); nevertheless, these systems do not supply any detailed evaluative comments about the internal content of the given positions. Moreover, these systems are not case based. We believe that using CBR can contribute to the task of giving detailed evaluative comments about the internal content of the given positions.

Our goal was the development of an intelligent educational chess system. We believe that using CBR can contribute to the task of giving detailed evaluative comments about the internal content of the given positions. In Ref. 55, we proposed a case-based model that supplies an analysis for any chess position (except for illegal and mate positions). A given position is analyzed by examining the most significant basic features found in the position. The proposed analysis is mainly directed at teaching chess evaluation and planning. This model should be helpful to weak and intermediate players wishing to improve their play.

Evaluation of Chess Positions Using Previously Analyzed Positions

Our model is based on a concept called the basic chess pattern. A basic chess pattern is defined as a certain minimal configuration of a small number of pieces and squares that describes only one salient chess feature. In order to discover as many different basic chess patterns as possible, we—with the help of chess masters—have constructed a hierarchical tree structure that includes most basic positional features concerning the evaluation of chess positions.

This tree is a hierarchical classification of most of the common chess features at different levels of abstraction. At the root of the tree, we have the concept of "static evaluation." Each leaf (a node at the last level) in this tree represents a unique basic pattern (e.g., "one isolated pawn in the endgame stage"). Each basic pattern has two suitable explanation patterns (one for white and one for black) that contain several important comments concerning the discussed pattern.

Most of the concepts were collected from a variety of relevant chess books (61–65). The highest-level concepts of this tree are shown in Figure 1. Figures 2 and 3 illustrate the subtrees describing the pawn and king concepts, respectively. A few important concepts mentioned in these trees are defined in the glossary.

Each leaf that represents a unique basic chess pattern has its own evaluative value. For example, the pattern "two isolated doubled pawns in the endgame stage" has an evaluative value of –0.5. It is important to mention that the evaluative value given to each basic pattern is only a general estimate that represents its value in the majority of the positions that include this pattern.

In addition, when evaluating a chess pattern, we often consider the stage in the

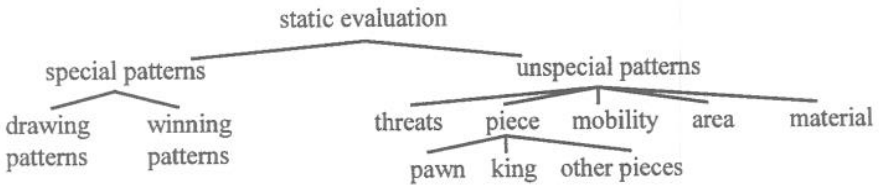

FIGURE 1 The evaluation tree.

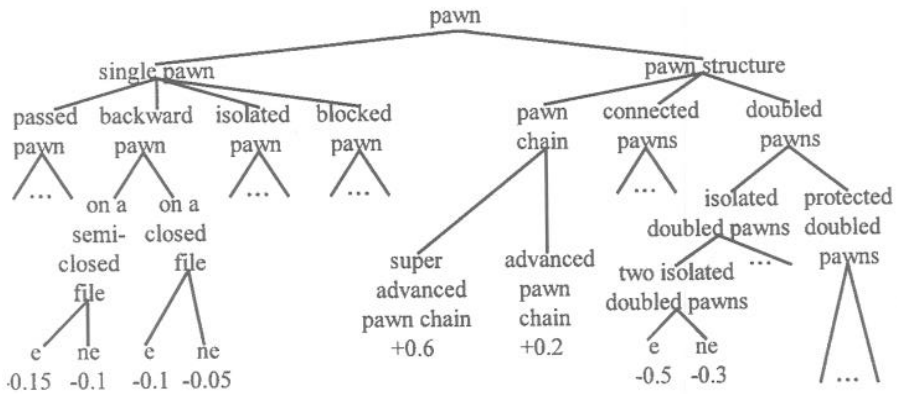

FIGURE 2 The evaluation subtree for the pawn concept.

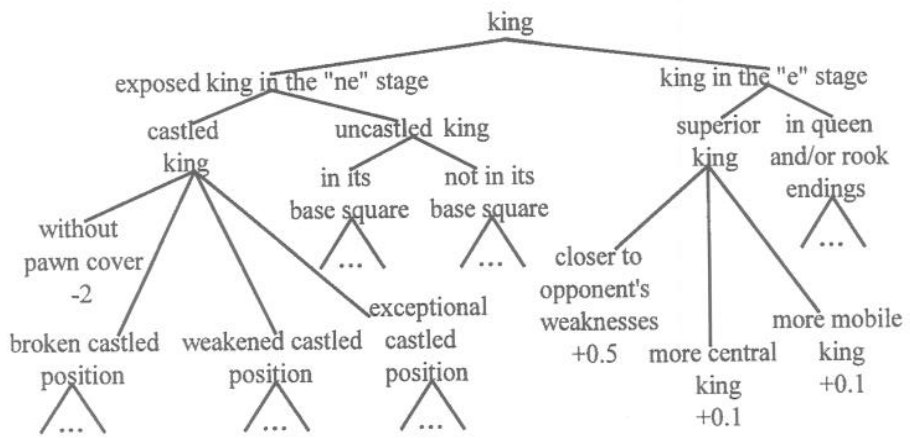

FIGURE 3 The evaluation subtree for the king concept.

game in which the pattern takes place. This distinction is important because the same pattern can be evaluated differently in the e and ne stages (the endgame stage and not the endgame stage, respectively). For example, the pattern "isolated pawn" becomes a greater weakness in the e stage because the importance of a pawn becomes greater.

The tree contains 613 nodes, of which 403 are leaves (i.e., basic patterns). This tree is used primarily for two main tasks: (1) searching the tree to find all basic patterns included in a given position and (2) determining pattern similarity in the adaptation process in our model. The structure of our evaluation tree is similar in some ways to the E-MOP, the memory structure introduced by Kolodner (66). Our chess concepts resemble Kolodner's generalized information items, and our basic chess patterns can be viewed as her "events."

Chess Evaluative Comments

Chess experts have established a set of qualitative evaluation measures. Each chess position can be evaluated by one of these measures. Table 1 presents a few qualitative measures, their equivalent quantitative measures, and their meanings. These measures are based on the common relative values of the queen, rook, bishop, knight, and pawn, which are 9, 5, 3, 3, and 1 points, respectively (4).

Little research has been done concerning the task of giving detailed evaluative comments for game positions. Most game-playing programs do not make the evaluation process explicit, but rather give only one evaluative score. A chess student is not always capable of understanding why the specific chess program he is working with evaluated the position the way it did. In order to increase his evaluating ability effectively, he needs to receive explanatory evaluative comments. There are programs that can supply a more detailed explanation concerning an evaluated position, however. A few such programs are presented below.

A theory of evaluative comments has been proposed by Michie (56). In addition to the construction of the classical minmax game tree, Michie has developed a model of fallible play. His theory assigns to each position two values: "game-theoretic value" and "expected utility." Based on combinations of these values, his theory supplies short

TABLE 1 A Few Qualitative Measures (LMs) in Chess, Their Equivalent Quantitative Measures (NMs), and Their Meanings

Qualitative measures	Quantitative measures	Meaning
l	$3 < NM$	White is winning.
y	$1 < NM <= 3$	White has a big advantage.
r	$0 < NM <= 1$	White has a small advantage.
=	$NM = 0$	The game is even.
t	$-1 =< NM < 0$	Black has a small advantage.
u	$-3 =< NM < -1$	Black has a big advantage.
o	$NM < -3$	Black is winning.

Note: The LMs are intervals of NMs, which are numbers based roughly on positional evaluations.

comments on chess positions (e.g., "Black has a theoretical win but is likely to lose."). His theory does not supply any comments about the internal content of the evaluated positions, however (e.g., the pawn structure). Moreover, this theory does not suggest any plans for the continuation of the game.

Another explanation mechanism has been constructed by Berliner and Ackley (57). Their system, called QBKG, can produce critical analyses of possible moves for a backgammon position using a hierarchical knowledge tree. It gives only two kinds of comments. The first is a general evaluation of the discussed position. The second is an answer to the question "Why did you make that move as opposed to this move?" In addition, Berliner and Ackley admit that their system is only able to produce comments on about 70% of the positions presented to it.

In the last years, a few additional programs that can explain their positions, have appeared [e.g., HOYLE (58) and METAGAMER (59)]. HOYLE and METAGAMER view a feature as an advisor that encapsulates a piece of advice about why some aspect of the position may be favorable or unfavorable to one of the players. Using these advisors, these programs can comment generally on positions.

To sum up, little research has been done concerning case-based detailed evaluations of game positions in general and case-based detailed evaluations of chess positions in particular. Michie, Berliner and Ackley, and Epstein and Pell contribute to the task of giving general evaluative comments concerning game positions. Their models (and to the best of our knowledge other existing models) nevertheless do not supply any detailed evaluative comments concerning the internal content of the given positions. We propose an initial framework for a detailed case-based evaluation model for computer chess programs.

DATA STRUCTURES FOR EVALUATION OF CHESS POSITIONS

XPs

XPs are explanation patterns (67). Kass (68, p. 9) regards Xps as "variablized explanations that are stored in memory, and can be instantiated to explain new cases." The XP, according to Schank (67, p. 39), contains the following slots: (1) a fact to be explained, (2) a belief about the fact, (3) a purpose implied by the fact, (4) a plan to achieve the purpose, and (5) an action to take.

The XP structure has been applied to criminal sentencing (69). We find this structure also appropriate for the domain of evaluating chess positions. Whereas judicial XP describes a specific viewpoint of a judge concerning a sentence, the chess XP describes a specific viewpoint of a chess position from either White's point of view or Black's point of view.

Because we are concerned with the evaluation of the given position in the case of chess, we use an evaluation slot instead of an action slot. The evaluation slot contains two kinds of evaluative values: a quantitative measure (NM) and a qualitative measure (LM) (demonstrated in Table 1).

In our model, each basic pattern in the evaluation tree has two *general XPs* (one for White and one for Black). Six different examples of XPs are included in the multiple explanation patterns (MXPs) presented in Figures 6 and 7. For the sake of convenience, we use some abbreviations: W for White, B for Black, K for king and Q for queen.

Without loss of generality, our examples will be evaluated from White's viewpoint, assuming that it is White's turn to move.

In summary, the XP structure seems to be a convenient data structure for describing and explaining a specific chess pattern of a given position. Each chess position usually includes more than one important pattern, however; thus, the XP structure does not suffice to explain an entire chess position.

Learning XPs

Learned XPs are important because they can direct player's attention to an important analysis that might have been overlooked otherwise. These XPs can improve their understanding, evaluating, and planning abilities. In Ref. 70, describe game-independent strategies capable of learning explanation patterns for evaluation of any basic game pattern. We have developed five game-independent strategies (replacement, specialization, generalization, deletion, and insertion) capable of learning XPs or parts of them. At present, the application is only in the domain of chess. These 5 strategies have been further developed into 21 specific chess strategies.

MXPs

The MXP structure (70) was first introduced to assist judges in deciding which sentence to hand down in a new case. The MXP is a detailed graphical explanation of a given case and its outcome. In general, the MXP is defined as a collection of XPs and an outcome slot. Each XP represents a unique important viewpoint concerning the given case and carries its weight in its evaluation slot to the outcome slot of the entire case. In our model, each important chess viewpoint regarding the given position is explained and evaluated by a suitable XP, and the general evaluation for the entire position is represented in the outcome slot.

In order not to overload the user with too much information, we stipulate that the chess MXP be composed of the three most important XPs (those with the highest absolute evaluation values) suitable to the discussed position. At present, our evaluation function is a summation over the evaluation values of these XPs. We use this simple rough function to enable the user to understand how the system reached its general evaluation. The summation is nevertheless meaningful because each XP included in the MXP describes only one unique independent basic pattern; that is, the quantitative measure of the "general evaluation" slot is a summation over its XPs' quantitative values. Its qualitative measure is dependent on its quantitative measure, as seen in Table 1.

Figures 4 and 5 present two chess positions. Figures 6 and 7 describe the MXPs that analyze these positions, respectively. The most important chess concepts mentioned in these MXPs are defined in the glossary.

Our MXP structure fits the way Steinitz (the first formal world chess champion, between 1886 and 1894) taught players as was written by Kotov (65, p. 24):

> Steinitz taught players most of all to split the position into its elements. Naturally they do not all play the same role in a given position, they do not have the same importance. Once he has worked out the relationship of the elements to each other, the player moves on the process of synthesis which is known in chess as the general assessment.

We believe that the MXP structure provides a better framework for explaining a given position than those given by other systems because we supply comments on the

FIGURE 4 Chess position 1.

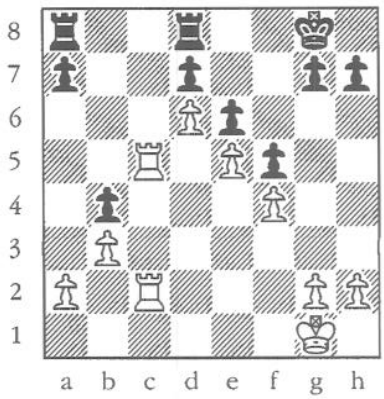

FIGURE 5 Chess position 2.

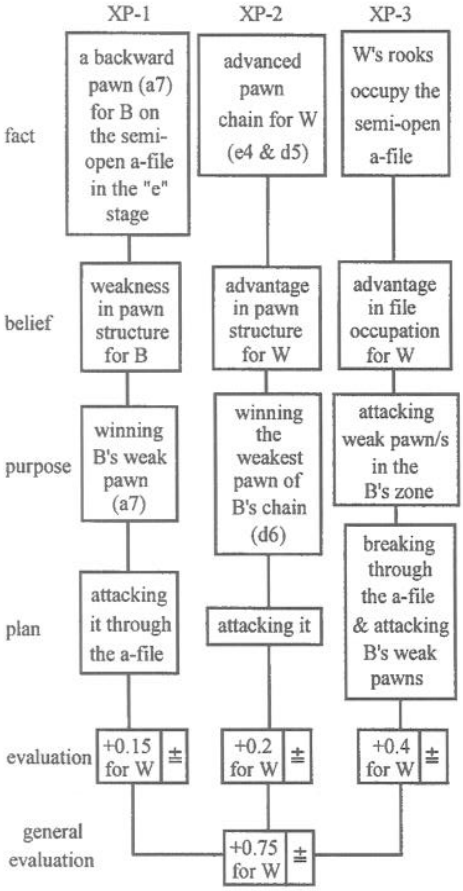

FIGURE 6 MXP for position 1.

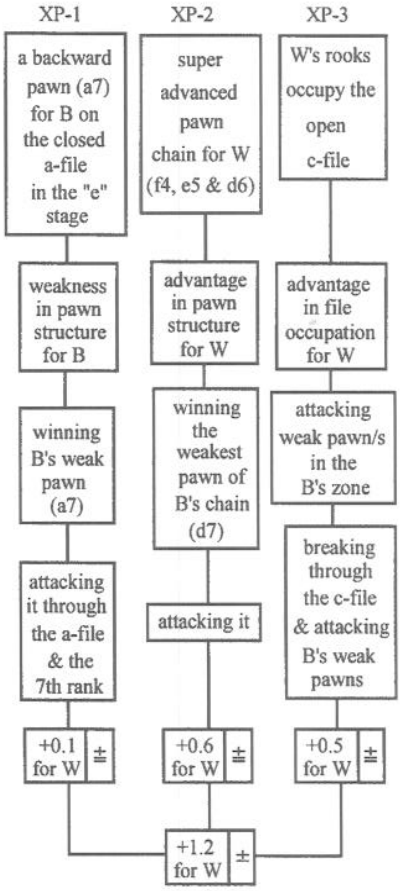

FIGURE 7 MXP for position 2.

internal content of the position and our comments are more detailed. Moreover, the application of the MXP structure for evaluating chess positions shows that the MXP structure is an appropriate knowledge structure for more than sentencing criminals. These findings led us to believe that the MXP should be examined as a suitable tool for other CBR domains in which there is a need to evaluate or to solve complex problems.

OUR MODEL

Algorithms for Evaluation of Chess Positions

A Simple Evaluation Algorithm

We aim at evaluating any given chess position by constructing a suitable MXP for it. We search our evaluation tree in order to find all basic patterns included in the position. We choose only the most important basic patterns (i.e., the patterns with the highest absolute evaluation values). We retrieve the stored XPs of these patterns and combine them into a new MXP. The retrieved XPs are analogical to the snippets (portions of cases) used in other CBR domains by Kolodner (71), Redmond (72), and Branting (73).

A description of this algorithm, algorithm 1, is given here. Given a new chess position (NCP):

1. Find the evaluation values of all basic patterns (features) included in the NCP.
2. Retrieve the XPs for the most important patterns.
3. Combine all these adapted XPs into an MXP.
4. Compute the general evaluation of the MXP of the NCP using a simple summation over the evaluation values of the retrieved XPs.

This algorithm has been used primarily in the establishment of the original database of positions and their MXPs. We use this database for evaluating new given chess positions in our case-based evaluation algorithms described in the next subsection. The positions of this database have been slightly adapted from positions taken from different relevant chess books (e.g., Refs. 64, 65, 74, and 75).

Case-Based Evaluation Algorithms

Case-based algorithms that are more creative than Algorithm 1 can also be proposed. These algorithms use the database of positions and their MXPs. Controlled learning of new positions with their MXPs enlarges the extent of this database and improves the explanation of the evaluation algorithms. General descriptions of two algorithms are given below.

Algorithm 2 uses only the MXP most suitable to the NCP. Given an NCP, the algorithm is as follows:

Retrieval of Suitable MXPs and Selection of the Best MXP

1. Find the evaluation values of all basic patterns included in the NCP.
2. Choose the most important patterns according to their absolute evaluation values. These patterns will be the indexes in the retrieval stage.
3. Retrieve all positions that their MXPs include at least one fact slot which is either one of the patterns found in Step 2 or a "brother" of one of them (according to the evaluation tree). In case of failure, jump to Step 8.

4. Compute the similarity measure relative to the NCP for each retrieved position.
5. Choose the most suitable MXP (i.e., with the highest similarity measure).

Adaptation and System Evaluation

6. Keep exactly matched XPs.
7. Adapt suitable XPs of the chosen MXP to the NCP using the evaluation tree and suitable general XPs.
8. Explain other important acts of the NCP by general XPs.

Construction of the Solution, Real-World Evaluation, and Storage

9. Combine all exactly found and adapted XPs into a new MXP.
10. Compute the general evaluation of the new MXP by summing the evaluation values of its XPs.
11. Test the proposed MXP by a chess expert and make optional improvements by hand where needed.
12. If the proposed MXP is found appropriate for acquisition then store it according to the fact slots of its XPs.

Algorithm 3 can use several MXPs suitable to the NCP. The differences between it and Algorithm 2 is Step 8. In Algorithm 3, Step 8 is as follows. For all important facts of the NCP not found in the facts of the MXP nor adaptable to the XPs of the MXP, select the next MXP suitable to the NCP, and return to Step 6.

In the next subsections, we shall give a detailed description of the most important CBR stages of Algorithm 2.

Retrieval of Suitable MXPs and Selection of the Best MXP

Given an NCP, we retrieve all MPXs that include at least one XP whose fact slot is either one of the patterns found in Step 2 or a "brother" of one of them (according to the evaluation tree). The MXP with the highest similarity measure (sm) to the NCP is chosen for the next stage of our CBR algorithm. Our similarity function has the following form: $\text{sm} = a * \text{ifs} + b * \text{ps}$, where sm is the computed similarity measure, a and b are specific constants, ifs is the important features' similarity, and ps is the position's similarity. The ifs is similar to the *contrast measure* of Tversky (76), and the ps is similar to the *nearby measure* of Botvinnik (77). Intuitively, the if and ps can be regarded as a semantic similarity and a structural similarity, respectively.

The ifs function is the computed similarity measure between the important facts (basic patterns) found in the NCP and the retrieved MXP. It is defined as follows. Let S1 be the set of all facts found both in the NCP and in the retrieved MXP. Let S2 be the set of all facts of the NCP for which we found near-neighbor patterns to them in the retrieved MXP (according to the evaluation tree). Let S3 be the set of all facts found in the NCP, but without near-neighbor facts in the retrieved MXP. Let S4 be the set of all facts found in the retrieved MXP, but without near-neighbor facts in the NCP. Then, $\text{ifs} = \alpha * \Sigma_{i \in S1}\, wi + \beta * \Sigma_{i \in S2}\, (wi * di) - \gamma * \Sigma_{i \in S3}\, wi - \delta * \Sigma_{i \in S4}\, wi$, where a, b, γ, and d are specific constants, w (weight) is the evaluation value of every discussed fact, whether it is an important fact found in the NCP or a fact slot of an XP of the discussed MXP, and d is a near-neighbor factor that measures the distance between each pair of facts.

The ps function measures the similarity between two positions: the NCP and the position related to the retrieved MXP. It is defined by $ps = c * ips + d * wms + e * bms$, where c, d, and e are specific constants, ips is the identical pieces' similarity, wms is White's material similarity, and bms is Black's material similarity. The ips is defined as the number of the exact pieces found on the same squares of the two positions divided by the number of pieces found in the NCP. The wms and bms are defined as follows:

$$wms = 1 - abs\{[wpm(NCP) - wpm(RP)]/wpm(NCP)\},$$

$$bms = 1 - abs\{[bpm(NCP) - bpm(RP)]/bpm(NCP)\},$$

where abs is the absolute function, wpm is the calculated material value of White's pieces (except the king) according to Table 1, bpm is the same function for Black's pieces, NCP is the new chess position, and RP is the retrieved position.

Adaptation and System Evaluation

After choosing the most suitable MXP, we adapt its XPs in order to construct an MXP for the NCP. In Step 6, for the facts found both in the NCP and in the retrieved MXP, we take exactly the XPs of these facts from the retrieved MXP.

In Step 7, for the facts of the retrieved MXP found as near neighbors (according to the evaluation tree) using suitable general XPs, we operate a learning process on each XP of the MXP (call each, in turn, XP-1) in order to adapt XP-1 to its matching fact in the NCP. To validate the proposed adaptation, we use some chess tests (e.g., a limited search). These tests are partly a simulation of the proposed adaptation and serve as the system evaluation to its own solution.

In Step 8, for all important facts of the NCP that are not found in the facts of the chosen MXP, we adapt suitable general XPs using chess tests.

Construction of the Solution, Real-World Evaluation, and Storage

An MXP for the NCP is proposed after combining all exactly found and adapted XPs and computing the general evaluation slot of the MXP using a summation over the evaluation values of the XPs of the new MXP. A chess expert will either approve or disapprove of this MXP. In case of disapproval, potential improvements are, at present, inserted by hand. In case of approval, a potential learning process is executed.

We have constructed a learning mechanism that is able to enlarge our database of MXPs. A new MXP will be added to the database of MXPs only if at least one adapted XP is learned (Steps 7 and 8). Such an MXP is inserted in the flat database of MXPs. The indexes that enable any kind of access (insertion or retrieval) to a MXP in this database are the fact slots of the XPs and the MXP.

A Short Example

In this section, we illustrate a use of Alogrithm 2. Assuming the NCP is position 2 (Fig. 5), we retrieve the MXP presented in Figure 6 (which is the MXP of position 1 presented in Fig. 4) as the best MXP for explaining position 2.

The adaptation process constructs the MXP presented in Figure 7 as an explanation for the NCP. Due to the lack of space, we will only explain the construction of the XP that describes the Black's backward pawn (a7) on the closed file in the e stage (i.e., XP-1 in Fig. 7) from White's viewpoint.

The fact slot of XP-1 of the NCP relates to a closed file, whereas the fact slot of

the XP-1 of the retrieved MXP relates to a semiopen file. These facts are close neighbors in the evaluation tree. We therefore choose XP-1 of the retrieved MXP for the adaptation process. In addition, we retrieve a suitable general XP according to the discussed fact of the NCP. Using these two retrieved XPs, we construct XP-1 of the new MXP.

In the fact slot, we write exactly the discussed fact of the NCP. Because the belief and purpose slots are the same in both retrieved XPs, we take them as they are. The plan slots in the two retrieved XPs are different. The plan slot of the general XP proposes to attack the weak pawn through its rank (i.e., the seventh rank). The plan slot of the XP-1 of the retrieved MXP proposes to attack the weak pawn through its file (i.e., the a file). Utilizing an elaboration strategy, we refine the plan slot of these two retrieved XPs and construct a new plan slot, which is to attack the weak pawn through both the a file and the seventh rank.

By using simple limited searching, we ensure that the new plan can be theoretically made on the board. We find a way to switch the white rook on c5 to the a file (a5) and to switch the white rook on c2 to the seventh rank (c7). For the evaluation slot, we take the evaluation value of the general XP because it relates to the same discussed fact.

Due to the approval of this MXP by our chess expert and to the construction of a new more complex plan slot, we store this MXP in the database of MXPs according to the facts slot of its XPs.

To sum up, we think that this controlled learning process will improve the evaluation ability of our algorithms because of the accumulation of new MXPs. These algorithms become more adequate by deriving more creative and better evaluations than those supplied with fewer MXPs.

Summary and Future Work Concerning the Discussed Model

We have made a contribution to CBR research by extending its range in the game-playing domain in general and in computer chess in particular. We have developed a case-based model that supplies a comprehensive positional evaluation for any chess position. This model seems to supply better explanations for chess positions than other existing computer game-playing programs. In addition, our model includes a learning mechanism that enables it to supply more adequate evaluation. We think that this model, in principle, can be generalized for evaluating any game position for any board game. The three highest levels of our evaluation tree are appropriate for game-playing in general. Whereas the king and pawn subtrees are unique for chesslike games, all other subtrees (e.g., threats and material) fit in general to all board games.

In computer chess, profound understanding has been shown to be inefficient without deep searching; therefore, to strengthen our model, there is a need to add a searching capability. Case-based planning is another important issue that we have to deal with more deeply. The plans that we retrieve for each XP of the MXP for the discussed position and adapt to fit it may be combined into one complete plan using a game-tree search. Finally, the addition of playing modules will enable our system to (besides play chess) learn in the real world and therefore to improve its evaluating, explaining, and planning capabilities.

GLOSSARY

Advanced pawn chain: White/black head of a series of pawns in a pawn chain is in the fifth/fourth rank, respectively.
Backward pawn: A pawn that has been left behind by neighboring pawns of its own color and can no longer be supported by them.
Blocked pawn: A pawn that is blocked by an opponent's piece other than a pawn.
Castled king: A king that has made a castle.
Castling: A special move between a king and a specific rook with the same color. In this move, the king is moved two squares to the direction of the rook and the rook is moved over the king and placed on the square near to it.
Center: Squares e4, d4, e5, and d5.
Closed file: A file with pawns of both colors.
Doubled pawns: At least two pawns of the same color on the same file.
Endgame: The last and deciding stage of the chess game. In this stage, the position becomes simplified and usually contains a relatively small number of pieces.
En passant capture: A special capture of a pawn that advances two squares in one move by an opponent pawn standing near the result square. This capture is done in a diagonal movement and is allowed only immediately after the advance.
Exposed king: A king without good defense, mainly without a good pawn cover.
Fork: A simultaneous attack by one piece on two of the opponent's pieces.
Grandmaster: The highest international rank in chess playing.
Isolated doubled pawns: At least two pawns of the same color on the same file that are not protected by any neighboring pawns of their own color.
Isolated pawn: A pawn that has no neighboring pawns of its own color.
Mobility: The number of potential single moves in the current position.
Open file: A file without pawns.
Passed pawn: A pawn that has no opponent's pawns that can prevent it from queening.
Pawn chain: Two consecutive series of pawns abutting on one another in consecutive diagonals.
Ply: Half-move. A move of only one side, either White or Black.
Protected doubled pawns: At least two pawns of the same color on the same file in which at least one is protected by a neighboring pawn of its own color.
Semiclosed file: A file with pawn(s) of only one's own color.
Semiopen file: A file with pawn(s) of only the opponent's color.
Superadvanced pawn chain: White/Black's head of a series of pawns in a pawn chain is in the sixth/third rank, respectively.

REFERENCES

1. K. Harkness and J. S. Battell, "This Made Chess History," *Chess Rev.*, (Feb.–Nov. 1947).
2. H. Vigneron, *Les Automates*, La Natura, 1914.
3. J. Von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, NJ, 1944.
4. C. E. Shannon, "Programming a Computer for Playing Chess," *Phil. Mag.*, *41*(7), 256–277 (1950).

5. A. M. Turing, C. Strachey, M. A. Bates, and B. V. Bowden, "Digital Computers Applied to Games," in *Faster Than Thought*, B. V. Bowden (ed.), Pitman, London, 1953, pp. 286–310.
6. M. Newborn, *Computer Chess*, Academic, New York, 1975.
7. H. J. Berliner, "A Chronology of Computer Chess and Its Literature," *AI*, *10*, 201–214 (1978).
8. P. W. Frey (ed.), *Chess Skills in Man and Machine*, 2nd ed., Springer-Verlag, Berlin, 1983.
9. D. N. L. Levy, *The Chess Computer Handbook*, Batsford, London, 1984.
10. T. Marsland and J. Schaeffer (eds.), *Computers, Chess, and Cognition*, Springer-Verlag, New York, 1990.
11. M. A. Bramer (ed.), *Computer Game-Playing: Theory and Practice*, Ellis Horwood, Chichester, 1983.
12. H. J. Berliner and C. Ebeling, "Pattern Knowledge and Search: The SUPREM Architecture." *AI*, *38*(2), 161–198 (1989).
13. H. J. Berliner and C. Ebeling, "Hitech," in *Computers, Chess, and Cognition*, T. Marsland and J. Schaeffer (eds.), Springer-Verlag, New York, 1990, pp. 79–109.
14. F.-h. Hsu, T. S. Anantharaman, M. S. Campbell, and A. Nowatzyk, "Deep Thought," in *Computers, Chess, and Cognition*, T. Marsland and J. Schaeffer (eds.), Springer-Verlag, New York, 1990, pp. 55–78.
15. J. Nievergelt, "Information Content of Chess Positions." *ACM SIGART Newslett.*, *62*, 13–14; reprinted as "Information Content of Chess Positions: Implications for Game-Specific Knowledge of Chess Players," in *Machine Intelligence*, J. E. Hayes, D. Michie, and E. Tyugu (eds.), Clarendon, Oxford, 1991, Vol. 12., pp. 283–289.
16. G. M. Adelson-Velsky, V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii, and A. V. Uskov, *Programming a Computer to Play Chess*, Russian Mathematics Surveys No. 25, Cleaver-Hume Press, London, 1970, pp. 221–262.
17. A. L. Zobrist and F. R. Carlsson, "An Advice-Taking Chess Computer," *Sci. Am.*, *228*(6), 92–105 (1973).
18. B. Balkenhol, "Data Compression in Encoding Chess Positions," *ICCA J.*, *17*(3), 132–140 (1994).
19. I. Althofer, "Data Compression Using an Intelligent Generator: The Storage of Chess Games as an Example," *AI*, *52*, 109–113 (1991).
20. A. D. de Groot, *Thought and Choice*, Mouton, The Hague, 1965.
21. A. Newell and H. Simon, *Human Problem Solving*, Prentice-Hall, New York, 1972.
22. H. A. Simon, "How Big Is a Chunk?" *Science*, *183*, 482–488 (1974).
23. K. Thompson, "Retrograde Analysis of Certain Endgames." *ICCA J.*, *9*, 131–139 (1986).
24. H. J. Berliner and M. Campbell, "Using Chunking to Solve Chess Pawn Endgames." *AI*, *23*, 97–120 (1984).
25. S. Flinter and M. T. Keane, "On the Automatic Generation of Cases Libraries by Chunking Chess Games," in *Case-Based Reasoning: Research and Development Proceedings of the First International Conference, ICCBR-95*, M. Veloso and A. Aamodt (eds.), Lecture Notes in Artificial Intelligence Vol. 1010, Springer-Verlag, Berlin, 1995, pp. 421–430.
26. R. D. Greenblatt, D. E. Eastlake, and S. D. Crocker, "The Greenblatt Chess Program," *Proc. AFIPS Fall Joint Computer Conference*, 1967, Vol 31, pp. 801–810.
27. J. S. Slate and L. R. Atkin, "CHESS 4.5: The Northwestern University Chess Program," in *Chess Skills in Man and Machine*, P. W. Frey (ed.), Springer-Verlag, Berlin, 1983, pp. 82–118.
28. D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
29. A. L. Zobrist, Technical Report 88, Computer Science Dept., University of Wisconsin, Madison (1970); reprinted in *ICCA J.*, *13*(2), 69–73 (1990).
30. D. M. Breuker, J. W. H. M. Uiterwijk, and H. J. van den Herik, "Replacement Schemes for Transposition Tables." *ICCA J.*, *17*(4), 183–193 (1994).

31. T. Marsland, "A Review of Game-Tree Pruning." *ICCA J.*, *9*(1), 3–19 (1986).
32. C. Ebeling, "All the Right Moves: A VLSI Architecture for Chess," Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, 1986.
33. J. Schaeffer, personal communication, 1994.
34. D. J. Slate, "A Chess Program That Uses Its Transposition Table to Learn From Experience." *ICCA J.*, *10*(2), 59–71 (1987).
35. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers." *IBM J. R&D*, *3*(3), 211–229 (1959).
36. T. Scherzer, L. Scherzer, and D. Tjaden, Jr., "Learning in Bebe." *ICCA J.*, *14*(4), 183–191 (1991).
37. E. Morales, "Learning Chess Patterns," in *Inductive Logic Programming*, S. Muggleton (ed.), Academic, London, 1992, pp. 517–537.
38. E. Morales, "Learning Patterns for Playing Strategies." *ICCA J.*, *17*(1), 15–26 (1994).
39. J. Nunn, "Extracting Information from Endgame Databases," in *Advances in Computer Chess, Vol. 7*, H. J. van den Herik, I. S. Herschberg, and J. W. H. M. Uiterwijk (eds.), University of Limburg, Maastricht, The Netherlands, 1993, pp. 19–34.
40. J. Schaeffer and T. A. Marsland, "The Utility of Expert Knowledge," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI-85)*, 1985, pp. 585–587.
41. D. Michie, "An Advice-Taking System for Computer Chess." *Computer Bull.*, *2*(10), 12–14 (1976).
42. I. Bratko, D. Kopec, and D. Michie, "Pattern-Based Representation of Chess End-Game Knowledge." *Computer J.*, *21*(2), 149–153 (1978).
43. I. Bratko and D. Michie, "A Representation for Pattern-Knowledge in Chess Endgames," in *Advances in Computer Chess, Vol. 2*, M. R. B. Clarke (ed.), Edinburgh University Press, Edinburgh, 1980, pp. 31–56.
44. J. Pitrat, "A Chess Combination Program Which Uses Plans." *AI*, *8*, 275–321 (1977).
45. D. E. Wilkins, "Using Patterns and Plans in Chess." *AI*, *14*, 165–203 (1980).
46. D. E. Wilkins, "Using Knowledge to Control Tree Searching." *AI*, *18*(1), 1–55.
47. R. Seidel, "A Model of Chess Knowledge: Planning Structures and Constituent Analysis," in *Advances in Computer Chess, Vol. 5*, D. F. Beal (ed.), Elsevier Science, Amsterdam, 1989, pp. 143–158.
48. K. D. Ashley and E. L. Rissland, "Compare and Contrast: A Test of Expertise," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Morgan Kaufmann, Los Altos, CA, 1987, pp. 273–278.
49. P. Koton, "Reasoning About Evidence in Causal Explanations, in *Proceedings of a Workshop on CBR*, Morgan Kaufmann, San Mateo, CA, 1988, pp. 260–270.
50. S. Bradtke and W. G. Lehnert, "Some Experiments with Case-Based Search," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA, 1988, pp. 133–138.
51. J. P. Callan, T. E. Fawcett, and E. L. Rissland, "Adaptive Case-Based Reasoning," in *Proceedings of a Workshop on CBR*, Morgan Kaufmann, San Mateo, CA, 1991, pp. 179–190.
52. R. Levinson and R. Snyder, "Adaptive Pattern-Oriented Chess," in *Proceedings of the Ninth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, CA, 1991, pp. 601–606.
53. R. Levinson, "Morph II: A Universal Agent: Progress Report and Proposal," Technical Report UCSC-CRL-94-22, University of California, Santa Cruz, CA (1994).
54. S. G. Lazzeri and R. Heller, "An Intelligent Consultant System for Chess." *Computers Educ.*, *27*(3/4), 181–196 (1996).
55. Y. Kerner, "Case-Based Evaluation in Computer Chess," in *Advances in Case-Based Reasoning*, J. P. Haton, M. Keane, and M. Manago (eds.), Proceedings of the Second European

Workshop, EWCBR-94, Lecture Notes in Artificial Intelligence Vol. 1984, Springer-Verlag, Berlin, 1995, pp. 240–254.
56. A. Samuel, "Some Studies in Machine Learning Using the Game of Checkers II—Recent Progress." *IMB J. R&D*, *11*(6), 601–617 (1967).
57. D. Michie, "A Theory of Evaluative Comments in Chess with a Note on Minimaxing." *Computer J.*, *24*(3), 278–286 (1981).
58. H. J. Berliner and D. H. Ackley, "The QBKG System: Generating Explanations from a Non-Discrete Knowledge Representation," in *Proceedings of the National Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, 1982, pp. 213–216.
59. S. Epstein, "The Intelligent Novice—Learning to Play Better," in *Heuristic Programming in Artificial Intelligence—The First Computer Olympiad*, D. N. L. Levy and D. F. Beal (eds.), Ellis Horwood, Chichester, 1989.
60. B. Pell, "A Strategic Metagame Player for General Chess-Like Game." in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Seattle, WA, 1994, pp. 1378–1385.
61. A. Nimzowitsch, *My System—A Chess Treatise*, Harcourt-Brace, New York, 1930.
62. R. Fine, *The Middle Game in Chess*, David McKay, New York, 1952.
63. L. Pachman, *Attack and Defense in Modern Chess Tactics*, P. H. Clarke (transl.), Routledge and Kegan Paul, London, 1973.
64. L. Pachman, *Complete Chess Strategy*, J. Littlewood (transl.), B. T. Batsford, London, 1976, Vols. 1 and 2.
65. A. Kotov, *Play Like a Grandmaster*, B. Cafferty (transl.), B. T. Batsford, London, 1978.
66. J. L. Kolodner, "Organization and Retrieval in a Conceptual Memory for Events of Con54, Where Are You?" in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Morgan Kauffmann, Los Altos, CA, 1981, pp. 227–233.
67. R. C. Schank (ed.), *Explanation Patterns: Understanding Mechanically and Creatively*, Lawrence Erlbaum, Hillsdale, NJ, 1986.
68. A. M. Kass, "Developing Creative Hypotheses by Adapting Explanations," Technical Report 6, Institute for the Learning Sciences, Northwestern University, Evanston, IL (1990), p. 9.
69. U. J. Schild and Y. Kerner, "Multiple Explanation Patterns," in *Topics in Case-Based Reasoning—EWCB R'93*, Lecture Notes in Artificial Intelligence Vol. 837, Springer-Verlag, Berlin, 1994, pp. 353–364.
70. Y. Kerner, "Learning Strategies for Explanation Patterns: Basic Game Patterns with Application to Chess," in M. Veloso and A. Aamodt (eds.), *Proceedings of the First International Conference*, ICCBR-95, Lecture Notes in Artificial Intelligence Vol. 1010, *Case-Based Reasoning: Research and Development*, Springer-Verlag, Berlin, 1995, pp. 491–500.
71. J. L. Kolodner, "Retrieving Events from a Case Memory: A Parallel Implementation," in *Proceedings of a Workshop on CBR*, Morgan Kaufmann, San Mateo, CA, 1988, pp. 233–249.
72. M. Redmond, "Distributed Cases for Case-Based Reasoning: Facilitating Use of Multiple Cases." in *Proceedings of the Eighth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, Menlo Park, CA, 1990, pp. 304–309.
73. L. K. Branting, "Reasoning with Portions of Precedents," in *Proceedings of the Third International Conference on AI and Law*, ACM, New York, 1991, pp. 145–154.
74. M. I. Shereshevsky, *Endgame Strategy*, K. P. Neat, (transl.), Pergamon, Oxford, 1985.
75. Y. Averbakh (ed.), *Comprehensive Chess Endings*, K. P. Neat, (transl.), Pergamon, Oxford, 1986.
76. A. Tversky, "Features of Similarity," *Psych. Rev.*, *84*(4), 327–352 (1977).
77. M. M. Botvinnik, *Computers in Chess: Solving Inexact Search Problems*, A. A. Brown, (transl.), Springer-Verlag, New York, 1984.

YAAKOV HACOHEN-KERNER