From: The Computer Journal. Vol.21 No.2 May 1978

Pattern-based representation of chess end-game knowledge

I. Bratko*, D. Kopec and D. Michie

Machine Intelligence Research Unit, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW

Students of computer chess aim at an operational theory of Master skill—operational in the sense that it can be run on the machine. In one form of the aspiration Masters must be defeated across the board under full tournament conditions, so far achieved only for 'blitz' chess but not for play under standard time control (see *SIGART Newsletter* No. 62, 1977). Another form of the 'Master skill' aspiration aims at *correct play* for defined subsets of chess. It is not known whether 'strong mastery' in this sense is attainable for the complete game or whether chess is 'hard' in the sense of Knuth (1976). We can, however, start with elementary endings such as King and Queen *versus* King (denoted KQK), KRK, KBBK, KBNK and KPK, and seek to extend mastery backwards a step at a time into the game's increasingly complex hinterland.

Work at Edinburgh follows the second approach, seen as a means for studying forms of knowledge representation in relation to three *desiderata*: (*a*) forms more powerful than present programming languages for specifying strategies, (*b*) forms more suitable for proofs of correctness of strategies and (*c*) forms more convenient for automatic optimisation of strategies ('machine learning').

None of the above listed end-games contains anything problematical from a Master's point of view and computer programs embodying correct strategies have been written for all of them. In reviewing this work Bramer (1977) remarks that the task, not of playing such end-games correctly, but of expressing in program form the knowledge required for correct play, has turned out to be surprisingly and disproportionately hard. For his own implementations of KRK and KPK Bramer uses pattern-based models of a general kind now accepted as indispensable to the extension of machine mastery into more complex chess subdomains. Such exercises as KRK and KPK can be done (with some difficulty) *without* special programming tools; but it needs only a small step in the direction of greater complexity to bring us into territory where the use of such tools become critical.

In this paper we describe pattern descriptional aids to strategy building in two areas more complex than KRK, KPK, KQK and the rest; namely (*a*) pawns-only positions and (*b*) the defence of king and knight against king and rook.

Describing pawn structures

Tan (1977) has developed a program which breaks down any K + P ending into a basic description of its components. Using a vocabulary which is defined in Kmoch's (1959) *Pawn Power*, the pawn formations are broken down into Fronts, i.e. islands containing opposing pawns, and further subdivided into Groups (same colour only). The rôle of each pawn in terms of its relationships to other pawns is then defined, as shown in the upper part of **Fig. 2** which uses terms explained by Tan as follows:

> 'An enemy pawn ahead on the same file is a *counterpawn*, and a *sentry* when it is on a neighbouring file . . . Counterpawns and mutual sentries of distance 1 are called *rams* and *levers* respectively. Friendly relations give rise to a *duo* when the pawns are abreast on two neighbouring files, and a *twin* (doublepawn) when they are on the same file. A backward neighbour is a *protector* (distance 1) or a *potential protector* (distance > 1).'

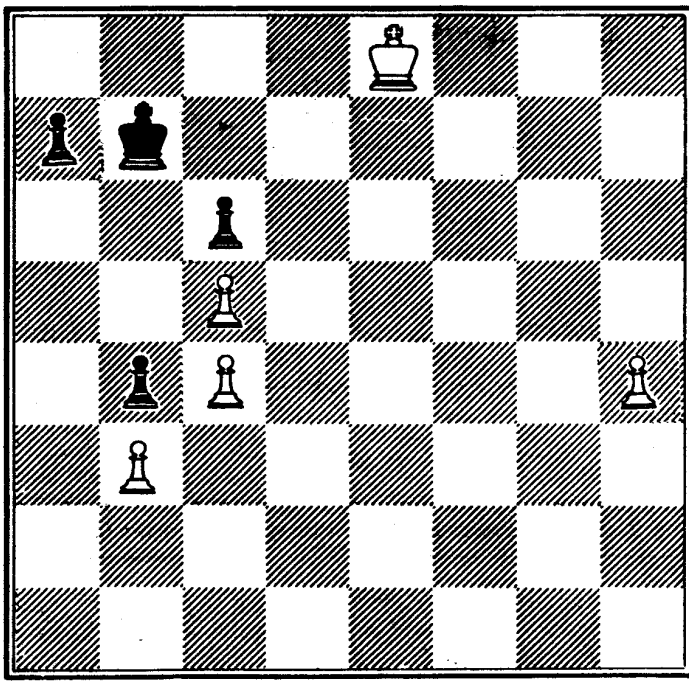

Fig. 1 From Basic Chess Endings (R. Fine)

Some of these relations may be very useful if developed further. For example, if a pawn is 'overloaded', in that it is performing several rôles at once, its removal may lead us to a winning strategy. An example in practice is shown by the problem position in **Fig. 1** taken from Fine's (1964) *Basic Chess Endings*. Its solution becomes clearer when we consider the number of rôles performed by the pawn on square 33 (square = 9 × file + rank).

Thus,

33 [1, 31], i.e. pawn on 33 is a Counterpawn to pawn on 31,

33 [2, 32], i.e. pawn on 33 is a Ram to pawn on 32,

33 [3, 21], i.e. pawn on 33 is a Sentry to pawn on 21,

33 [8, 33], i.e. pawn on 33 is a Potential Protector to pawn on 22.

Rather than queening the passed RP immediately, which would result in a quick draw due to stalemate, the solution lies in White's capture of that pawn on 33.

A graph representation of the same position is given in the lower part of Fig. 2, using Tan's notation with the addition of ∧ to denote a passed pawn.

Another concept put forward by Tan is the ADD (Attack Defence Diagram, see **Fig. 3**). Some of the relations proposed for an ADD are as follows: (*a*) relations within fronts; (*b*) defences to threats arising from (*a*); (*c*) possible attacks of kings against pawns; (*d*) defences to (*c*); (*e*) support possibilities; (*f*) joint attacks. Tan also breaks down the relationships that go into the ADD into Backus-Naur Form. As yet there is

*Present address: Jozef Stefan Institute, Ljubljana, Yugoslavia.

	GRAPHICAL NOTATION	
HOSTILE RELATIONS	Counterpawn	
	Ram	< + >
	Sentry	
	Lever	
FRIENDLY RELATIONS	Duo	
	Twin	
	Potential Protector	
	Protector	
	Passer (our notation)	

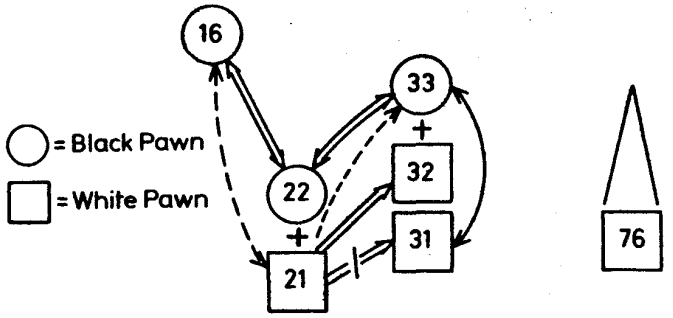

Fig. 2 Graphical representation for chess position shown in Fig. 1

no working program for the ADD. As a step towards implementation, we have tested how the evaluations would work on some simple K + P endings. It seems that at least three concepts must be added: (*a*) opposition, (*b*) triangulation, (*c*) outside passed pawns. These features are important and occur frequently.

AL1 'advice taker' system

In a related use of pattern-based representations of chess knowledge we have developed a linguistic vehicle for applying McCarthy's (1959) 'advice taker' concept. In Advice Language 1 (Michie, 1976) knowledge is conveyed to the system in the form of one or more *advice tables*, each specifying a number of *rules*. A rule is applied to a position (in a manner familiar to commercial users of decision tables, and to academic users of production systems) if and only if the position matches the rule's 'condition pattern'. Associated with each rule is a list of *pieces of advice*. Each piece of advice is specified in terms of Huberman-type (Huberman, 1968) better-goals and holding-goals, together with move-constraints to control the branching of the search and a depth limit to terminate it when no way has been found of achieving the given better-goals.

AL1 has been implemented in the POP-2 programming language as a package consisting of four comparatively independent modules (core-occupancy of compiled code on the PDP-10 is shown in parentheses):

1. A *problem-solver* performs tree search in whatever problem space is specified to it by the legal move generator, using the domain specific knowledge contained in the currently loaded Advice Tables (2K).
2. An *Advice Table editor* acts as a link between the system and the user, enabling him to create, extend and modify the system's Table held knowledge interactively (4K).
3. A *playing module* executes a strategy generated by the problem-solver in the form of a Huberman-type forcing tree (6K).
4. *Chess-relevant but subdomain independent POP-2 predicates* act as the building blocks from which the table writer assembles relevant patterns and pieces of advice from which to construct his rules (13K).

In addition subdomain specific predicates are normally required for each new Advice Table. The POP-2 system itself occupies 19K 36 bit words of store.

Modules 1 and 2 are chess independent and can be used for solving other combinatorial problems. The domain specific knowledge directs the action of module 1 in the following way: a rule is invoked by a pattern-match with the current situation (chess position) and the corresponding pieces of advice are then tried one by one until module 1 can find a 'forcing tree' that guarantees the achievement of better-goals while preserving holding-goals.

Considered as an ultra-high level programming language, AL1 seems to provide a natural means of describing heuristics in combinatorial algorithms. In one experiment (Michie, 1976) the King + Rook v. King ending (KRK), regarded by Zuidema (1974) as a laborious programming task, was expressed as an Advice Table and a strategy checked out at a cost of only two man-days. More recently (Bratko, 1978) the whole mating procedure known from the chess books was compressed into a Table of only one rule, comprising 5 pieces of advice expressible as follows:

1. Look for a way to mate opponent's king in two moves.
2. If the above is not possible, then look for a way to further constrain the area on the chessboard to which the opponent's king is confined by our rook.
3. If the above is not possible, then look for a way to move our king closer to opponent's king.
4. If none of the above pieces of advice 1, 2, 3, works, then look for a way of maintaining the present achievements in the sense of 2 and 3 (i.e. make a waiting move).
5. If none of 1, 2, 3, or 4 is attainable then look for a way of obtaining a position in which our rook divides the two kings either vertically or horizontally.

The 'and-or' tree search, carried out by module 1 of the AL1 system when generating a forcing tree satisfying a corresponding piece of advice, is limited to a depth of 2 ply for pieces 2, 3 and 4, and to 3 ply for pieces 1 and 5. Quality of play was respectable by the standards of the chess books, never needing more than 25 moves to force the mate. The worst case minimax-optimal path length is known to be 16 moves (Clarke, 1977).

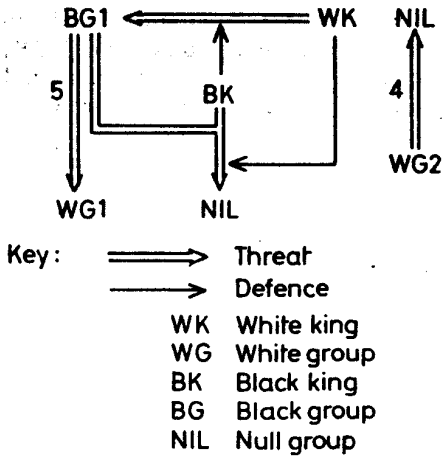

Fig. 3 The ADD corresponding to the position shown in Fig. 1. In the form described by Tan (1977) the ADD would not indicate the self-stalemate threat which the BK generates jointly with BG1. The above diagram is based on an extended notation which takes care of this. For a fuller account Tan's paper should be consulted. Integers denote the minimal number of moves required to carry out a threat

HOW DIFFICULT IS THE KNKR PROBLEM?

From chess *player's* point of view:

—**Chess books:** Keres, KNKR, 2 pages.
Fine, KRKN, 8 pages.
Longest variation in Fine before capture of the Knight: 24 moves; longest known variation 27 moves.

—**Tournament games:** usually a comparatively easy draw, but there are examples where the weaker side went wrong (e.g. Neumann-Steinitz, 1870).

From chess *programmer's* point of view:

—**Knowledge v. search:**

1. Rules of chess: *don't get mated!* Up to 24 moves before losing knight plus up to 16 moves before being mated:

$\approx$ 80 ply

2. Additional advice: *don't lose knight!*

$\sim$ 48 ply

3. Additionally: *keep king and knight together!*
Necessary, and probably sufficient, lookahead is:

10 ply

4. *Advice contained in KNKR table.*
To preserve draw, and conserve king-centrality:

4 ply

Fig. 4 Salient features of the KNKR end-game. The size of the total problem space, after reduction by disregarding symmetric cases, lies between 3×10^6 and 4×10^6. The boxed figures show the depth of search necessary, for the program's given state of knowledge, to select a draw-preserving move. Ply = half move

The KNKR game

In further experiments with the AL1 system, the king + knight v. king + rook ending (KNKR) was used as an experimental domain that is not trivial from the human expert's point of view. **Fig. 4** presents some data that illustrate the difficulty of this end-game. The first two items, chess books and tournament games, give some insight into the difficulty from the *chessplayer's* point of view. The difficulty from the *programmer's* point of view is illustrated by the third item which indicates the relationship between the amount of knowledge possessed by the program and the depth of game-tree search required for correct play.

The KNKR ending is usually drawn, but under certain circumstances the stronger side (the one with the rook) can win. A winning procedure, when there is one, consists usually of combining three basic principles (Fine, 1964):

1. Create mating threats.
2. Force separation of king and knight.
3. Stalemate and capture the knight.

The KNKR advice table must cope with the above threats and thus preserve a draw for the weaker side when starting from a theoretically drawn position (in all such, the king and knight are not separated). The table, shown in **Fig. 5**, contains enough knowledge (according to experimental tests) both to preserve the draw in positions with king and knight sufficiently close together *and* to maintain the degree of centrality of the weaker side's king *while searching to a depth of at most 4 ply* (half-moves). Centralisation of the king is important to the weaker side because mating threats can occur only when the king is on the edge. Therefore when the king is on the edge the defence becomes considerably more difficult. The KNKR table thus actually conserves the degree of easiness of defence. When the king is started on the edge and not separated from the knight, for those positions which are theoretically drawn, the table preserves the draw in all cases tested. Recently a class of specially tricky positions has been discovered by D. Kopec, not previously known in chess literature, where the only correct defence requires a counter-intuitive separation of king and knight. A correct treatment of such positions with king and knight separated would require additional knowledge.

The upper table of Fig. 5 specifies four rules (CR, R1, R2, ER) by the 'Yes, No, Don't-care' column patterns. These patterns refer to POP-2 predicates flanking the table:

OKEDGE —our king on the edge;

OKONSEP —our king and our knight separated (distance greater than 4 in king moves);

CORNCASE—corner case, a special 'classical' situation (e.g. Fine, 1964) with the king in the corner, requiring exceptional treatment.

The current chess position is matched against the rule patterns from left to right. As soon as a match is found the corresponding rule is applied. For example, if the position does not satisfy the CORNCASE condition and the king is on the edge and the king and knight are not separated, then rule R2 matches the position and the list of pieces of advice 1, 5, 6, 7, 8, 12 is applied. The pieces of advice are defined by the lower table in Fig. 5. Consider for example Advice no. 1 called KILLROOK. The better-goal to be satisfied in them-to-move positions specified with this piece of advice is TRDEAD (their rook dead). As already stated, depth of search is limited to 4 ply, and to improve search efficiency we also specify holding-goals NOT ONLOST (not our knight lost without compensation) and TRDEAD OR CHECK (their rook dead or their king in check). By this the search is limited to consideration of immediate captures and checking-moves, which amounts to looking for ways of forking their king and rook. This tactic is indicated by the fact that there is no other way to force capture of the rook.

When testing the correctness of the table a variety of players, two of National Master strength (rated over 2300 on the international scale), have engaged the system in play for a total elapsed time of more than 10 hours (150 moves on each side, starting from different positions). No absolute way exists of proving correctness for all possible positions short of either (*a*) exhaustive checking through the total space of 3-4 million

	KNKR Table	CR	R1	R2	ER
				rules	
condition predicates	OKEDGE	—	N	Y	—
	OKONSEP	—	N	N	—
	CORNCASE	Y	—	—	—
lists of pieces of advice		9	1	1	10
			2	5	11
			3	6	12
			4	7	
			11	8	
				11	
				12	

		Better-goals		*us-to-move*			*Holding-goals*				*them-to-move*				
		OKONNDLT (utm)	TRDEAD (ttm)	NOT MATE	NOT ONLOST	ONSAFE2P	TRDEAD OR CHECK	NOT ONOKATT	OKEDNDGE	OKCONDGE	OKONDLE2	KINGSCLOSE	OKONDLE3	TKONDGT1	OKONDEQ4
pieces of advice	1: KILLROOK	—	Y	(Y)	Y	—	Y	—	—	—	—	—	—	—	—
	2: HOLD1	—	—	(Y)	Y	Y	—	Y	Y	Y	Y	Y	(Y)	Y	—
	3: HOLD2	—	—	(Y)	Y	Y	—	Y	Y	—	Y	Y	(Y)	—	—
	4: HOLD3	—	—	Y	Y	Y	—	Y	—	Y	Y	Y	(Y)	—	—
	5: HOLDEDG1	—	—	Y	Y	Y	—	Y	—	Y	Y	—	(Y)	—	—
	6: HOLDEDG2	—	—	Y	Y	Y	—	—	—	Y	—	—	Y	—	—
	7: HOLDEDG3	—	—	Y	Y	Y	—	—	—	—	Y	—	(Y)	—	—
	8: HOLDEDG4	—	—	Y	Y	Y	—	—	—	—	—	—	Y	—	—
	9: CORNCASE	—	—	(Y)	(Y)	(Y)	—	—	—	—	—	—	—	—	Y
	10: APPROKON	Y	—	Y	Y	Y	—	—	—	—	—	—	—	—	—
	11: SURVIVE1	—	—	Y	Y	Y	—	—	—	—	—	—	—	—	—
	12: SURVIVE2	—	—	Y	—	—	—	—	—	—	—	—	—	—	—

Fig. 5 **The upper table is the KNKR Advice Table as written and tested. The integer lists index a repertoire of 12 pieces of advice. These are shown in the lower table expanded in the form of calls to indicated subsets of the 14 goal predicates listed along the top. 'Y's enclosed in brackets are logically implied by the other predicates selected in the same row. The symbols in parentheses (utm) and (ttm) mean 'us-to-move' and 'them-to-move' respectively**

positions or (*b*) formal proof. AL1's tabular format offers simplifications which make the latter an attractive topic for study.

When playing the KNKR ending on the PDP-10 computer the present implementation of AL1 spends on average about one minute of computer time per move, mostly due to the comparative inefficiency of the forcing-tree generating routine. The program examines about 10 nodes in the game-tree per second. When run on comparable machines, other chess-playing programs, e.g. CHESS 4.5 (Slate and Atkin, 1977) or MASTER (Birmingham and Kent, 1977) examine at least a few hundred positions per second. A new version, AL2, is under construction with an eye to increased run-time efficiency, among other improvements. But considering AL1's efficiency from the point of view of programmer productivity, these experiments gave evidence of great savings. Table writing and check-out for KNKR occupied one of us (I.B.) for less than six weeks. We doubt whether correct play, especially if 'centrality preservation' is to be included, could be programmed using standard methods in less than a substantial multiple of this figure other than by promoting large proliferations of forward search. As an annotation on this last remark, we append results, kindly supplied by D. Slate, of having the leading US tournament program CHESS 4.5 play the KNKR game against an expert opponent from selected starting positions.

Performance of CHESS 4.5 tournament program with KNKR

Tournament programs have the aim of playing reasonably well, but not of course with guaranteed correctness, in all phases of the game: opening, mid-game, and ending. Such 'general' chess programs cannot pay much attention to specific features of different position-types. Rather these programs embody generalised chess principles, or heuristics, hopefully applicable to the large majority of possible positions. Lack of position-type specific knowledge is to some extent balanced by deep lookahead, facilitated by fast game-tree search routines, efficient tree-pruning, efficient coding, and fast hardware.

CHESS 4.5 was required to defend the weaker side of KNKR against a human opponent rated just over 2000 on the US Chess Federation scale, i.e. an 'expert'. The program ran on a CDC 6400 machine, on which it was able to win the 1976

ACM Computer Chess Championship (more recently it has had highly successful trials on the much faster Cyber 176). CHESS 4.5's general evaluation function was used without allowing any adjustment or special 'tuning' to the KNKR problem. Search depth was set to 7 ply. Since forced variations are searched beyond this pre-set horizon, moves 8 ply deep were occasionally searched in the present case. Under these conditions, CHESS 4.5 typically looked at a few tens of thousands of nodes per move and spent up to 120 seconds per move, typically between 30 and 60 seconds.

Three trials were made, using test positions taken from those used in the experimental validation of the KNKR table earlier described.

1. A 'classical' difficult defence (Fine, 1964; Keres, 1974) with the weaker side's king in the corner. CHESS 4.5 found correctly the move considered most difficult in the books, but then stumbled on the fourth move of the main 'book' variation, obtaining a lost position.
2. Another difficult position, with the weaker side's king on the edge (Keres, 1974). CHESS 4.5 found the only correct defence against the main line given by Keres (i.e. 8 best moves in a row).
3. A further position taken from our own tests, with the weaker side's king in the centre (easiest defence). CHESS 4.5 allowed its king to be driven to the edge resulting in a harder defence. This enabled the opponent to create mating threats, and after additional weaker moves by the program the king and knight got separated, leading to a lost position.

It is interesting to observe that the program's opponent, although an expert, after achieving theoretically won positions never grasped the opportunity actually to defeat the program. This has a bearing on the level of difficulty of this subdomain.

Conclusion: thanks to efficient tree-search, CHESS 4.5 was able to find a correct move in many difficult positions. But the lack of specific advice: 'Keep king and knight together!' and 'Preserve the centrality of the king!' could not be entirely compensated by the efficient and comparatively deep search to 7 or 8 ply.

Discussion

It was pointed out by Shannon (1950) on general grounds, and more recently by Berliner (1974) on the basis of authoritative new theoretical and experimental work, that fast tree-search and uniform heuristics will not suffice for mechanising the highest levels of chess skill. In spite of impressive recent progress up the human tournament scale by 'brute force' programs the knowledge-gap from which these programs suffer still bars them from the higher reaches. The work here reported shows that the weaknesses inherent in the brute force style can be shown up even by quite a simple chess subdomain. The same subdomain, however, yielded readily to a more knowledge-oriented approach, for which the AL1 system provided highly effective support.

The underlying formal model of the AL1 problem-solver closely matches the basic structure of a range of combinatorial problems. Our experience supports the idea that the Advice Language methodology should be applicable to problems in such areas as algebraic manipulation, symbolic integration, robot plan-formation, and a variety of optimisation and scheduling tasks. While detailed accounts appear elsewhere (Bratko, 1978; Bratko and Michie, 1978), this brief overview was prepared in the hope of arousing interest among those actively engaged in one or another of such areas.

Acknowledgement

Thanks are due to the Research Community of Slovenia, to the British Council and to the University of Edinburgh for support and facilities.

References

BERLINER, H. (1974). Chess as problem solving: the development of a tactics analyser, Ph.D. Thesis. Pittsburgh: Carnegie-Mellon University.

BIRMINGHAM, J. A. and KENT, P. (1977). Tree-searching and tree-pruning techniques, *Advances in Computer Chess 1* (ed. M. R. B. Clarke), pp. 89-107, Edinburgh: Edinburgh University Press.

BRAMER, M. A. (1977). Representation of knowledge for chess endgames: towards a self-improving system, Ph.D. Thesis, Milton Keynes: The Open University.

BRATKO, I. (1978). Proving properties of strategies expressed in the AL1 assertional language, (*Inf. Proc. Letters*, forthcoming.)

BRATKO, I. MICHIE, D. *et. al.* (1978). A representation for pattern-knowledge in chess end-games, *Advances in Computer Chess 2*, (ed. M. R. B., Clarke, forthcoming).

CLARKE, M. R. B. (1977). A quantitative study of king and pawn against king, *Advances in Computer Chess 1* (ed. M. R. B. Clarke) pp. 108-118, Edinburgh: Edinburgh University Press.

FINE, R. (1964). *Basic Chess Endings*, New York: David McKay Company.

HUBERMAN, B. J. (1968). A program to play chess end games, *Technical report no. CS106*, Stanford University: Computer Science Department.

KERES, P. (1974). *Practical Chess Endings*, London: Batsford Ltd.

KMOCH, H. (1959). *Pawn Power*, New York: David McKay Company.

KNUTH, D. (1976). Mathematics and computer science: coping with finiteness, *Technical report STAN-CS-76-541*, Stanford: Department of Computer Science.

MCCARTHY, J. (1959). Programs with common sense, *Mechanisation of Thought Processes 1*, London: HMSO.

MICHIE, D. (1976). An advice-taking system for computer chess, *Computer Bulletin*, Series 2, No. 10, pp. 12-14.

SHANNON, C. E. (1950). Programming a computer for playing chess, *Phil. Mag*, (Lond.), 7th ser. 41, pp. 256-75.

SLATE, D. J. and ATKIN, L. R. (1977). CHESS 4.5—the Northwestern University chess program, *Chess Skill in Man and Machine* (ed. P. Frey), pp. 82-118, New York: Springer Verlag.

TAN, S. T. (1977). Describing pawn structures, *Advances in Computer Chess 1* (ed. M. R. B. Clarke), pp. 74-88, Edinburgh: Edinburgh University Press.

ZUIDEMA, C. (1974). Chess, how to program the exceptions? *Afdeling Informatica*, IW21/74, Amsterdam: Mathematisch Centrum.