STATIC RECOGNITION OF POTENTIAL WINS IN KNNKB AND KNNKN

E.A. Heinz*

International University (IU), School of IT, Kasernenstr. 12, D-76648 Bruchsal, Germany

ernst_a_heinz@web.de; http://www.i-u.de/schools/heinz/

Abstract The fact that the strong side cannot enforce a win in KNNK makes many chess players (both humans and computers) prematurely regard KNNKB and KNNKN to be trivially drawn too. This is not true, however, because there are several tricky mate themes in KNNKB and KNNKN which occur more frequently and require more complicated handling than common wisdom thinks. The text analyzes the mate themes and derives rules from them which allow for the static recognition of potential wins in KNNKB and KNNKN without further lookahead by search. Although endgame databases achieve the same goal, they are normally far less efficient at doing so because of their additional I/O and memory requirements (even when compressed).

Keywords: computer chess, endgame play, KNNKB, KNNKN, static recognition

1. Introduction

Usually, two bare Knights are not much of a force when it comes to mating in late endgames such as KNNK, KNNKB, and KNNKN. It is well-known that these endgames are generally drawn despite the substantial material advantage enjoyed by the strong side (Thompson, 1991; The Editors, 1992; Nalimov, Haworth, and Heinz, 2000, 2001). Human chess players and chess programs alike tend to incorporate rules of thumb classifying bare KNN constellations as most unlikely to win. Thus, common chess wisdom avoids KNN types of positions when being ahead in material and goes for them otherwise. A crude way to implement the heuristic is by scoring essentially all KNNK, KNNKB, and KNNKN positions as draws. Like many others, an early version of our own chess program DARKTHOUGHT (Heinz, 1997, 2000) did exactly this back in

*This work originally started back in the mid-1990s while the author still was a Ph.D. candidate at the School of Computer Science, University of Karlsruhe, Germany, and then continued throughout his stay as a postdoctoral fellow at the M.I.T. Laboratory for Computer Science, USA, from 1999 to 2001.

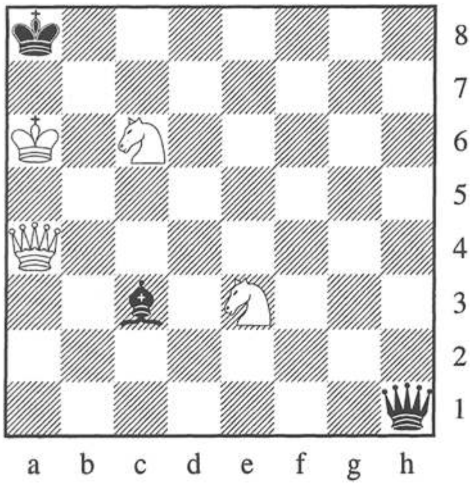

Figure 1. Black's 1..Qa1? loses to 2. Qxa1 Bxa1 (see Fig. 3) – 1..Qb1! draws.

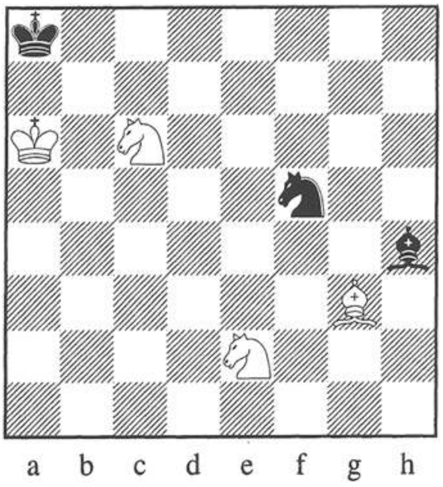

Figure 2. White's 1.Bxh4! wins as 1..Nxh4 is a forced mate (see Fig. 4).

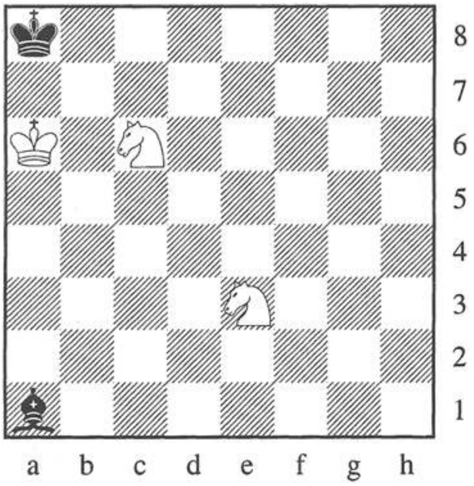

Figure 3. White mates in 2 moves: 1.Nd5! and 2.Nc7# or 2.Nb6#.

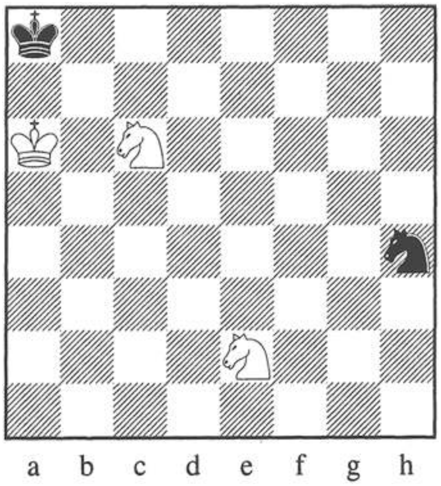

Figure 4. White mates in 3 moves: 1.Nc3 {Nf4}, 2.Nd5, 3.Nc7# {Nb6#}.

mid-1995. Then, at the end of some blitz test games, it encountered the two positions shown in Figures 1 and 2 where it happily went for the continuations leading to Figures 3 and 4 respectively, mistakenly scoring them both as draws. DarkThought played without endgame databases and short on time, so it saw the loss only after having manoeuvered itself into it.

Of course, the aforementioned scenario with the so-called "horizon effect" visible at low search depths is nothing unusual in computer chess. It was quite special, however, that the horizon effect occurred with full severity (score dropping from draw to being mated) in seemingly trivial circumstances (KNNKB and KNNKN). This strongly aroused my curiosity and sparked the work that eventually led to the development of interior-node recognizers (Heinz, 1998, 2000), knowledgeable RAM-based endgame databases (Heinz, 1999a, 2000), and efficient endgame indexing (Heinz, 1999b, 2000; Nalimov *et al.*, 2000, 2001) plus their implementation in DARKTHOUGHT. Hence, those rather innocent-looking two positions from Figures 1 and 2 were in fact instrumental for much of my endgame-related research up to date.

Solving the KNNKN "mate in 3" of Figure 4 requires a 5-ply search with 4 quiet half-moves before the final checkmate: White's 1. Nc3 {Nf4} and 2. Nd5 plus Black's respective answers. Consequently, standard quiescence searches following either checks and captures or captures only cannot spot the win unless supported by lucky hits in the transposition table. The same holds for normal full searches with remaining depths of ≤ 3 plies in case of capture-check quiescence and ≤ 4 plies in case of capture-only quiescence. Therefore, the search alone most likely fails to resolve the mate in this simple position if it occurs far out near the lookahead boundary. According to Thompson (1991), The Editors (1992), and Nalimov *et al.* (2000, 2001) the endgames KNNKB and KNNKN feature even harder positions than the ones from Figures 3 and 4: the longest forced win for KNNKB is "mate in 4" (see Figure 5) and for KNNKN it is "mate in 7" (see Figure 6). Because of the checks and single-reply moves involved here, normal full searches with extensions and quiescence searches with checks included might actually resolve these deeper mates more easily than my two example positions with their many quiet moves.

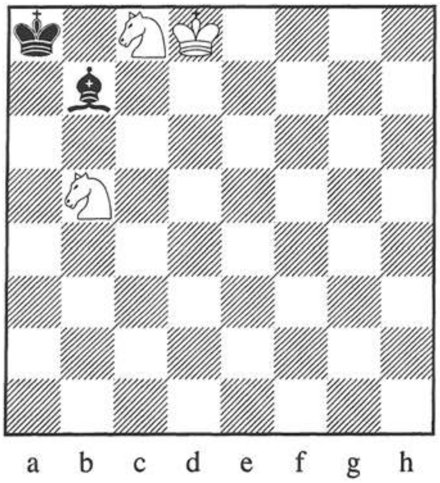

Figure 5. White mates in 4 moves: 1. Nb6+ Kb8 2. Nd7+ Ka8 3. Kc7 Bh1 {or any other legal move by B} 4. Nb6#.

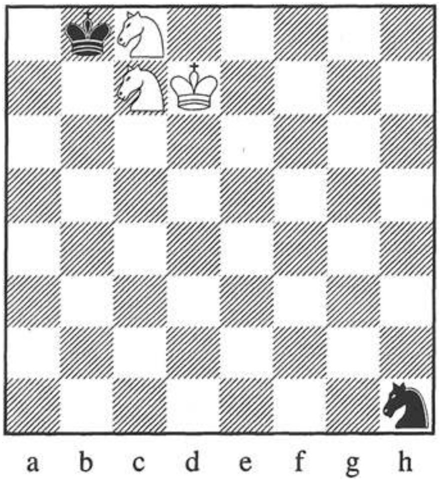

Figure 6. White mates in 7 moves: 1. Na6+ Kb7 2. Nc5+ Kb8 3. Ne7 Ng3 4. Nc6+ Ka8, 5. Kc7, 6. Nd7, 7. Nb6#.

An obvious solution to the problem is the usage of omniscient endgame databases that return the exact distance-to-win (mate or conversion to another won subgame) when queried. In practice, this does not really work out because endgame databases (even in compressed format) usually reside on secondary storage media due to their large sizes. Thus, their querying incurs considerable performance penalties and additional memory consumption for caching purposes. As a good compromise between accuracy and speed, most chess programs do not query any endgame databases in the quiescence search and very often stop doing so a few plies above the main lookahead boundary already. Please note, however, that in the particular case of KNNKB and KNNKN it is possible to copy Nalimov's compressed tablebases (Nalimov *et al.*, 2000, 2001) to a RAM disk requiring less than 1 MB of memory and access them from there. Performance-wise, the necessary I/O, index calculations, and data decompression still lose against the static recognition rules suggested by me later on in this text – but actually not by much. Yet, the special database setup does not allow for any generalization regarding other similar positions. In particular towards this end, I see excellent promise of the rule-based approach though.

The remainder of this text is structured as follows. The next section discusses related work. Then, the subsequent sections focus on the various mate themes in KNNKB, KNNKN, and their subgames (namely KBKN, KNKN, and KNNK). These themes lead to the derivation of recognition rules and the final formulation of the full algorithm for the static recognition of potential wins in KNNKB and KNNKN. Last but not least, a wrap-up of the main findings and a look into the future conclude the work.

2. Related Work

There exists an ample body of related work covering endgame databases and infallible rule-based endgame play in chess. Both areas feature a long and rich history of interesting research. The introductory section above already referred to some important contributions in the field of endgame databases, namely (Thompson, 1991; The Editors, 1992; Nalimov *et al.*, 2000, 2001). An elaborate discussion of endgame databases and their history was provided by Heinz (1999b, 2000). The introduction also mentioned the interior-node recognizers and knowledgeable endgame databases of DARKTHOUGHT (Heinz, 1998, 1999a, 2000). Both are of special interest here because the static recognition rules for KNNKB and KNNKN are to augment that very recognizer framework.

The rest of this section now focusses on infallible rule-based endgame play in chess. As early as 1890, Torres y Quevedo built a marvelous electro-mechanical machine which played and won many of the hardest KRK positions. Tan (1972) implemented the first program that achieved seemingly infallible play for the KPK endgame. Tan's excellent set of rules solved all difficult KPK positions known by then, including Averbakh's and Fine's famous examples. But because omniscient KPK endgame databases were not yet available in 1972, the hypothesized perfectness of the program remained unproven. However, Tan's program doubtlessly pioneered the usage of decision trees with multi-valued nodes and leaves representing specific pattern knowledge about the respective endgame domain. Decision trees became an integral part of nearly all subsequent works which focussed on the explicit construction or automatic deduction of complete rule sets for infallible endgame play in chess. Later on, Bratko, Kopec, and Michie (1978), Bramer and Clarke (1979), and Bratko and Michie (1980) presented more refined representation schemes for pattern knowledge in chess endgames. Several good examples of these and other predominantly hand-crafted rule sets are listed below.

- **KBNK** – van den Herik (1983);
- **KNNKP(h)** – Herschberg, van den Herik, and Schoo (1989);
- **KNP(h)K** – van den Herik (1980, 1982);
- **KPK** – Tan (1972), Beal (1977), Beal and Clarke (1980), Bramer (1980a, 1980b), Niblett (1982), Barth and Barth (1992);
- **KPKP** (both P passed) – Bratko (1982), Barth (1995);
- **KRK** – Torres y Quevedo [1890], Zuidema (1974), Bramer (1980a, 1982);
- **KRKN** – Bratko and Niblett (1979), Kopec and Niblett (1980);
- **KQKP** – Barth and Barth (1992);
- **KQKQ** – Barth and Barth (1992), Weill (1994).

The surprising complexity of rules and knowledge bases for "simple" problem domains (such as 3-piece endgame databases in chess) ignited the interest of researchers in learning such infallible rules automatically. Especially the KRK and KPK endgame databases became extremely popular for automatic learning experiments. Many works that try to automate the inductive acquisition of rules for infallible endgame play in chess also employ decision trees as their central resources for the representation of semantic knowledge. The following brief overview of publications about automatic learning of infallible endgame play in chess and the inductive acquisition of rule-based knowledge therefor is meant to serve as a mere introduction to the field. Any more comprehensive summary clearly lies beyond the scope of this text.

- **KBBKN** (selected positions) – Muggleton (1988);
- **KBRK** (extended chess version) – Coplan (1998);

- **KNNKP(h)** – van Tiggelen and van den Herik (1991), van Tiggelen (1991, 1998);
- **KPK** – Michalski and Negri (1977), Negri (1977), Shapiro and Niblett (1982), Shapiro (1987), Coplan (1998);
- **KP(a7)KR** – Shapiro and Michie (1986), Shapiro (1987), Muggleton (1990);
- **KRK** – Bain (1994), Bain and Muggleton (1994), Bain and Srinivasan (1995);
- **KRKN** – Quinlan (1979, 1983), Shapiro (1987), Verhoef and Wesselius (1987).

3. Checkmates in KBKN and KNKN

Although the subgames KBKN of KNNKB and KNKN of KNNKN are trivially drawn, they actually do feature a few checkmate positions. These are shown in Figure 7 where each quadrant of the board depicts its own mate theme to be viewed independently of the others. Yet, normal non-mate positions in KNKB and KNKN do never forcibly lead to the side on move being mated (i.e., there are only direct "mates in 1" because the side on move can always evade any mating attempt). Still, the mate themes of Figure 7 demand proper attention because they also apply to KNNKB and KNNKN where both the strong as well as the weak side might mate the opponent accordingly. The static recognition rules must take all these possibilities into full account.

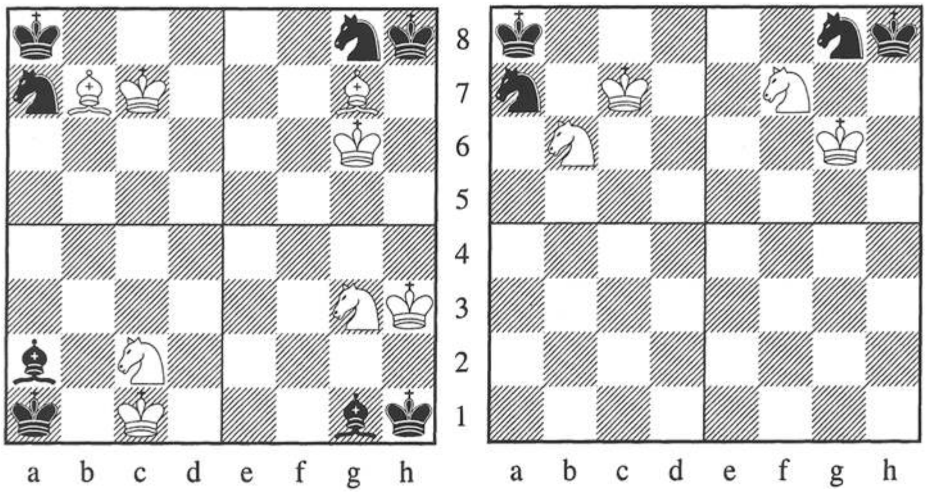

Figure 7. Mate themes in KBKN and KNKN (corner traps, not enforceable).

4. Checkmates in KNNK

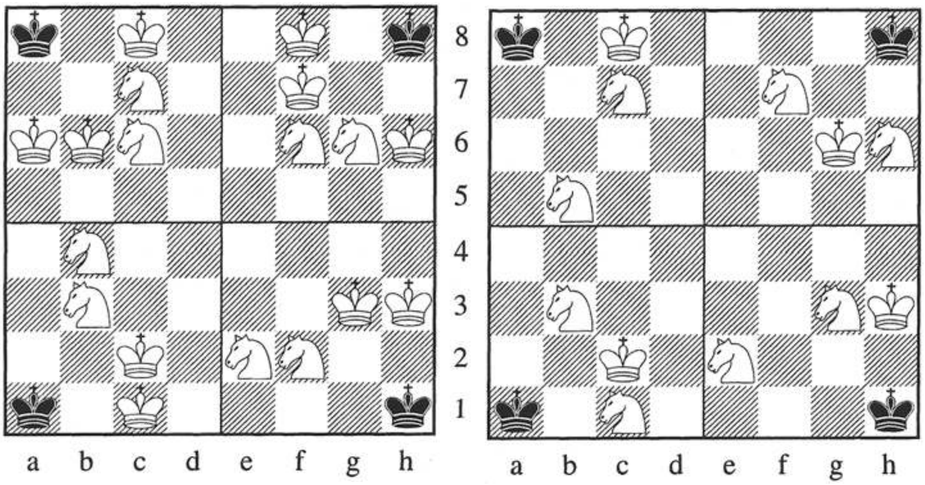

Figure 8. Mate themes in KNNK (corner traps, not enforceable).

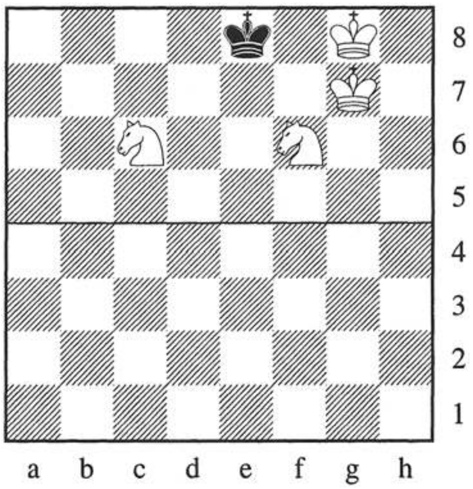

Figure 9. Another mate theme in KNNK (edge trap, not enforceable).

Despite the considerable material advantage of the strong side, the subgame KNNK of KNNKB and KNNKN is generally drawn too (Thompson, 1991; The Editors, 1992; Nalimov *et al.*, 2000, 2001). Again, there are no enforceable checkmates in KNNK but the number and variety of mating themes and direct mates are much higher here than in the materially balanced subgames KBKN and KNKN. Figure 8 visualizes whole sets of mate themes in KNNK by showing several alternative locations of the strong King together with a fixed placement of the two Knights and the weak King in the same single quadrant. The checkmate positions in each such set differ solely by the location of the strong King. Like the mate themes of KBKN and KNKN, these KNNK checkmates all involve trapping the weak King in a corner of the board. In addition to the corner traps, there is another special mate theme in KNNK that works by trapping the weak King on the edge of the board away from the corner. Figure 9 depicts this additional mate theme for one possible location of the weak King on the edge in the upper half of the board. The theme remains valid when shifting it to the left or right within the bounds of the board, of course. Although not being actively enforceable within KNNK, all the mate themes of Figures 8 and 9 exemplify potential wins by the strong sides in KNNKB and KNNKN. Therefore, the static recognition rules must also cover them properly.

5. Checkmates in KNNKB and KNNKN

Those readers who are still not convinced that KNNKB and KNNKN positions deserve better than being scored as some kind of draw might finally reconsider after taking a look at the following numbers found in Nalimov's tablebase summary files (Nalimov *et al.*, 2000, 2001). Roughly 10,000 position templates in KNNKB and 40,000 position templates in KNNKN are won for the KNN side. The vast majority of them are non-direct forced wins requiring several moves to mate. Including symmetries, the real numbers of won positions for the KNN side amount to 4x – 8x as many: i.e., 40,000 to 80,000 in KNNKB and 160,000 to 320,000 in KNNKN.

Compared with the 300 to 600 forced wins of KNNK (all direct mates), there are orders of magnitude more forced wins in KNNKB and KNNKN where the weak side features a minor piece in addition to the King. Hence, the KNN side is actually better at enforcing checkmate if the opponent defends itself with more material than just a lone King. As counter-intuitive as this might seem at first glance, it is quite well-known and not too hard to understand because the additional piece prevents stalemates and may even block an escape route of the weak King.[1] However, the added material also enables the weak side to mate the opponent in some non-enforceable circumstances brought about by bad play of the strong side. Consequently, KNNKB and KNNKN contain some positions where the weak side wins and the KNN side is mated.

5.1 Weak Side Wins

The mate themes of all subgames still apply in KNNKB and KNNKN as well, with the excess piece (a strong Knight) located anywhere else on the

[1]The famous forced wins in $\leq$ 7 moves with a single Knight against a Pawn in KNKP(a,h) exploit the very same strategy.

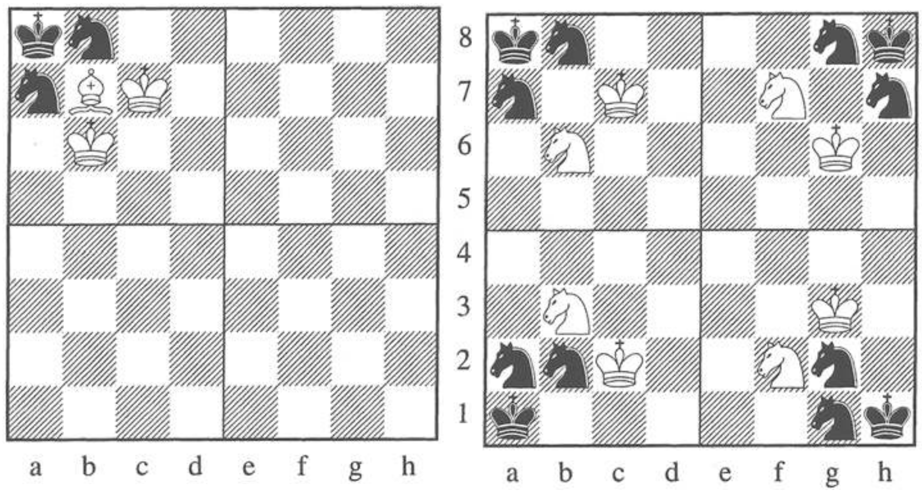
Figure 10. Additional mate themes for weak side in KNNK[B,N] (not enforceable).

board in legal fashion. If the second Knight of the KNN side also resides directly beside the strong King trapped in a corner, the noteworthy additional mate themes shown in Figure 10 arise. The static recognition rules must take all such possibilities into full account.

5.2 KNN Side Wins

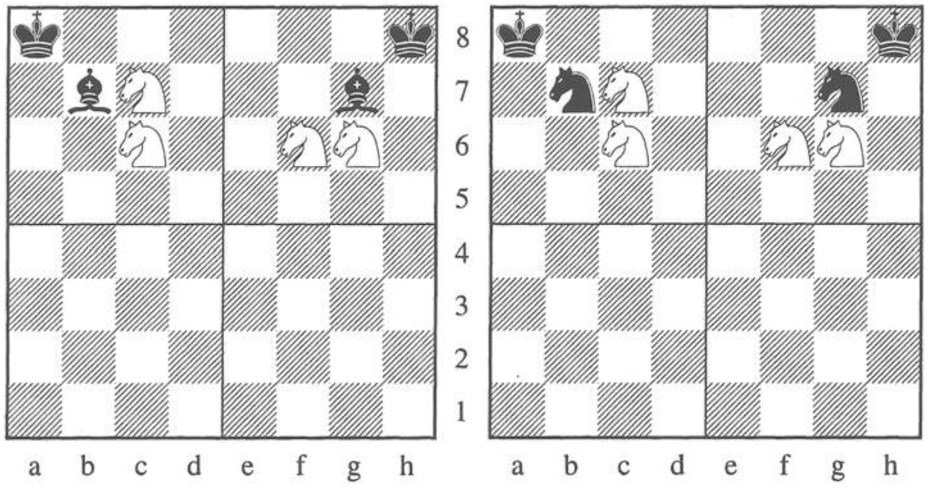
Figure 11. Mate themes without King support in KNNK[B,N] (not enforceable).

As before, the mate themes of all subgames apply in KNNKB and KNNKN too. On top of these, the KNN side may now mate the opponent even without any support of its own King. Figure 11 presents the according NN-checkmates which are quite exceptional and not enforceable. Other additional mate themes of KNNKB involving the full set of 5 pieces on the board are shown in Figures 12 and 13 (corner traps) and Figure 14 (edge traps). This overview of positions with the KNN side winning is by no means exhaustive. But due to space limitations, the remaining positions won by the strong side in KNNKB cannot be shown here. Unfortunately, the very same holds for all additional KNNKN mate themes and positions won by the strong side there. Nevertheless, the static recognition rules must of course cover them all in a suitable way too.

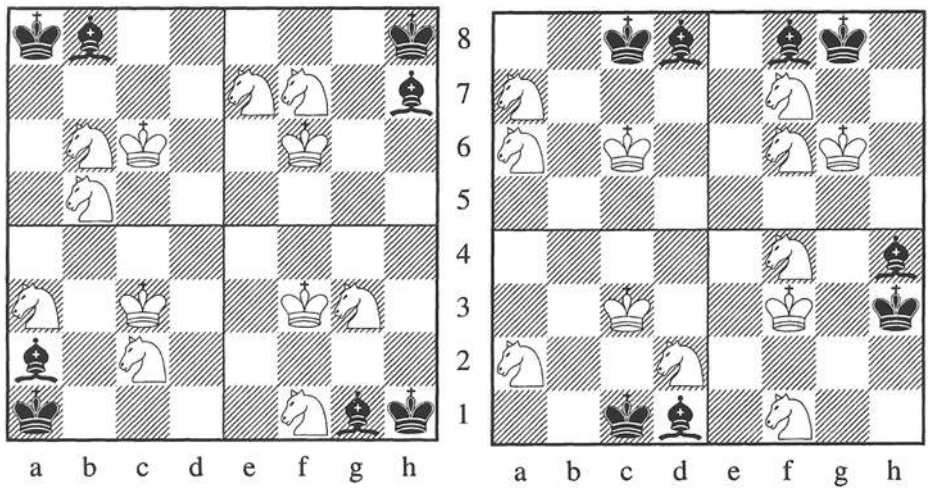

Figure 12. Additional mate themes for strong side in KNNKB (I).

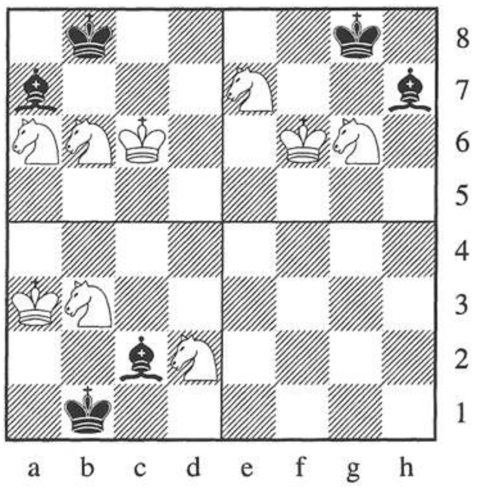

Figure 13. Additional mate themes for strong side in KNNKB (II).

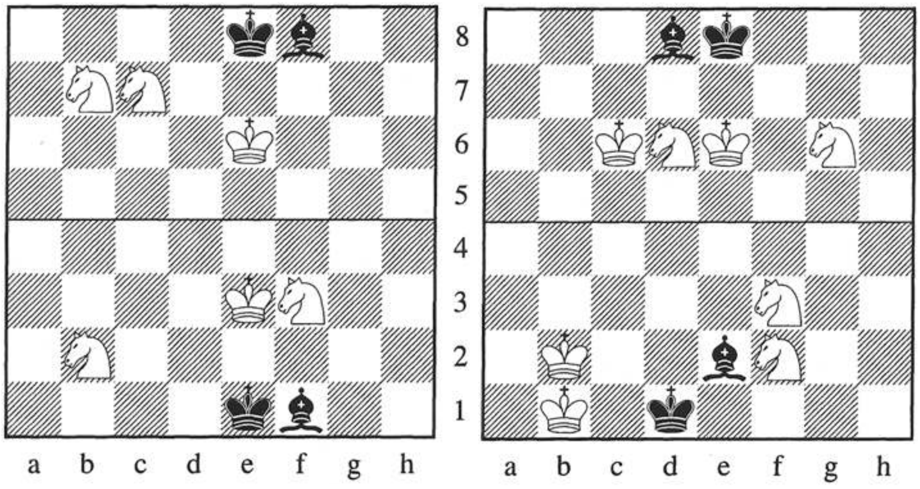
Figure 14. Additional mate themes for strong side in KNNKB (III).

6. Static Recognition Rules

The preceding sections on checkmates in KNNKB and KNNKN plus all their subgames (KBKN, KNKN, KNNK) argue that all possible mate themes in these endgames involve trapping the enemy King in either the corner or on the edge of the board. The omniscient endgame databases confirm this notion but their exhaustive querying also reveals some forced wins for the KNN side in KNNKN where the weak King resides on one of the "extended corner" squares of the board, namely b2, b7, g2, and g7. Figure 15 shows such a position which arises from the forced win in 7 moves of Figure 6 after 1. Na6+ Kb7.

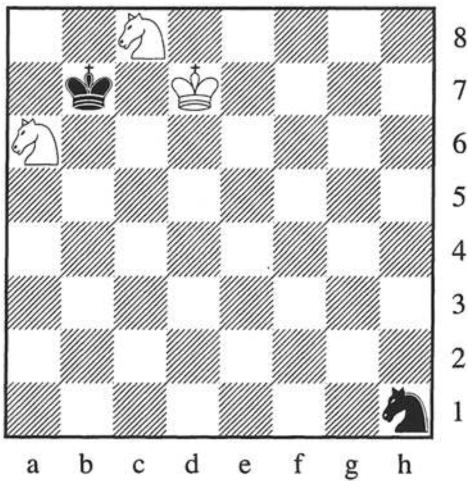
Figure 15. White mates in 6 moves – see Figure 6 after 1. Na6+ Kb7.

The ensuing motif how to enforce the final checkmate does not work against a defending Bishop. Hence, there are no forced wins with the weak King located on "extended corner" squares in KNNKB.

Strong-Win Potential. The winning chances of the strong crucially hinge on its ability to keep the weak King trapped on the edge and, in case of KNNKN, the "extended corner squares of the board. Success in doing so is quite tedious to determine exactly because of possible checks, attacks on the strong Knights, and even pins by the weak Bishop in KNNKB.

NN-Mate Rule. If the weak King is located in a corner of the board with its Bishop or Knight directly beside it on an "extended corner" square and a strong Knight trapping it from the next square on the long diagonal, then the special mate themes of Figure 11 loom. They do not require any direct support by the strong King. So, the position is a guaranteed win for the KNN side if the other strong Knight already gives a check or is on move and able to deliver a direct check (in KNNKN this holds even if the strong King is currently in check itself). Otherwise, the position is drawn in KNNKB if the weak side is on move or the strong side is in check because then the weak Bishop can capture a Knight (see Figure 11).

Weak-Draw Rule. If the weak King does not reside on the edge of the board and not on any "extended corner" square in case of KNNKN either, then the weak side at least draws. The same holds if the weak side is on move and the weak King can directly step off the edge and the "extended corner" in case of KNNKN. If the distance between the two Kings exceeds 4 steps measured in squares on the board, the position is drawn too as discovered by exhaustive analyses of the endgame databases KNNKB and KNNKN. Depending on the side-to-move and whether it is a KNNKB or KNNKN position, the distances between the two Kings triggering a draw are even smaller (see recognition algorithm below for more details).

Weak-Win Rule. If the strong King is located in a corner of the board with at least one of its Knights directly beside it on the edge of the board and the weak King covers the "extended corner" square next to the strong King, then the weak side might even win whereas the strong side at most draws. If so and the strong side is on move but not checkmated, then the position is drawn. If so and the weak side is on move but cannot directly check and mate the opponent, then the position is drawn as well.

7. Static Recognition Algorithm

Constant and Type Declarations

```
TYPE boardstate = ... ;           /* state of a given position on chess board */
TYPE score      = ... ;                               /* range of valid scores */
TYPE side       = ENUM {black, white};
TYPE square     = ENUM {a1, ..., h1, ..., a8, ..., h8};

const SET OF square: corner  = {a1, h1, a8, h8};
const SET OF square: edge    = {a1, ..., h1, a2, h2, ..., a7, h7, a8, ..., h8};
const SET OF square: xcorner = {b2, g2, b7, g7};
```

KNNK[B,N] Recognition Function

```
FUNC score knn_k_b_n_recog(const boardstate: pos; const side: strong, weak) {

    const        square: strong_k      = k_sqr(strong, pos);
    const SET OF square: strong_k_area = k_attck(strong_k);
    const SET OF square: strong_nn     = n_sqrs(strong, pos);
    const SET OF square: weak_b_n      = b_sqrs(weak, pos) + n_sqrs(weak, pos);
    const        square: weak_k        = k_sqr(weak, pos);
    const SET OF square: weak_k_area   = k_attck(weak_k);
    const        square: weak_minor    = ANYELEM(weak_b_n);

    /***** WEAK-WIN PART *****/

    IF (strong_k IN corner)    /* strong K trapped by own Ns and weak K with */
        && EMPTY(strong_k_area - strong_nn - weak_k_area)      /* no escape */
    {
        const SET OF square: b_mates = xcorner * strong_k_area;     /* target */
        const SET OF square: n_mates = n_attck(strong_K);      /* squares for */
                                                   /* B, N to mate strong K */
        IF (side_to_move(pos) == strong)
            && ((is_knnkb(pos) && !EMPTY(weak_b_n * b_mates))
                || (is_knnkn(pos) && !EMPTY(weak_b_n * n_mates)))
           RETURN stm_mated_score(pos)
        ELSE IF ((side_to_move(pos) == weak)   /* weak side on move may mate */
             && EMPTY(n_attck(weak_k) * strong_nn)        /* if not in check */
             && ((is_knnkb(pos) && !EMPTY(b_attck(weak_minor, pos) * b_mates))
              || (is_knnkn(pos) && !EMPTY(n_attck(weak_minor) * n_mates))))
           RETURN stm_mates_score(pos);
        ELSE
           RETURN draw_score(pos);          /* otherwise, position is drawn */
    }
    /***** WEAK-DRAW PART (I) *****/
                                                /* drawn if weak K not on edge */
    IF !(weak_k IN edge) && !((weak_k IN xcorner) && is_knnkn(pos))   /* and */
        RETURN draw_score(pos);         /* not on "extended corner" in KNNKN */

    /***** NN-MATE PART *****/

    IF (weak_k IN corner)         /* weak K in corner trapped by own B, N on */
        && !EMPTY(weak_k_area * xcorner * weak_b_n)     /* "extended corner" */
        && !EMPTY(strong_nn * {c3, f3, c6, f6} * k_attck(weak_minor))
    {                                /* and by strong N in diagonal opposition */
       IF !EMPTY(strong_nn * n_attck(weak_k))    /* weak K also in check by */
          RETURN stm_mated_score(pos);       /* 2nd strong N ==> checkmate! */

       IF is_knnkb(pos) && ((side_to_move(pos) == weak) || (strong_k IN
                  (k_attck(weak_minor) * {c1, f1, a3, h3, a6, h6, c8, f8})))
          RETURN draw_score(pos);    /* drawn in KNNKB if weak side on move */
                                               /* or strong side in check */
       IF (side_to_move(pos) == strong) && !EMPTY(n_attck(weak_k) * n_attck(
             ANYELEM(strong_nn - {c3, f3, c6, f6} * k_attck(weak_minor))))
         /* strong side on move and other strong N ready to deliver mate */
          RETURN  (is_knnkb(pos) || !(strong_k IN n_attck(weak_minor)))
              ? stm_mates_score(pos) : stm_mates_next_score(pos);

       RETURN rcg_fail_score(pos);        /* weak side on move in KNNKN ==> */
    }                   /* may still draw (unwind the trap by removal of N) */

    /***** WEAK-DRAW PART (II) *****/
                                          /* drawn if K distance > 4 steps */
    IF sqr_dist(strong_k, weak_k) > 4 RETURN draw_score(pos);

    IF side_to_move(pos) == weak
    {                                 /*  calculate escape squares of weak K */
       const SET OF square: esc_area = weak_k_area - weak_b_n - strong_k_area
            - n_attck(FIRSTELEM(strong_nn)) - n_attck(LASTELEM(strong_nn));

       IF !EMPTY(esc_area - edge - (is_knnkn(pos) ? xcorner))     /* weak K */
          RETURN draw_score(pos);          /* can escape from trap ==> draw */
    }
    /***** STRONG-WIN PART *****/

    RETURN rcg_fail_score(pos);    /* handle tricky issues by further search */
}                                   /* and trigger an extension in this line */
```

7.1 Algorithm Description

Auxiliary Functions. The recognition algorithm relies on several auxiliary functions not specified in detail here. There are a number of routines to access and query the current state of the chess board passed in the parameter pos of type boardstate: k_sqr returns the King location of the desired side; b_sqrs and n_sqrs return the locations of all Bishops and Knights respectively for the desired side; is_knnkn and is_knnkb identify the exact material balance; and side_to_move returns the side on move in the given position. Another group of auxiliary functions handles the encoding of recognizer failures, checkmates, draws, and mates in this or the next move after it into valid scores: rcg_fail_score, stm_mated_score, draw_score, stm_mates_score, and stm_mates_next_score. The numerical function sqr_dist returns the distance between two squares on the board as measured in single-square steps that a King needs in moves on an empty board to travel from one to the other. Last but not least, the algorithm requires support for the calculation of sets of squares attacked by Bishops, Kings, and Knights located anywhere on the board. The functions `b_attck`, `k_attck`, and `n_attck` perform the according attack generations for B, K, and N respectively. The sliding coverage of Bishops along their diagonals specifically depends on the full board state, whereas Kings and Knights always attack the same sets of squares from a given location regardless of any other pieces.

Constants and Types. The constant sets `corner`, `edge`, and `xcorner` capture the important corner, edge, and "extended corner" squares of the chess board. The enumeration type `side` contains just two items: `black` and `white`. The enumeration type `square` covers all board squares denoted by the 64 items `a1`, ..., `h1`, ... `a8`, ..., `h8`. The anonymous types `boardstate` and `score` represent the full states of chess positions and scoring values respectively.

Pattern Recognition. The algorithm applies basic set operations on sets of squares to achieve location-indepedent pattern recognition. As an example take the core NN-mate pattern of the weak King in any corner, the weak Bishop or Knight directly beside it on the corresponding "extended corner" square, and one of the strong two Knights diagonally beside the weak minor piece as depicted in Figure 11. The membership test `weak_k IN corner` assures that the weak King resides in a corner. Then, the intersection `weak_k_area * xcorner * weak_b_n` gives the set of "extended corner" squares with a weak Bishop or Knight directly beside the weak King. If the set is not empty, it contains the square of the weak minor piece as a single element and the second pattern condition holds. Finally, intersecting `k_attck(weak_minor) * {c3, f3, c6, f6} * strong_nn` computes the set of squares with strong Knights directly and inwardly beside the weak minor. If this set is not empty, the full core pattern is identified independent of the specific corner square the weak King is located on.

Weak-Win Part. The recognition starts with the exceptional wins by the weak side where the strong King is trapped in a corner by at least one of its Knights and the weak King. Depending on which side is on move and whether the weak minor piece can actually deliver a checkmate, the algorithm returns mate or mated scores and a draw score otherwise. A clever trick used here to determine if a single square is attacked by any piece from a set of like pieces works as follows: call the specific attack function of the given piece type with the very square in question as the location parameter, then intersect the resulting attack squares with the original set of like pieces $\rightarrow$ if and only if the intersection is not empty, the square in question is under attack by some piece from the set of like ones. This scheme excels at check detection. The term `EMPTY(n_attck(weak_k) * strong_nn)`, for instance, assures that the weak King is not in check by any of the strong Knights.

Weak-Draw Part (I). This straightforward section detects draws by the rule that the weak King is not on the edge of the board and not on any "extended corner" squares in KNNKN either.

NN-Mate Part. The paragraph on pattern recognition above already discussed the core NN-mate pattern and its recognition in detail. After establishing that the core NN-mate pattern applies, the algorithm tests for checkmate by the second strong Knight attacking the weak King, for draws in KNNKB with the weak side on move or the strong side in check, and for forced mates by the strong side with the second strong Knight ready to deliver the final check. Otherwise, the weak side is on move in KNNKN and may still draw by removing the weak Knight from the "extended corner" square, thus unwinding the trap. The static recognizer intentionally fails at this point in order to resolve the resulting complications of checks and Knight forks by further search.

Weak-Draw Part (II). First, the algorithm detects draws by the rule "Kings more than 4 steps apart". Then, the next draw detection deals with the case that the weak side is on move and may directly step off the edge and the "extended corner" in case of KNNKN. The available escape squares of the weak King are those squares around it not blocked by the weak minor piece and not attacked by either the strong King or its Knights. If the set difference of these escape squares and the edge of the board (plus the "extended corner" squares in case of KNNKN) is not empty, then the weak King directly escapes from the trap and the position is drawn.

Strong-Win Part. Whenever no obvious drawing rule for the weak side applies, the static recognizer fails. In case of KNNKN, the weak King still seems to be trapped on the edge of the board or the "extended corner" squares. Further search then resolves the tricky issues of possible checks, attacks on the strong Knights, and pins of the weak Bishop in KNNKB. In general, such explicitly intended failures of static recognizers should trigger search extensions in the current line. If so desired, more ambitious analyses of the piece constellation and attack relations on the board aiming for an even better identification of real wins in KNNKB and KNNKN may easily be added in front of the fail-value return at the end.

7.2 Algorithmic Complexity

The recognition algorithm heavily depends on sets of squares and basic operations on them: set difference, element count, emptiness, intersection, membership, member selection, and union. Other important auxiliary functions are those for attack generation and access to the data structure holding the full state of the current board position.

Sets of Squares. There are 64 squares on a chess board. Hence, the best way to handle sets of squares is by means of a standard bit-vector representation with exactly 64 bits (one for each square) where square i is in the set if and only if the i-th bit of the vector is 1. Thus, sets of squares nicely map to 64-bit unsigned integers which are natural data types of modern CPUs. In computer chess such 64-bit values are also known as "bitboards".

Basic Set Operations. For sets represented as bit vectors, all basic set operations map to simple constant-time computations involving unsigned 64-bit data: *difference* $\rightarrow$ bit-wise `AND` complement, *element count* $\rightarrow$ count bits (a.k.a. population count), *emptiness* $\rightarrow$ compare with 0, *intersection* $\rightarrow$ bit-wise `AND`, *membership* $\rightarrow$ test bit, *selection* $\rightarrow$ find bit, and *union* $\rightarrow$ bit-wise `OR`. Most of these computations actually finish within a single clock cycle on modern CPUs. The 64-bit unsigned integer value 0 represents the empty set and comparisons for set equality are done by standard tests comparing 64-bit unsigned integer values.

Attack Generation. The squares attacked by Kings and Knights depend on their specific locations only, regardless of the placement of any other pieces. Straightforward table lookups indexed by square numbers suffice to perform the according attack calculations `k_attck` and `n_attck`. Bishops, on the other hand, are sliding pieces that depend on the full board constellation to determine the exact extent of their attack coverage. Even if implemented by looping over squares in the four diagonal directions, the respective attack calculations of `b_attck` are constant-time bound because their are at most 13 squares to traverse (7 on the middle diagonal of the board and another 6 on one next to the middle). Moreover, so-called "rotated bitboards" (Hyatt, 1999; Heinz, 1997, 2000) enable the full Bishop attack calculations to be done by a few table lookups.

Remaining Auxiliary Functions. Except for attack generation, the auxiliary functions either encapsulate simple access protocols to the data structure carrying the current state of the chess board or they perform equally simple score value encodings. All these computations are constant-time bound and take only a few clock cycles to finish on modern CPUs. The same holds for `sqr_dist`, an auxiliary function not covered up to now:

```
sqr_dist(x,y) = MAX( ABS(VAL(x)/8 - VAL(y)/8), ABS(VAL(x)%8 - VAL(y)%8) ).
```

All in all, the recognition algorithm contains only constant-time bound computations and no loops. Hence, it is of constant time complexity in **O(1)**. As the average and longest execution paths through the algorithm are short and most of the calculations actually finish within a few clock cycles on modern CPUs, the whole algorithm also features good efficiency in practice where acceptably small constants cap its average and worst-case execution times.

8. Conclusion and Future Work

Hundreds of thousands of positions in KNNKB and KNNKN are won for the KNN side. Tricky mate themes occur more frequently and require more complicated handling in these two endgames than common wisdom makes people think. In fact, they are not trivial at all! This paper may very well be the first ever to present a rule-based static recognition algorithm for any complete non-trivial 5-piece endgame because the fine works by Herschberg *et al.* (1989), van Tiggelen and van den Herik (1991), and van Tiggelen (1991, 1998) consider only the subset KNNKP(h) of the full KNNKP endgame.

All mate themes and rules were developed a-priori by hand. Then, later on, their validity was checked against omniscient endgame databases a-posteriori. In particular, the "trapped King" feature seems very important and powerful for endgames in general and is probably good for static recognition in other endgames as well. Such trapping and the number of escape squares for each King could possibly be used as a crucial position feature and input parameter for machine-learning algorithms that try to extract useful knowledge from endgame databases automatically. The trap patterns look interesting for chess problem composers, too, who have certainly discovered them on their own already.

In the future, I like to use the KNNKB and KNNKN recognition rules as a foundation to statically detect possible draws and "mates in X" in other positions not covered by endgame databases directly (e.g., additional material might not save Black in Figure 3). Moreover, one can still extend the current algorithm to include better static mate detection and further knowledge about enforceable "mate in X" positions. It is also possible to down-scale and specifically adapt the algorithm for the subgames KBKN, KNKN, KNNK, and the endgame KBKB.

References

Bain, M. and Srinivasan, A. (1995). Inductive Logic Programming with Large-Scale Unstructured Data. *Machine Intelligence 14*, K. Furukawa, D. Michie, and S. Muggleton (eds.), pp. 233–267, Oxford University Press.

Bain, M. (1994). *Learning Logical Exceptions in Chess*. Ph.D. Thesis, University of Strathclyde [printed as Thesis 7866, Dept. of Statistics and Modelling Science, University of Strathclyde].

Bain, M. and Muggleton, S. (1994). Learning Optimal Chess Strategies. *Machine Intelligence 13*, K. Furukawa, D. Michie, and S. Muggleton (eds.), pp. 291–309, Oxford University Press.

Barth, W. (1995). Combining Knowledge and Search to Yield Infallible Endgame Programs. *ICCA Journal*, Vol. 18, No. 3, pp. 149–159.

Barth, W. and Barth, S. (1992). Validating a Range of Endgame Programs. *ICCA Journal*, Vol. 15, No. 3, pp. 132–139.

Beal, D.F. and Clarke, M.R.B. (1980). The Construction of Economical and Correct Algorithms for King and Pawn against King. *Advances in Computer Chess 2*, M.R.B. Clarke (ed.), pp. 1–30, Edinburgh University Press.

Beal, D.F. (1977). *Discriminating Wins from Draws in King+Pawn versus King Chess Endgames.* Unpublished Report, Queen Mary College.

Bramer, M.A. (1982). Refinement of Correct Strategies for the Endgame in Chess. *SIGART Newsletter*, Vol. 80, pp. 155–163 [reprinted in *Computer Game-Playing: Theory and Practice*, M.A. Bramer (ed.), pp. 106-124, 1983, Ellis Horwood].

Bramer, M.A. (1980a). Correct and Optimal Strategies in Game-Playing Programs. *Computer Journal*, Vol. 24, No. 4, pp. 347–352.

Bramer, M.A. (1980b). An Optimal Algorithm for King and Pawn against King using Pattern Knowledge. *Advances in Computer Chess 2*, M.R.B. Clarke (ed.), pp. 82–91, Edinburgh University Press.

Bramer, M.A. and Clarke, M.R.B. (1979). A Model for the Representation of Pattern-Knowledge for the Endgame in Chess. *Intl. Journal of Man-Machine Studies*, Vol. 11, No. 5, pp. 635–649.

Bratko, I. (1982). Knowledge-Based Problem Solving in AL3. *Machine Intelligence 10*, J.E. Hayes, D. Michie, and Y.-H. Pao (eds.), pp. 73–100, Ellis Horwood.

Bratko, I. and Michie, D. (1980). A Representation for Pattern Knowledge in Chess Endgames. *Advances in Computer Chess 2*, M.R.B. Clarke (ed.), pp. 31–56, Edinburgh University Press.

Bratko, I. and Niblett, T. (1979). Conjectures and Refutations in a Framework for Chess Endgame Knowledge. *Expert Systems in the Micro-Electronic Age*, D. Michie (ed.), pp. 83–102, Edinburgh University Press.

Bratko, I., Kopec, D., and Michie, D. (1978). Pattern-Based Representation of Chess End-Game Knowledge. *Computer Journal*, Vol. 21, No. 2, pp. 149–153.

Coplan, K.P. (1998). Synthesis of Chess and Chess-like Endgames by Recursive Optimization. *ICCA Journal*, Vol. 21, No. 3, pp. 169–182.

Heinz, E.A. (2000). *Scalable Search in Computer Chess.* Vieweg.

Heinz, E.A. (1999a). Knowledgeable Encoding and Querying of Endgame Databases. *ICCA Journal*, Vol. 22, No. 2, pp. 81–97.

Heinz, E.A. (1999b). Endgame Databases and Efficient Index Schemes for Chess. *ICCA Journal*, Vol. 22, No. 1, pp. 22–32.

Heinz, E.A. (1998). Efficient Interior-Node Recognition. *ICCA Journal*, Vol. 21, No. 3, pp. 156–167.

Heinz, E.A. (1997). How DARKTHOUGHT Plays Chess. *ICCA Journal*, Vol. 20, No. 3, pp. 166–176.

Herik, H.J., van den (1983). Representation of Experts' Knowledge in a Subdomain of Chess Intelligence. *8th International Joint Conference on Artificial Intelligence*, Proceedings Vol. I, A. Bundy (ed.), pp. 252–255, Kaufmann.

Herik, H.J., van den (1982). Strategy in Chess Endgames. *SIGART Newsletter*, Vol. 80, pp. 145–154 [reprinted in *Computer Game-Playing: Theory and Practice*, M.A. Bramer (ed.), pp. 87–105, 1983, Ellis Horwood].

Herik, H.J., van den (1980). Goal-Directed Search in Chess Endgames. *Delft Progress Report*, Vol. 5, No. 4, pp. 253–279, Delft University of Technology [reprinted in *Computer Chess Compendium*, D.N.L. Levy (ed.), pp. 316–329, Springer, 1989].

Herschberg, I.S., van den Herik, H.J., and Schoo, P.N.A. (1989). Verifying and Codifying Strategies in the KNNKP(h) Endgame. *ICCA Journal*, Vol. 12, No. 3, pp. 144–154 [reprinted in *Computers, Chess, and Cognition*, T.A. Marsland and J. Schaeffer (eds.), pp. 183–196, Springer, 1990].

Hyatt, R.M. (1999). Rotated Bitmaps, a New Twist on an Old Idea. *ICCA Journal*, Vol. 22, No. 4, pp. 213–222.

Kopec, D. and Niblett, T.B. (1980). How Hard is the Play of the King-Rook-King-Knight Ending? *Advances in Computer Chess 2*, M.R.B. Clarke (ed.), pp. 57–81, Edinburgh University Press.

Michalski, R. S. and Negri, P. (1977). An Experiment on Inductive Learning in Chess End Games. *Machine Intelligence 8*, E.W. Elcock and D. Michie (eds.), pp. 175–192, Ellis Horwood.

Muggleton, S. H. (1990). *Inductive Acquisition of Expert Knowledge*. Turing Institute Press.

Muggleton, S. (1988). Inductive Acquisition of Chess Strategies. *Machine Intelligence 11*, J.E. Hayes, D. Michie, and J. Richards (eds.), pp. 375–389, Oxford University Press.

Nalimov, E.V., Haworth, G.McC., and Heinz, E.A. (2001). Space-Efficient Indexing of Endgame Tables for Chess. *Advances in Computer Games 9*, H.J. van den Herik and B. Monien (eds.), pp. 93–113, Institute for Knowledge and Agent Technology, University of Maastricht.

Nalimov, E.V., Haworth, G.McC., and Heinz, E.A. (2000). Space-Efficient Indexing of Chess Endgame Tables. *ICGA Journal*, Vol. 23, No. 3, pp. 148–162.

Negri, P. (1977). Inductive Learning in a Hierarchical Model for Representing Knowledge in Chess End Games. *Machine Intelligence 8*, E.W. Elcock and D. Michie (eds.), pp. 193–204, Ellis Horwood.

Niblett, T.B. (1982). A Provably Correct Advice Strategy for the End-Game of King and Pawn versus King. *Machine Intelligence 10*, J.E. Hayes, D. Michie, and Y.-H. Pao (eds.), pp. 101–120, Ellis Horwood.

Quinlan, J.R. (1983). Learning Efficient Classification Procedures and Their Application to Chess End Games. *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonnell, and T.M. Mitchell (eds.), pp. 463–482, Tioga [reprinted by Springer, 1983].

Quinlan, J.R. (1979). Discovering Rules by Induction from Large Collections of Examples. *Expert Systems in the Micro-Electronic Age*, D. Michie (ed.), pp. 168–201, Edinburgh University Press.

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

Shapiro, A.D. and Michie, D. (1986). A Self-Commenting Facility for Inductively Synthesized Endgame Expertise. *Advances in Computer Chess 4*, D.F. Beal (ed.), pp. 147–165, Pergamon.

Shapiro, A.D. and Niblett, T.B. (1982). Automatic Induction of Classification Rules for a Chess Endgame. *Advances in Computer Chess 3*, M.R.B. Clarke (ed.), pp. 73–92, Pergamon.

Tan, S.T. (1972). *Representation of Knowledge for Very Simple Endings in Chess*. Memorandum MIP-R-98, School of Artificial Intelligence, University of Edinburgh.

The Editors. (1992). Thompson: All about Five Men. *ICCA Journal*, Vol. 15, No. 3, pp. 140–143.

Thompson, K. (1991). Chess Endgames Volume 1. *ICCA Journal*, Vol. 14, No. 1, p. 22.

Tiggelen, A., van (1998). *Heuristic Search Methods in Parameter Space*. Engineering Bureau van Tiggelen.

Tiggelen, A., van (1991). Neural Networks as a Guide to Optimization. *ICCA Journal*, Vol. 14, No. 3, pp. 115–118.

Tiggelen, A., van and van den Herik, H.J. (1991). ALEXS: An Optimization Approach for the Endgame KNNKP(h). *Advances in Computer Chess 6*, D.F. Beal (ed.), pp. 161–177, Ellis Horwood.

Verhoef, T.F. and Wesselius, J.H. (1987). Two-Ply KRKN: Safely Overtaking Quinlan. *ICCA Journal*, Vol. 10, No. 4, pp. 181–190.

Weill, J.-C. (1994). How Hard is the Correct Coding of an Easy Endgame? *Advances in Computer Chess 7*, H.J. van den Herik, I.S. Herschberg, and J.W.H.M. Uiterwijk (eds.), pp. 163–175, University of Limburg.

Zuidema, C. (1974). *Chess: How to Program the Exceptions*? Technical Report IW21/74, Department of Informatics, Mathematical Center Amsterdam.