COMPUTER PROGRAMMING OF KRIEGSPIEL ENDINGS: THE CASE OF KR VS. K

A. Bolognesi and P. Ciancarini
Dipartimento di Scienze dell'Informazione, University of Bologna - Italy
abologne,cianca@cs.unibo.it, http://www.cs.unibo.it/

Abstract Kriegspiel is a chess variant invented to make chess more similar to real warfare. In a Kriegspiel game the players have to deal with incomplete information because they are not informed of their opponent's moves. Each player tries to guess the position of the opponent's pieces as the game progresses by trying moves that can be either legal or illegal with respect to the real situation: a referee accepts the legal moves and rejects the illegal ones. However the latter are most useful to gain insight into the opponent's position. While in the past this game has been popular in research centres such as the RAND Institute, currently it is played mostly over the Internet Chess Club.

The paper describes the rationale and design of a Kriegspiel program to play the ending for King and Rook versus King. Such a kind of ending has been theoretically shown to be won for White, however no programs exist that play the related positions perfectly. We introduce an evaluation function to play these simple Kriegspiel positions, and evaluate it.

Keywords: Kriegspiel

1. Introduction

The game of chess has been widely studied because it is a microcosm that mirrors decision making in real-world situations. However, a basic limit of chess as a field for studying decision making is that decisions by players have nothing to do with uncertainty in the sense in which the term is used in game theory, since the goal and the best strategy for each player can be computed easily and completely.

The game of Kriegspiel is a chess variant invented around 1896 to make chess more similar to real warfare. It involves incomplete information: both the premises and the consequences of a decision are partially unknown, thus it is considered a complex game because of the asymmetry in the knowledge available to the players as the game progresses. In fact, when a player makes an illegal move, from his failure he can infer data that cannot be inferred by his opponent as well. Thus, in general, during a Kriegspiel game each player knows what he knows, but he does not know what his opponent knows.

Kriegspiel is a game interesting in several ways. First, it is based on the same rules as chess, but is has a completely different (and not well studied) theory.

It is a game of imperfect information, such as Poker. However Kriegspiel has no stochastic element, which makes it different from Poker. To play Kriegspiel well we have to use logic and the mathematics of probability.

At the moment there are no programs that play a reasonable Kriegspiel game. On the Internet Chess Club (ICC) a couple of programs are available, which are able to play Kriegspiel, however none of these programs is among the best players (on ICC there are several hundreds of Kriegspiel players, and every day they play hundreds of games).

We recall that a number of papers have studied some aspects of Kriegspiel or Kriegspiel-like games. Below we provided some instances of related work. Ferguson (1992, 1995) analyses the endings KBNK and KBBK, respectively. Ciancarini, DallaLibera, and Maran (1997) describe a rule-based program to play the KPK ending according to some principles of game theory. Sakuta and Iida (2000) describe a program to solve Kriegspiel-like problems in Shogi (Japanese Chess). Bud et al. (2001) describe an approach to the design of a computer player for a subgame of Kriegspiel, called Invisible Chess.

In this paper we explore some issues of the ending KR vs K in Kriegspiel. We aim to design a program that will be a prototype component of a multiagent system able to play Kriegspiel. We describe how we have built such a component, and how we evaluate its behaviour, with the purpose to improve its playing ability.

This paper has the following structure. In Section 2 we describe the basic rules of Kriegspiel, including a study of its main variants. In Section 3 we introduce the theory of the KRvsK ending in Kriegspiel. In Section 4 we describe our search algorithm. In Section 5 we describe our evaluation function: it is specific for this ending, but in our knowledge it is the first time an evaluation function for playing Kriegspiel has been defined. In Section 6 we describe how we use a transposition table to support the search across a tree of metapositions. Finally, in Section 7 we evaluate our approach.

2. The rules

Perhaps the lack of standard rules has been an obstacle to the diffusion of Kriegspiel as a research subject. In fact, there are several different sets of rules, basically classified into two families as Eastern rules (widespread in UK and Eastern US) and Western rules (widespread in Western US) (Pritschard, 1994; Li, 1994). The rules given by J.D. Wilkins in Williams (1950) have been used for years in the RAND Institute. The ICC rules are derived from the RAND rules. However, ICC managers introduced some variants which make the play over the Internet slightly more difficult.

A Kriegspiel player tries a move selected among the set of his pseudo-legal moves, including possible pawn captures. For instance, in the Diagram 1:

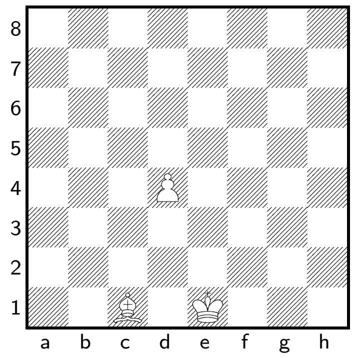


DIAGRAM 1 *: Possible tries.*

possible tries for White are ♗b2,♗a3,♗d2,♗e3, ♗f4,♗g5,♗h6, ♔d1, ♔d2,♔e2,♔f2,♔f1, d5,dc5,de5.

The referee, who knows the list of legal moves for both sides, answers all tries with one of the following six messages.

End of game If the list of legal moves is empty the position is checkmate or stalemate, and the referee announces the corresponding finish.

Move accepted If the try is legal, the referee says "White moved" (or "Yes") and gives no further information. We denote this situation also as "Silent referee", because he gives no useful information.

Illegal move The try selected by White might be illegal on the referee's board. For instance, in the position of Diagram 2 (as it is on the referee's board) White could try ♗h6.

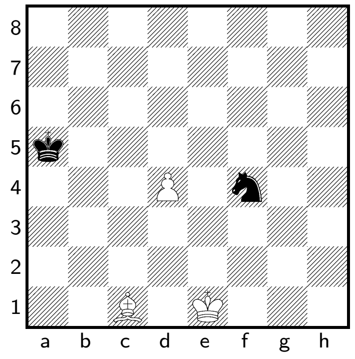


DIAGRAM 2 *: The referee's board.*

The referee says “Illegal move” (or “No”) and White infers that either the diagonal to h6 is obstructed by an enemy piece, or the Bishop is pinned by a black major piece in a1 or b1.

Impossible move The message “Impossible move” is given when a player tries a move outside his set of pseudo-legal moves. In Diagram 2 an impossible try could be ♔e3.

Check If a move is accepted and gives check, the referee announces the check and its direction (row, column, major diagonal, minor diagonal, Knight). In the example, the move ♗d2 gets the answer “Check on major diagonal”.

Capture The referee announces all captures, but he says only on which square the capture takes place, and says nothing about the capturing or captured piece. In the example, the move ♗f4 gets the answer “Capture on f4”.

This list describes the basic messages from the referee. However, in all Kriegspiel versions there is a special treatment of positions where captures by Pawns are possible (We continue numbering of messages).

Are there any? In the original set of rules (Eastern rules) a player could ask before each move *“Are there any?”*, intending “Are there any captures by my Pawns?”. The referee answers “*No*” if no capture is possible, or “*Try!*” if one or more captures are available. With RAND rules the referee announces before each move all possible pawn captures, naming the squares where they can take place. In the set of rules which is used on the Internet Chess Club (Western rules) the referee announces *before each move* how many pawn captures are available.

The Diagram 3 shows the differences among the different set of rules.

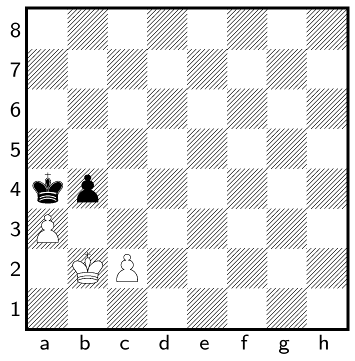


DIAGRAM 3 *: Different pawn capture rules.*

Eastern rules : The referee says: “White to move”. White can choose to ask “Are there any?”; if in the above position White asks the question, the referee says "Try!"; White then has to try at least one capture out of three, namely ab4, cb3, or cd3.

RAND rules : *before* White moves the referee says to both players "*possible pawn capture on b4*". White is not obliged to capture.

Western rules, ICC : *before* White moves the referee says to both players "*possible* one *pawn capture*". White is not obliged to capture.

If now White moves his pawn to c4,

Eastern rules : the referee announces: "*Black moves*";
RAND rules : the referee announces: "*possible pawn captures on a3 and c3*";
Western rules, ICC : the referee announces: "*possible two pawn captures*".

We report these differences for completeness, but we also note that they are not important for endings without Pawns. More important when dealing with endings is the fact in the original form of Kriegspiel no 50-move rule is included; instead on ICC the 50-move rule is enforced.

As a final remark, we note that there are several other forms of Kriegspiel-like games, like Dark Chess, Invisible Chess, Stealth Chess, and others. They are all based on some form of invisibility. We plan to report the features of this family of games in a future paper.

3. KR vs K in Kriegspiel

The ending KR vs K in chess is won in at most 16 moves starting from any position. This chess ending is quite easy to study by brute force, because excluding symmetric positions only 28,000 positions have to be evaluated, as shown in Clarke (1977).

The ending KR vs K in Kriegspiel is also won. However according to H.A.Adamson who published some analysis in the magazine *Chess Amateur* in 1923 and 1926, it can take even 40 moves to give checkmate to Black. More recently, this ending has been studied by Leoncini and Magari (1980) and Boyce (1981). The studies proved that this ending is algorithmically won, i.e., White can force mate against any defense, even the most clairvoyant; there are, instead, several endings (e.g., KP vs K or KBB vs K) which are only probabilistically won, that is Black has a chance to draw (or, equivalently, if the referee suggests Black the right move) (cf. Ferguson 1992, 1995; Ciancarini et al., 1997).

Below we start with developing an algorithm for KRK. Therefore we define the notion metaposition. A *metaposition* is a position describing a set of positions: this can be done graphically. In our case we have diagrams with several black Kings, meaning that its position is uncertain. Subsequently we can evaluate how many KRK metapositions we have to deal with. The number of metapositions for this ending can be approximated by fixing the position of white pieces and considering the number of the ways to choose n BK's positions among the remaining positions. If we assume as a worst case for White ♖a1 and ♔b1, we have 52 possible positions which are not controlled by White. The possible metapositions are then

$$\sum_{1 \leq n \leq 52} \binom{52}{n} = 2^{52} - 1 \qquad (1)$$

For these positions, the reflections of the BK position with respect to the diagonal a1 to h8, as described in Bain (1994), do not decrease the numerical complexity of the problem. So we are not able to study this ending completely by brute force.

Diagram 4 shows a typical ending. This diagram shows a *metaposition*: the double black King means that White is not sure whether the black King is on a8 or on b8. Alas, he has to find the best (most rapid) route to checkmate.

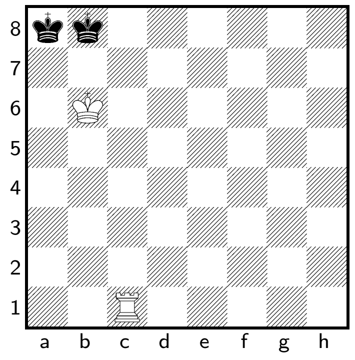


DIAGRAM 4 *: White moves and wins (Adamson, 1923).*

White tries 1.♔c7: (1) if the referee says “No” then White tries 1.♔a6 or 1.♖c2 then mate; (2) if the referee says “Yes” then 2.♖a1 #.

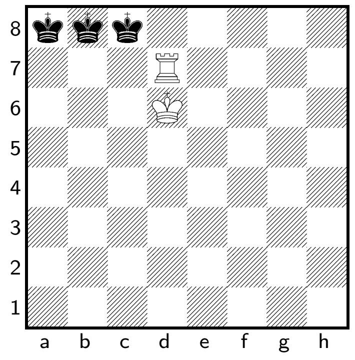


DIAGRAM 5 *: White moves and wins (Adamson 1923).*

In Diagram 5, White tries 1.♔c7:

(1) if the referee says "Yes" then the BK is in a8 and . . . ♔a7 2.♖d6♔a8 3.♖a6#.

(2) if the referee says "No" then White plays 1.♖c7 and:

(2a) with silent referee White identifies the Black King on a8 and the mate is very simple;

(2b) if the referee says "check" then 2.♔d7:

(2b1) if "No" the BK is on d8 then 2.♖c1♔e8 3.♖f1♔d8 4.♖f8#;

(2b2) if "Yes" Black played 1. . . ♔b8 then 2.♔d7♔a8 3.♔c6♔b8 4.♔b6 ♔a8 5.♖c8#.

A general algorithm for any position, in which White knows nothing about the BK whereabouts, is given in Leoncini and Magari (1980). The procedure includes several phases.

In the first phase White has to configure his own pieces as in Diagram 6.

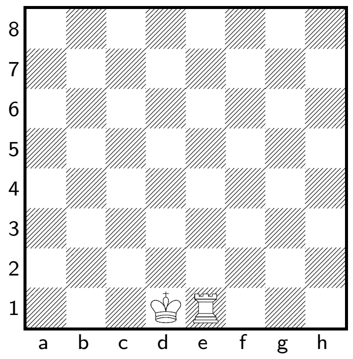


DIAGRAM 6 *: The first phase.*

The second phase consists of looking for the BK by moves like

♔d2,♖e2,♔d3,♖e3,...,♔d8,♖e8:

if the referee never says "check" then the BK is in the left-hand halfboard, otherwise when a check occurs the BK is in the right-hand halfboard, and White's task will be easier to fulfil. We assume the first hypothesis in the metaposition shown in Diagram 7.

Interestingly, Kriegspiel metapositions have been compared by Magari to probability waves as in Quantum Physics. According to such a metaphor, the black King is not a body with a precise position, but a wave, or a set of possibilities. The white King has to destroy such a wave entering it and reducing the freedom of the black King.

In the final position of Diagram 8 White mates with ♖a8#.

If at any time the referee says "Illegal move", White will find the BK earlier, and will be able to use his Rook to restrict further the space available to Black.

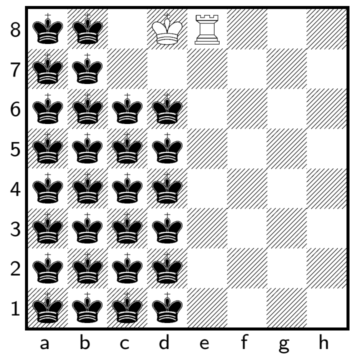


DIAGRAM 7 : *The BK is in the left-hand halfboard.*

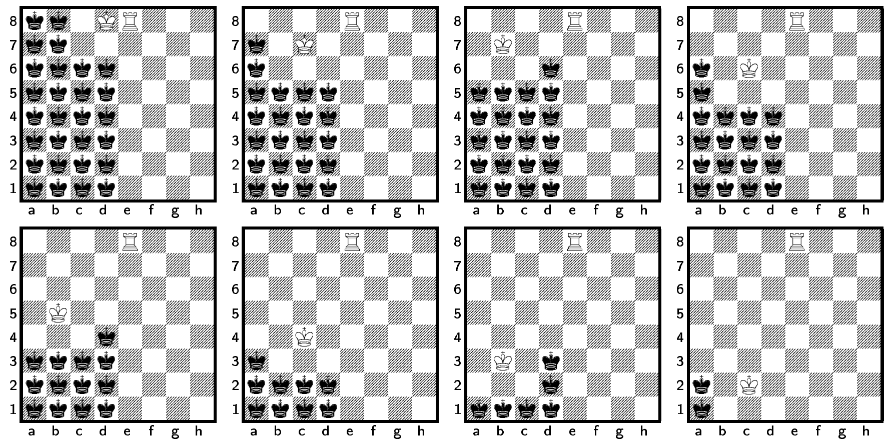


DIAGRAM 8 : *The second phase.*

3.1 Exploiting the referee's answers

In any KRK ending, when White has to try a move, there are three possible situations.

1 The referee's answer is 'silent'. This allows us only to update our reference board cleaning the squares around the WK and along the WR row and column.

2 The WR can check the BK, in that case the player updates his reference board and assumes that the BK possible position is on the WR row or column.

3 A try may be illegal because the WK tries to go in a square which is under attack or because the WR is going across an occupied square.

Assume we are in the situation shown in the leftmost position in Diagram 9. If White moves ♖e3 we distinguish two cases: (1) the referee's answer is 'silent' (second position) or (2) the referee says a check has occurred (third position). If White moves ♔d5 two cases can be outlined too, with a 'silent' answer or with an 'illegal' answer. In the rightmost position we show the result of obtaining the answer 'illegal', since the case in which we get a silent answer is similar to the Rook's one.

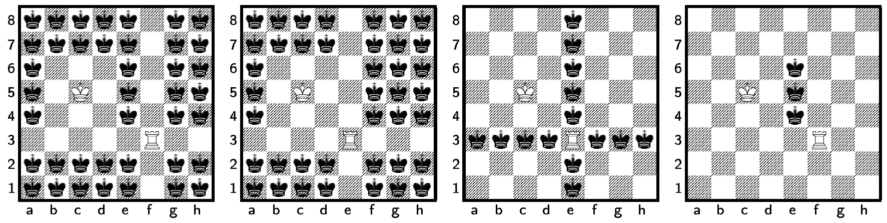


Diagram 9 *: Analysis of the referee's answers.*

4. The search algorithm

When drawing out the search algorithm we are first led to a problem caused by the fact that the move is described only by the referee's answer. This implies that the evaluation of a move can be made only with respect to the referee's answers, using some probabilistic reasoning.

Considering for example a situation where the WK is on c2, the white Rook is on f2, and Black's positions traced in the white player's reference board are on a1, a2, a3, or e3 with a likelihood of 1/4 each. If the WK moves to b1 and receives an 'Illegal' answer that move will be a good move, decreasing the uncertainty (leftmost metaposition in Diagram 10), but if the move receives a 'silent' answer he will achieve a state of danger, where the WR risks to be captured (rightmost position in Diagram 10). So White should not play such a move.

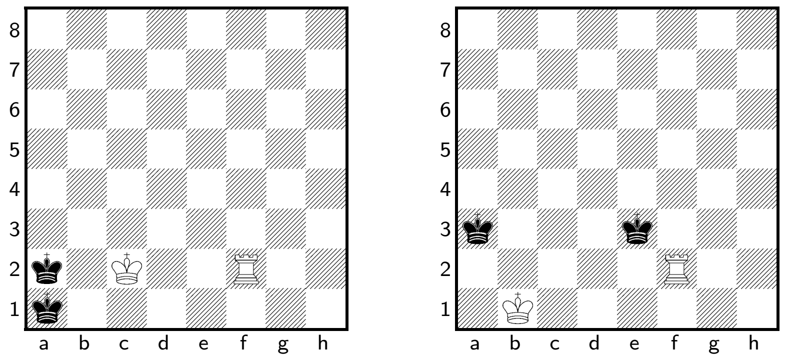


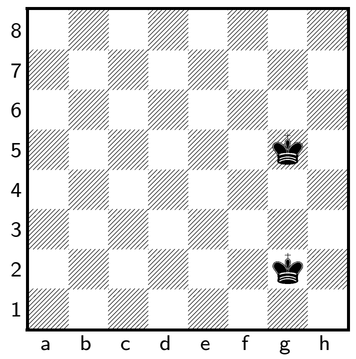
Diagram 10 *: Analysis of metapositions.*

Our solution consists of making a first evaluation during the generation of the pseudo-legal moves considering both cases, either ‘illegal’ or ‘check’ and ‘silent’ answers, and inserting in the possible moves vector the one with the lowest value. In other words the player assumes the worst case and makes available to the search algorithm only one answer per move. In this manner the number of moves we are handling becomes more similar to that of classic chess.

In Kriegspiel the player is in the dark about his opponent’s position so a minmax-like search cannot be executed in this context, unless we find how to represent all possible BK moves. Simply adding a new layer to the algorithm and calculating for each White’s legal move all possible BK positions and for each of those positions repeat the procedure in a minmax way, has an exponential cost that forces us to choose some alternatives.

The way we have chosen to represent the invisible BK on White’s reference board is to define a metaposition which is a set of possible BK squares with the same likelihood. Also, we define an uncertainty index as the count of the possible positions of a metaposition, as in Sakuta and Iida (2000). In some sense White has to play against an unspecified number of black Kings, that can move simultaneously. It is quite simple to define a metamove as a move from one metaposition to another metaposition. Playing a metamove corresponds to playing all the moves for each black position of the metaposition. This trick allows us to use an algorithm like minmax or similar, where we use a metamove generator. We represent a metaposition as an array of possible positions.

One distinctive aspect to note is that we are changing the meaning of search depth. It now refers only to White’s branching factor, since the generation of a metaposition from another involves the introduction of a single edge. Diagram 11 describes the state reached from a reference board where the BK is assumed to be on g2 or g5 with a likelihood of 1/2 each.



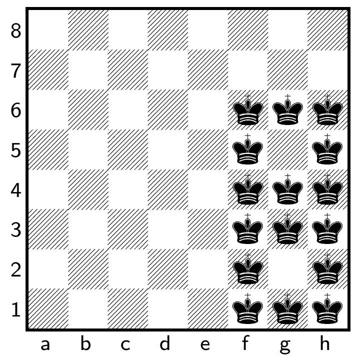


DIAGRAM 11 *: Representing matapositions with likelihood.*

Figure 1 shows the pseudo-code describing the search algorithm.

```
Search Algorithm (int depth) {

    generate the white's legal moves $\Gamma$;

    for each moves $j \in \Gamma$
    {
      if(rook plays the move $j$)
        $j$.value=Min(evaluate($j$,check),evaluate($j$,silent))
      if(king plays the move $j$)
        $j$.value=Min(evaluate($j$,illegal),evaluate($j$,silent))
    }

    for each moves $j \in \Gamma$
    {
        if ($depth! = 1$) {
            makemove($j$);
            generate the opponent's metamove;
            if(!CheckHash(depth$-1$,&value))
              $j$.value += Search(depth-1);
            else
              $j$.value += value;
            unmakemove();
        }
        if ($j$.value $> max$)
            $max$=$j$.value;
    }

    RecordHash(depth,max);

    return $max$;
}
```

Figure 1. The search algorithm.

The algorithm generates all legal white moves and for each resulting position it evaluates both possible referee answers using an evaluation function we will discuss later. So, for each possible position, it is able to distinguish between ‘check’ or ‘illegal’ and ‘silent’ answers and it marks the move with the worst case according to the value returned by the evaluation function. If it has reached the desired search depth it simply returns the max move’s value, otherwise it plays each move and in each metaposition obtained it makes the metamove, then it decrements the depth of search and it recursively calls itself; after that, it retracts the move played and adds to the move’s value the vote which is returned by the recursive call. Finally, it updates the max on that particular search depth.

A move’s value is modified during the path that the algorithm is analysing. If we did not make such updates, a move would obtain a good vote even crossing bad states, where, as an example, we run the risk of losing the Rook. Figure 2 shows the search tree which describes a hypothetical visit. The first evaluation

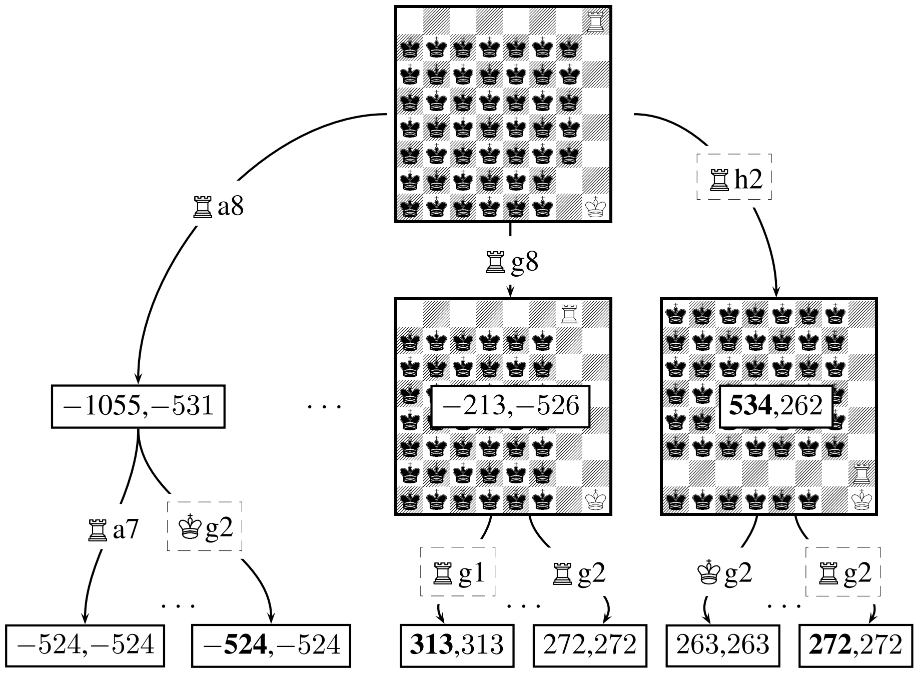


Figure 2. A depth-2 search tree.

is on the right of the node and the updated value of the move is on the left; the bold type indicates the best move.

If we did not add the static evaluation value to the recursive value, at the first depth, moves would respectively obtain −524, 313 and 272; so the second move (which has a bad static value) would be chosen by the search algorithm, while the third move (which has the greatest static value) would be discarded.

5. The evaluation function

We will implement the evaluation knowledge using a weighted linear function, as follows:

$$Evaluate(S) = c_1 f_1(S) + c_2 f_2(S) + ... + c_5 f_5(S) \tag{2}$$

where $c_1, c_2, .., c_5$ are constants and $f_1(S), .., f_5(S)$ are functions which set up the heuristic evaluation.

The first aspect we want to make sure of is to avoid having a position where the WR risks to be captured. For this reason the first boolean function $f_1(S)$ evaluates the possibility that the R ook is under attack, in that case it returns FALSE.

Once we are certain that the Rook is safe, we try to bring the two Kings closer. That means to let the WK patrol the board. Thus the second function

$f_2(S)$ estimates the distance between the WK and all the possible BK positions, by considering the furthest one. The way we calculate the distance is the sum of columns and rows between the WK and the furthest BK. In Diagram 12 we show an example where this distance is 10.

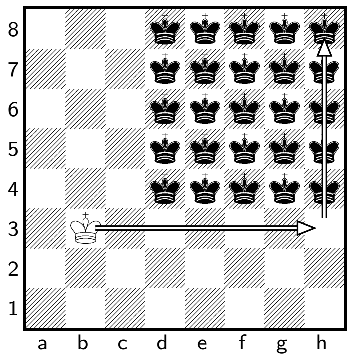


DIAGRAM 12

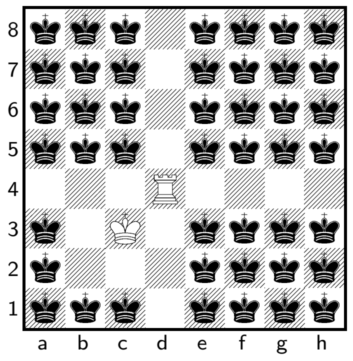


DIAGRAM 13

Let us assume that it is White's turn to move, so the BK certainly is on one of those quadrilateral regions with which the white Rook divides the board. The aim of the white player is therefore to reduce all the regions' areas that contain the black Kings. Again the uncertainty about BK's real position is a problem. The third function $f_3(S)$ estimates which one of the four regions holds the BK and tries to reduce its area. We define it as

$$f_3(S) = EvalArea(S) = c \cdot (a_1 + a_2 + a_3 + a_4) \tag{3}$$

where $c \in \{1, 2, 3, 4\}$ is the value which traces the number of quadrilaterals that possibly contain the opponent's King, and $a_i (i = 1, ..4)$ represents the number BK's possible positions in each quadrilateral. As shown in Diagram 13, in the worst case where uncertainty is maximal, the function's result is 180.

The fourth function $f_4(S)$ is a boolean function which evaluates whether the WR is on the squares around the WK, in that case it increases by one the move's value.

The fifth function $f_5(S)$ considers good moves those that push the BK toward the board's corner. For each positions, where the BK might be, $f_5(S)$ adds to the move's value the correspondent value from the matrix, shown in the Figure 3.

It is useful to note that $f_3(S)$ function calculates a positive value, but in order to evaluate the best move we have to minimize this value.

The same remark on the others functions leads us to the following evaluation function:

$$Evaluate(S) = -420 + 840 \cdot f_1(S) - f_2(S) - f_3(S) + f_4(S) + f_5(S)$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -2 & -4 & -4 & -2 & 0 & 0 \\ 0 & 0 & -4 & -4 & -4 & -4 & 0 & 0 \\ 0 & 0 & -4 & -4 & -4 & -4 & 0 & 0 \\ 0 & 0 & -2 & -4 & -4 & -2 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Figure 3. The simple numerical matrix used by $f_5(S)$.

where $c_1 = 840$ is a weight that gives $f_1(S)$ top priority.

We finally add, after the search algorithm, a function, which catches checkmate cases and consequently avoids playing moves to stalemate states.

6. The transposition table

Since during the search algorithm we would cross states of the board previously analysed, it is interesting to avoid to analysing them a second time. As we have seen the number of metapositions is extremely large and it is impossible to maintain each of them in memory. A natural solution to the comparison between the states involves creation of a signature value, typically using Zobrist (1970) keys.

We define a three-dimensional vector indexed on {KNIGHT, ROOK}, {WHITE, BLACK}, and on the number of squares; then we fill each element with a random 64-bit number. To create a Zobrist key for a metaposition, we set it to zero, then for each piece on the board we add it into the key via the XOR operator. The pieces can be either the Kings or the Rook, and the black King may appear several times.

This technique has the advantage of creating good hash keys, that are not related to the metaposition being keyed. If a single piece is moved, we obtain a value that is completely different. So, these keys do not collide often. Another good peculiarity is that we can manage Zobrist keys incrementally, improving the artificial player's performance, as described by Moreland (2002).

We use the Zobrist keys to implement a transposition table, which is a large hash table that allows us to trace metapositions that we have met during the search. It is impossible to create a big data structure that includes all the metapositions, but in the event of collisions, i.e., when two states are mapped on the same vector's element, we use the Zobrist keys to identify the correct one.

In the Figure 1 we used two functions whose pseudo-code is shown in Figure 4. These two functions are used to store the elements into the transposition table and to load them from it.

```
CheckHash (int depth, int ∗value) {
    hash_element ∗hashpt = &table[(WRB.key % MHE)+MHE];
    if(hashpt-> key == WRB.key)
      if(hashpt-> depth >=depth) {
        value=hashpt->value;
        return TRUE;
      }
    return FALSE;
}
RecordHash (int depth, int max) {
    hash_element ∗hashpt = &table[(WRB.key % MHE)+MHE];
    hashpt-> key = WRB.key;
    if(hashpt==NULL) hashpt-> value = value;
    else if(hashpt-> value >value)
          hashpt-> value = value;
    hashpt-> depth = depth;
}
```

Figure 4. Updating the hash table. WRB means White's Reference Board and MHE is the Max number of the Hash Elements into the table.

The CheckHash function does the load operation. If the element previously stored is the one we have to analyse and it has been examined with a depth grater or equal to the required depth, then the element is loaded from the table.

The RecordHash function does the store operation. It inserts the key and the search depth into the table. When it is not saving a new element, it inserts the value only if this value is smaller than the previous one. That means that the metapositions are randomly divided into clusters.

7. Tests

We have executed a first test on 26,536 initial positions, randomly selected from the 175,168 legal positions of KRK endgame.

Each initial position has the maximum uncertainty on White's reference board, meaning that the BK has the maximum freedom in terms of possible squares.

Black's strategy always consists in playing the move that allows him to go away from the edge of the board.

This test shows that 95.6% of the games are won by White, while 4.4% is lost. In particular 75.9% of this percentage refers to a game that has been stopped for a loop, 24% is draw, and 0.1% is a stalemate, as shown in Table 1. The average number of moves needed to give mate is 36, and the worst game played has been 117 moves long.

Result	*number of games*
mate	25372
loop	883
draw	279
stalemate	2

Table 1. The 26536 games' result, during the first test.

Result	*number of games*
mate	18469
loop	2
draw	122
stalemate	0

Table 2. The games' result during the second test.

In the histogram shown in figure 5 we show the number of the matches won per moves needed, with intervals of 5.

In order to have a comparison, we have executed a second test on all initial positions using the referee's point of view, namely we play this ending using ordinary chess rules and our Kriegspiel evaluation function. During a match, if the game either begins at or goes across some positions previously played, the referee stops it and considers it won or lost, depending on the result of previous games.

In this test 99.5% of the games is won, which is not bad but it shows that our evaluation function is not perfect for ordinary chess. We show the entire results in Table 2 and in Figure 6 we show the sets of won games and the number of moves needed during the second test.

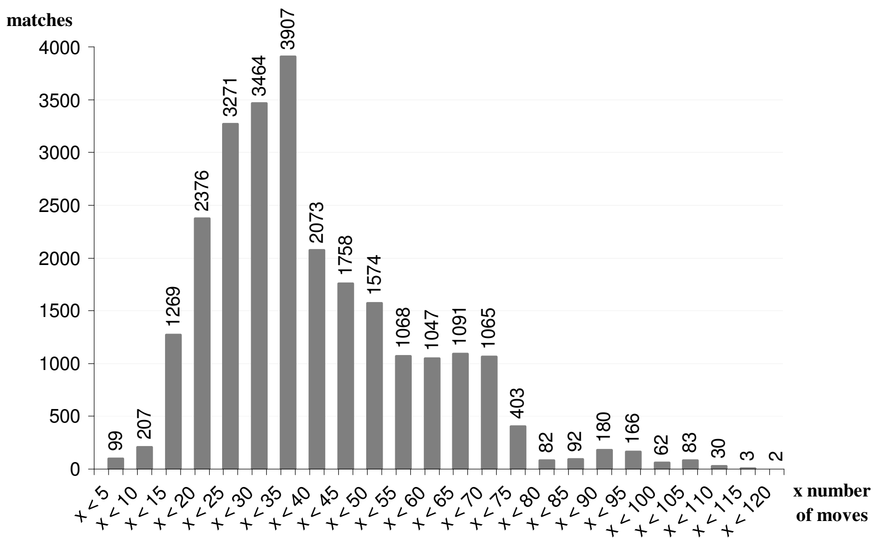


Figure 5. Won games and number of moves (first test).

We show how our program deals with the position in Diagram 5.

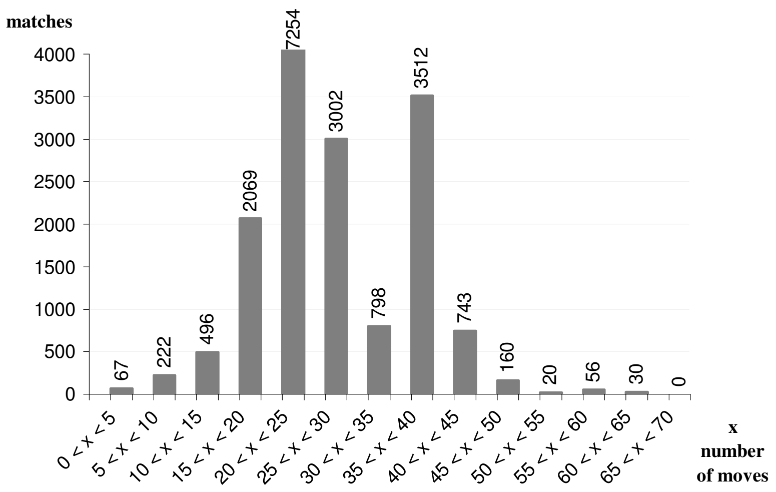


Figure 6. Won games and number of moves (second test).

If we assume that the BK is on a8, the program plays efficiently (in the scores, I means that the referee says *illegal*, C means *check*):

1. d6c7 a8b7{I} a8a7 2. d7d6 a7b6{I} a7b7{I} a7a6{I} a7a8 3. d6a6{C} 1-0 {White mates}

If we assume that the BK is on c8, the program actions are also very effective, as it achieves the checkmate in 2 moves:

1. d6c7{I} d7f7 c8d7{I} c8c7{I} c8b7{I} c8d8 2. f7f8{C} 1-0 {White mates}

Let us assume that the black King is on b8. After playing ♔c7 and receiving an 'illegal' answer, the program plays less precisely ♖f7, and then it takes 23 moves to mate:

1. d6c7{I} d7f7 b8c7{I} b8b7{I} b8a7{I} b8c8 2. f7f8{C} c8d7{I} c8c7{I} c8b7 3. d6c7{I} f8g8 b7c6{I} b7b6 4. d6c6{I} g8g5 b6c5{I} b6c6{I} b6b5{I} b6b7 5. d6c7{I} d6d7 b7c6{I} b7b6 6. d7c6{I} d7d6 b6c5{I} b6c6{I} b6b5{I} b6b7 7. d6c6{I} g5c5 b7c6{I} b7b6 8. d6c7{I} d6c6{I} d6d5 b6c5{I} b6c6{I} b6b5{I} b6b7 9. d5c6{I} d5d6 b7c6{I} b7b6 10. d6c6{I} d6d5 b6c5{I} b6c6{I} b6b5{I} b6b7 11. d5d6 b7c6{I} b7b6 12. d6c6{I} c5c7 b6c5{I} b6c6{I} b6b5 13. d6c5{I} c7c6 b5c4{I} b5c5{I} b5c6{I} b5b4 14. d6c5{I} d6d5 b4c3{I} b4c4{I} b4c5{I} b4b3 15. d5c4{I} c6d6 b3c3 16. d5c4{I} d5c5 c3d4{I} c3d3{I} c3c4{I} c3d2{I} c3c2 17. c5b4 c2c3{I} c2d3{I} c2d2{I} c2b3{I} c2b2 18. b4c3{I} b4a3{I} d6c6 b2c3{I} b2c2{I} b2b3{I} b2c1{I} b2b1 19. b4b3 b1b2{I} b1c2{I} b1c1{I} b1a2{I} b1a1 20. c6c4 a1b2{I} a1b1 21. c4a4 b1b2{I} b1c2{I} b1c1 22. a4d4 c1b2{I} c1c2{I} c1d2{I} c1d1{I} c1b1 23. d4d1{C} 1-0 {White mates}

8. Future Work and Conclusions

In this paper we have described a program which plays a Kriegspiel endgame. We started from a normal chess program and modified it to deal with the uncertainty typical for Kriegspiel playing. In order to evaluate our player, we have played several thousands of games showing that the evaluation function developed is a good basis for further refinements.

We could have implemented a rule-based player based on the procedures reported in Leoncini and Magari (1980) and Boyce (1981). A first problem is that these papers do not prove that their procedures are correct and complete. So, we have no guarantee to obtain a program playing perfectly the KR vs K ending. Moreover, any rule-based solution would have been specialized in KR vs K only. Instead we have adapted our player rather easily to another ending, namely KQ vs K, and now we plan to make similar experiments for other basic endings such as KRBK, KRNK, etc.

References

Bain, M. (1994). *Learning Logical Exceptions in Chess*. PhD thesis, Dept. of Statistics and Modelling Science, University of Strathclyde, Glasgow, Scotland.

Boyce, J. (1981). A Kriegspiel Endgame. In Klarner, D., editor, *The Mathematical Gardner*, pages 28–36. Prindle, Weber & Smith.

Bud, A., Albrecht, D., Nicholson, A., and Zukerman, I. (2001). Playing "Invisible Chess" with Information-Theoretic Advisors. In *Proc. 2001 AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 6–15, California, USA. American Association for Artificial Intelligence.

Ciancarini, P., DallaLibera, F., and Maran, F. (1997). Decision Making under Uncertainty: A Rational Approach to Kriegspiel. In van den Herik, J. and Uiterwijk, J., editors, *Advances in Computer Chess 8*, pages 277–298. University of Limburg, Maastricht, The Netherlands.

Clarke, M. (1977). A Quantitative Study of King and Pawn against King. In Clarke, M., editor, *Proc. First Conf. on Advances in Computer Chess*, pages 108–118, Edinbourgh, Scotland. Edinbourgh University Press.

Ferguson, T. (1992). Mate with Bishop and Knight in Kriegspiel. *Theoretical Computer Science*, 96:389–403.

Ferguson, T. (1995). Mate with two Bishops in Kriegspiel. Technical report, UCLA.

Leoncini, M. and Magari, R. (1980). *Manuale di Scacchi Eterodossi*. Tipografia Senese, Siena.

Li, D. (1994). *Kriegspiel. Chess under Uncertainty*. Premier Publishing.

Moreland, B. (2002). Computer Chess. `http://www.seanet.com/~brucemo/chess.htm`.

Pritchard, D. (1994). *The Encyclopedia of Chess Variants*. Games & Puzzles Publications.

Sakuta, M. and Iida, H. (2000). Solving Kriegspiel-like Problems: Exploiting a Transposition Table. *ICCA Journal*, 23(4):218–229.

Williams, J. D. (1950). Kriegsspiel rules at RAND. (Unpublished manuscript).

Zobrist, A. (1970). A new hashing method with application for game playing. *ICCA Journal*, 13(2):69–73. Reprinted (1990).