Searching over Metapositions in Kriegspiel

Andrea Bolognesi[1] and Paolo Ciancarini[2]

[1] Dipartimento di Scienze Matematiche e Informatiche "Roberto Magari", University of Siena, Italy,
abologne@cs.unibo.it ,
[2] Dipartimento di Scienze dell'Informazione, University of Bologna, Italy
cianca@cs.unibo.it,

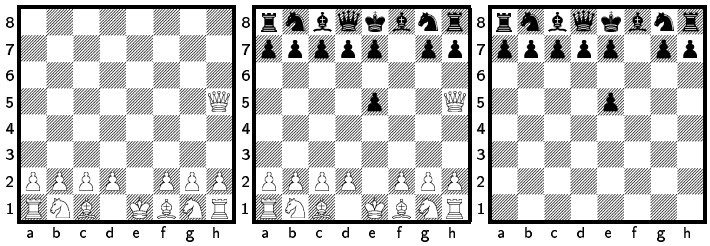
Abstract. Kriegspiel is a Chess variant similar to wargames, in which players have to deal with uncertainty. Kriegspiel increases the difficulty typical of Chess by hiding from each player his opponent's moves. Although it is a two person game it needs a referee, whose task consists in accepting the legal moves and rejecting the illegal ones, with respect to the real situation. Neither player knows the whole history of moves and each player has to guess the state of the game on the basis of messages received from the referee. A player's try may result *legal* or *illegal*, and a legal move may prove to be a *capture* or a *check*.
The paper describes the rationale of a program to play basic endgames of Kriegspiel, where a player has left only the King. These endings have been theoretically studied with rule-based mechanisms, whereas few researches exist on a gametree-based approach.
We show how the branch of game tree can be reduced in order to employ an evaluation function and a search algorithm. Then we deal with game situations dependent on stochastic element and we show how we resolve them during the tree visit.

1 Introduction

Kriegspiel is a Chess variant similar to wargames, in which players have to deal with uncertainty. All Chess rules are valid, but the players are not informed of their opponent's moves. Although it is a two person game, it needs a referee, whose task consists in accepting the legal moves and rejecting the illegal ones, with respect to the real situation. As the game progresses, each player tries to guess the position of his opponent's pieces by trying moves to which the referee can stay *silent*, if the move is legal, or he can announce *"illegal move"*, if the move is illegal. Then the player has to make another try. If the move is legal and it gives check or captures a piece, the referee says *"check"* or *"capture"*, respectively. Moreover, in order to speed up the game if a Pawn can capture an opponent's piece this is also announced.

For instance, this is a simple game (we omit illegal tries):
1.e4 f6. The referee stays silent (both moves are legal).
2.e5. The referee announces: "Black has a Pawn try"
2...fe5. The referee says "Pawn captured on e5".

3.♕h5 The referee announces: "Check on short diagonal"



3. ..g6. The referee stays silent.

4. ♗e2 The referee announces: "Black has a Pawn try"

4. ..gh5. The referee says "piece captured on h5".

5. ♗xh5 The referee announces: "Checkmate"

The three diagrams (left for White, center for referee, right for Black) shown after move three are a typical way to display the partial knowledge that players have about the current state of a Kriegspiel game: a player does not know what his opponent has done up to his turn to move. Therefore Kriegspiel is considered a game of imperfect information. Incidentally, we call the leftmost and rightmost boards *reference boards* for White and Black, respectively.

The design of a Kriegspiel playing program is an interesting problem, because we can adapt most techniques already developed for Computer Chess. However, it remains a problem to adapt to Kriegspiel the game tree search and the evaluation function typical of chess playing programs, because each player is uncertain about the position of his opponent's pieces. In theory it would be possible to build a huge game tree taking into account all possible positions compatible with past announcements from the referee. In practice this is an impossible task, because the complete game tree of Kriegspiel is much larger than in the case of Chess.

In this article we propose a way to reduce the game tree which leads to a representation through metapositions instead of normal chess positions. In order to simplify our task, in the next sections we will consider simple endings, ie. Black having only his King and White having a King and a Rook (in the rook ending or ♔♖♚), a King and a Queen (♔♕♚), a King and two Bishops (♔♗♗♚), and a King and a Pawn (♔♙♚).

These endings are simple but all are quite difficult to play under uncertainty about the position of the opponent King. We have developed algorithmic solutions, which solve a given ending in all positions, however we will have to distinguish between positions without a stochastic element, that are states where it is possible to deterministically find the best move to play among the possible ones, and positions with a stochastic element, that are states from which a player may reach several equivalent metapositions through different moves. We will deal with the latter by randomly choosing one of the equivalent moves.

In Section 3 we start describing how we represent uncertainty using metapositions and we show the adjustments done on the game tree to have a deterministic search, then we deal with the case of search including moves randomly chosen. In

section 4 we propose the evaluation function for basic endgames and we describe the search algorithm which use the evaluation function. In this section we deal with several endings (including ♔♖♚, ♔♕♚, ♔♗♗♚, ♔♙♚) and we show some examples of games played by the program.

2 Related works

Although it is a fascinating game, played by several hundreds of people every day on the Internet Chess Club, only a small number of papers have studied some aspects of Kriegspiel or Kriegspiel-like games. Below we provide some instances of related work.

Boyce proposed a procedure to solve the ♔♖♚ ending, that we have implemented to be able to evaluate our algorithm [2] . Ferguson analysed the endings ♔♗♘♚ ([5]) and ♔♗♗♚ ([6]), respectively. Ciancarini, Dalla Libera and Maran ([4]) described a rule-based program to play the ♔♙♚ ending according to some principles of game theory. Sakuta and Iida in ([7]) described a program to solve Kriegspiel-like problems in Shogi (Japanese Chess). Bud and others ([3]) described an approach to the design of a computer player for a sub-game of Kriegspiel, called Invisible Chess. Finally, our paper [1] describes a preliminary research on ♔♖♚ endings in Kriegspiel. The present paper expands and generalizes that work to other endings.

3 Metapositions

3.1 Game situation without stochastic element

Diagram 1 shows an example of position during a game of ♔♖♚ ending. Suppose it is White turn to move. At the first ply, the game tree whose root is the position on diagram 1 has a branch of 11 moves, corresponding to the possible White's moves, plus the Black's ones, which are, in 7 cases, 5; in 2 different cases they are 6, finally, in 2 last cases they are 3.

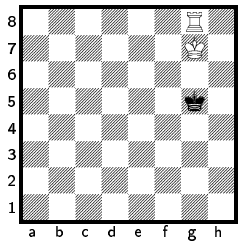


Table 1. Example of ♔♖♚ ending position.

Black's choices compose the information sets for White, who does not know where Black has moved his pieces. Thus there are 32 information sets for White, in all 53 nodes; going on with the tree visit we have to handle a numerical growth, that is really difficult to be dealt with brute force. Consider the tree depicted in figure 1 reached from the miniature in diagram 1 with move ♖f8: Black's possible moves are ♜h5, ♜h4 or ♜g4. White now faces two information sets of 2 and 1 elements respectively. The former includes 21 possible moves[3], while the latter includes 19 moves. Thus, the game tree branches out with $2 \cdot 21 + 19 = 61$ further nodes.

[3] 7 for the King and 14 for the Rook

In figure 1 x, y, and z denote the value of positions computed by an evaluation function we will deal with in section 4.

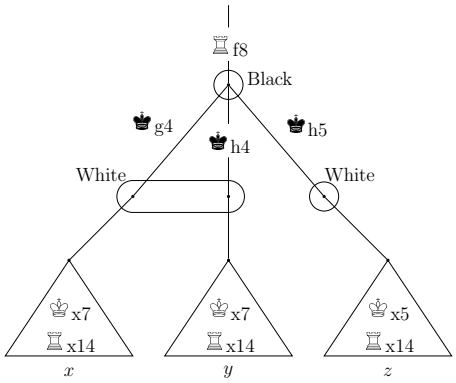


Fig. 1. White has 1 information set with more than 1 element.

In general, whether there is a information set with more than an element, it is not possible to employ a search algorithm on the game tree to find a optimal solution. If Black had a priority over his choices, which lead to the same information set, White could perform a tree visit to deduce which move of Black is the most dangerous. The problem arises when in the information set with more than one element we have to deal with alternative positions having the same probability.

A possible solution consists in joining the information sets into a single position which describes them all. By collecting those states reached with same likelihood we can simultaneously represent them, without the constraint of choosing one of them. In this way moves with different priorities can be safely represented without the risk of losing information. Thus, we adopt the notion of *metaposition* [7], which is a special position denoting a set of positions. Besides the referee's board, each player updates his own board, which is annotated with all the possible opponent's positions. We recall that we refer to the metaposition representing the knowledge of a player with the term *reference board*.

Now we join those moves made by Black which lead to the same information set, obtaining identical metapositions with similar uncertainty about the White's pieces positions. The example in figure 1 is reduced as in figure 2. Moves ♚g4 ∧ ♚h4, which led to the same information set, lead now to a unique metaposition.

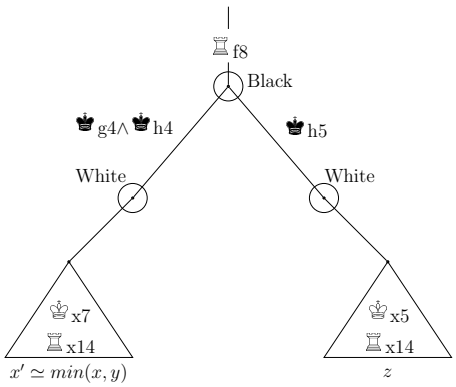


Fig. 2. White has 2 information sets with 1 element.

According to the definition, the tree in figure 2 represents a *game of perfect information*. In the example above the unification of moves leads to sub-gametrees whose evaluation is given by the minimum value of the original branches as evaluated before the join. The formula $x' = min(x, y)$, where x' is the new evaluation of the metaposition, requires the calculation of both x and y. Thus, the problem is not dissimilar from the previous one.

An improvement to the tree visit can be made considering the evaluation function not only with its recursive role, returning a value for a position at a particular depth, but also with a static meaning, in order to give a value during the tree visit and to distinguish the promising one among several branches.

Because of the complexity of a procedure which distinguishes between Black's moves that lead to different information sets for White, that is moves that lead to metapositions or to simple positions, we define the game tree in a simpler but equivalent way.

We define the notion of *metamove*, which is a move that the black King can perform and that transforms a metaposition into another one. A metamove allows White to update his reference board expanding all the possible moves for Black. The metamove does not include only the moves that lead to the same information set, but it comprises all the possible moves that Black can play from that particular metaposition. In some sense, we are transforming a basic Kriegspiel endgame in a new game where White has to confront several black Kings. In doing this we loose the property of information sets which claims that from each node into the set there are the same moves. We introduce the concept of *pseudomove* to indicate the moves by White on a metaposition. Pseudomoves are moves whose legality is not known to White.

The Black's moves joined for the previous example are ♚h5, ♚h4 and ♚g4. Figure 3 shows the new game tree with metapositions.

The set P of pseudomoves has a cardinality equal to that of the union of possible legal moves by White for each position.

$$|P| = |\cup(\text{legal moves})| \qquad (1)$$

For the example in figure 3 White always considers 21 pseudomoves, even if he is in the case with 19 legal moves, because he cannot distinguish between the two situations.

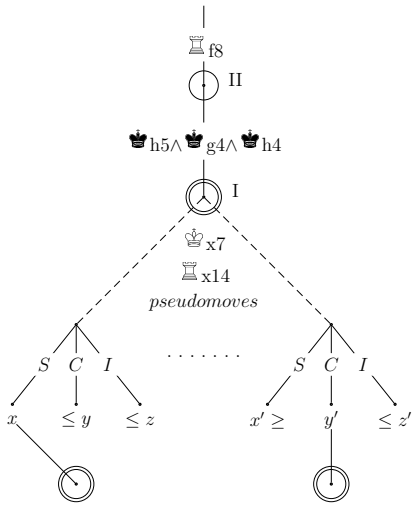


Fig. 3. The game tree with metapositions

In order to have the new tree equivalent to the former, we introduce the information given by the referee. In fact, what characterizes the pseudomoves is the referee's answer, which can be *silent* (S), *check* (C) or *illegal* (I). Thus, the game tree has a branching factor equal to $3 \cdot i$, where i is the number of metapositions.

Starting from diagram depicted in 1, there are 11 moves for White which lead, after considering Black's move, to 11 metapositions, as showed in figure

4^4. If we considered separately moves made by Black for each move by White we would have obtained 33 information sets and 53 positions in total. If we consider only metapositions, instead of all possible positions, we obtain just 11 metapositions. Thus, the game tree with the referee's answers has a branching factor equal to $3 \cdot 11 = 33$ nodes.

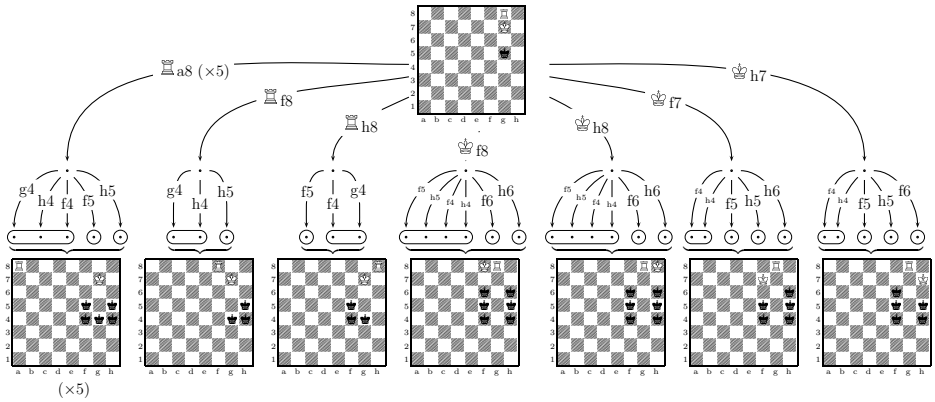


Fig. 4. Example of game tree

In figure 3, metapositions are depicted with a double circle and pseudomoves are depicted with a dotted line. During the search visit on the game tree, we use the heuristic that chooses the worse referee's answer among the three. Thus, in the previous example the branching becomes equal to 11 nodes. In figure 3 we indicate with x, x', y, y', z, z' the vote given statically by the evaluation function to the metapositions reached after playing each pseudomove.

3.2 Game situation with stochastic element

In this section we will deal with game situations where players have to consider probability.
For example a situation with a stochastic element may happen when players have to choose between moves that lead to symmetric positions and therefore they have to draw moves by lot.

Suppose we have the position where ♔c5, ♙c6 and ♚c7, depicted in figure 5. In this case, the unification of all Black's moves leads to a unique information set, and White's pseudomoves can be actually considered legal.

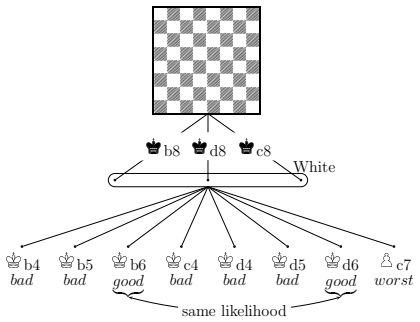


Fig. 5. Moves have same likelihood

[4] the move ♖a8 from diagram showed in 1 is similar to ♖b8 or ♖c8 or ♖d8 ♖e8, so in figure 4 we depicted only the first one and we indicated the whole number of moves (×5)

We define an evaluation function which allows us to classify the choices for White.This function is based on the following rules, in decreasing order of importance.

1. it never risks the capture of the Pawn;
2. it favours the advancing of the Pawn;
3. it pushes the Pawn to the seventh row if the white King is on the seventh row;
4. it keeps the white King and Pawn adjacent;
5. among those moves that lead the King on the row below the Pawn, it favours the move which brings the King on the same column of the Pawn.

Using these rules as an evaluation function, the move ♙c7 is considered the worst move, then it is discarded; with ♚b8, ♚c8 or ♚d8, after ♙c7 the Pawn would risk to be captured. Also the moves ♔b4, ♔c4 and ♔d4 are discarded, because they move the King away from the Pawn; moves ♔b5 and ♔d5 are better but not really good, because they do not push the Pawn. Finally, moves ♔b6 and ♔d6 are equivalent and best, since they push the Pawn and let the Pawn stay adjacent to its King.

The equivalence between ♔b6 and ♔d6 is inevitable. Figure 6 shows the symmetries between the two metapositions reached with these moves. Thus it is not possible to have a numerical value which correctly represents the grade for a metaposition and which is not correct for the symmetric one. In other words, in the game tree we would face two different nodes with same evaluation.

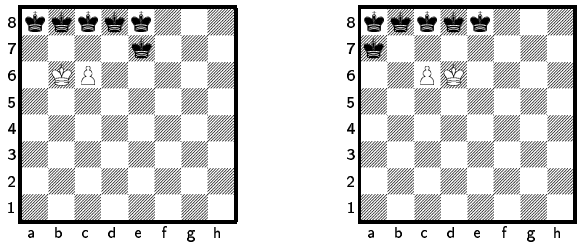


Fig. 6. Metapositions reached with ♔b6 and ♔d6

During the tree search, we use a random number generator to randomly assign a bonus with likelihood 1/2. In this case we use the term *aleatory metaposition*. We remark that, with an aleatory metaposition, each visit of its game tree becomes aleatory. A negative consequence of using aleatory metapositions is that we cannot employ techniques to accelerate the search, such as hash tables or Zobrist keys, since we would lose the stochastic nature of tree search.

In our example, let ♔b6 be the move randomly chosen, so White's reference board is the one on the left in figure 6. The White's pseudomoves ♔a7, ♔a6 and ♔a5 are discarded because they move the King away from the Pawn; the pseudomoves ♔b5 and ♔c5 are discarded, since they do not help to advance the Pawn. If ♔c7 is illegal, then White chooses the remaining ♔b7, followed

by ♙c7 if the referee's answer is silent. Otherwise, if ♔b7 proves to be illegal, White plays ♔c5, owing to the fifth rule, since the Pawn is on the c column.

4 The evaluation function

The evaluation function contains the rules which synthesize the notion of progress leading the player towards the victory. It is a linear weighted sum of features like the following

$$\text{EVAL}(m) = w_1 f_1(m) + w_2 f_2(m) + ... + w_n f_n(m) \tag{2}$$

where, for a given metaposition m, w_n indicates the weight assigned to a particular subfunction f_n. For example, a weight might be $w_1 = -1$ and $f_1(s)$ may indicate the number of black Kings.

The EVAL function is different according to each single ending, but it has some invariant properties: it avoids playing those moves that lead to stalemate and it immediately returns the move which gives directly checkmate, if it exists.

In the following sections we briefly describe the search algorithm used for some basic Kriegspiel endings, then we go into the evaluation of metapositions in more depth.

4.1 The search algorithm

As we have seen in Section 3.1, we consider that each node of the game tree consists of a metaposition. For example, suppose that the White reference board is the one depicted in figure 7 and that it is White turn to move.

The search algorithm proceeds by generating all the pseudomoves and, for each metaposition reached, it creates three new metapositions according to the three possible answers from the referee. Then it chooses the one with the smallest value as given by the evaluation function. In the example we have 21 pseudomoves which lead to 63 metapositions, but after filtering the information from the referee we obtain again 21 nodes.

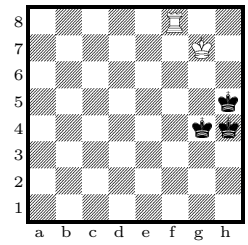


Fig. 7. Rook ending metaposition

Then, if the search algorithm has reached the desired search depth it simply returns the evaluation for the best node, that is the max value, otherwise it applies the metamove on each nodes, it decrements the depth of search and it recursively calls itself obtaining a value from the subtree.

Finally, it retracts the pseudomove played and adds to the metaposition's value the vote which is returned by the recursive call. Then it updates the max on that particular search depth.

When the algorithm ends visiting the tree, it returns the best pseudomove to play. Since it may happen that the same candidate pseudomove is proposed in

two different sequential turns to move, bringing to a loop and so not progressing, the algorithm avoids to choose those pseudomoves, which appear in the history of recently played moves.

4.2 The rook ending (♔♖♚)

The evaluation function for this ending considers $n = 6$ different features.

1. it avoids jeopardizing the Rook: $w_1 = -1000$ and f_1 is a boolean function which is true if white Rook is under attack;
2. it brings the two Kings closer: $w_2 = -1$ and f_2 returns the distance (number of squares) between the two Kings;
3. it reduces the number of black Kings on the quadrants of the board as seen from the Rook and it favors having the black Kings grouped together in as few quadrants as possible: $w_3 = -1$ and $f_3 = c\sum_{i=1}^{4} q_i$ where $c \in \{1, 2, 3, 4\}$ is a constant which counts the quadrants that contains a black King and q_i counts the number of possible black Kings on i^{th} quadrant;
4. it avoids the black King to go between white Rook and white King: $w_5 = -500$ and f_5 is a boolean function which returns true if the black King is inside the rectangle formed by white King and white Rook on two opposite corners;
5. it keeps White pieces close to each other: $w_5 = +1$ and f_5 is a boolean function which returns true if the Rook is adjacent to the King;
6. it pushes the black King toward the corner of the board: $w_6 = +1$ and $f_6 = \sum_{i=0}^{63} v[i]$, where v is a numerical 64-element vector, shown in figure 8, that returns a grade for each squares which possibly holds the black King or returns 0 otherwise.

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

Fig. 8. The simple numerical matrix $v[]$

Here we propose some example of games. We consider that the program plays White against a Black whose strategy consists in centralize himself on the board. Starting from the metaposition depicted on figure 9 on the right, where ♔ is on b6 and ♖ is on c1, the game continues as follows:
Considering ♚ on b8:
1. ♔c7 I, ♔a6; ♚a8
2. ♖c8 #.
Considering ♚ on a8:
1. ♔c7; ♚a7
2. ♖a1 #

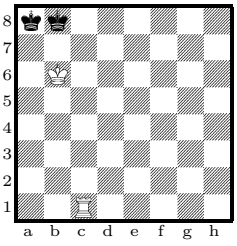


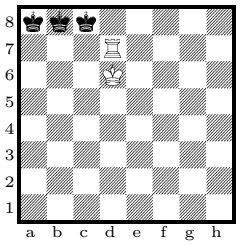
Fig. 9. Rook ending example.

Starting from the metaposition depicted on figure 10 on the left, where ♔ is on d6 and ♖ is on d7, the game continues as follows:

Suppose ♚ on c8:
1. ♔c7 I, ♔c6; ♚b8
2. ♔c7 I, ♖d6; ♚c8
3. ♔d7 I, ♔b6; ♚b8
4. ♖d8 #

Suppose ♚ is on b8:
1. ♔c7 I, ♔c6; ♚c8
2. ♔c7 I, ♖d6; ♚b8
3. ♔d7; ♚b7
4. ♔c7 I, ♔c6 I, ♖e6; ♚a7
5. ♔c7; ♚a8
6. ♔b6; ♚b8
7. ♖e8 #

Suppose ♚ on a8:
1. ♔c7; ♚a7
2. ♖d5; ♚a6
3. ♔b6 I, ♔c6; ♚a7
4. ♖c5; ♚a6
5. ♔c7; ♚a7
6. ♖a5 #



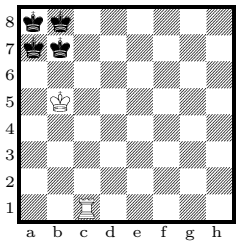


Fig. 10. Rook ending examples.

Starting from the metaposition depicted on figure 10 on the right, where ♔ is on b5 and ♖ is on c1, the game continues as follows:

Suppose ♚ on b7:
1. ♔b6 I, ♖c4; ♚b8
2. ♖c5; ♚b7
3. ♖c4; ♚b8
4. ♖c6; ♚b7
5. ♔c5; ♚b8
6. ♔b6; ♚a8
7. ♔c7; ♚a7
8. ♔c8; ♚a8
9. ♖a6#

Suppose ♚ is on a7:
1. ♔b6 I, ♖c4; ♚b7
2. ♖c5; ♚b8
3. ♖c4; ♚b7
4. ♖c6; ♚b8
5. ♔c5; ♚b7
6. ♔b6I, ♔b5; ♚b8
7. ♔b6; ♚a8
8. ♔c7; ♚a7
9. ♔c8; ♚a8
10. ♖a6#

Suppose ♚ on a8:
1. ♔b6; ♚b8
2. ♔c7 I, ♔a6; ♚a8
3. ♖c8#

Suppose ♚ on b8:
1. ♔b6; ♚a8
2. ♔c7; ♚a7
3. ♖a1#

Figure 11 shows an histogram which represents the number of moves needed to win each game, starting from metapositions with greatest uncertainty, that is from metapositions where each square not controlled by White may contain a black King. The number of matches won is on the ordinate and the number of moves needed is on the abscissa. The graph depicts the result of all the possible 28000 matches, which correspond to the 28000 possibilities for the referee's board or to the 28000 possible metapositions with greatest uncertainty. We can notice that the program wins the whole games with 25 moves in the average.

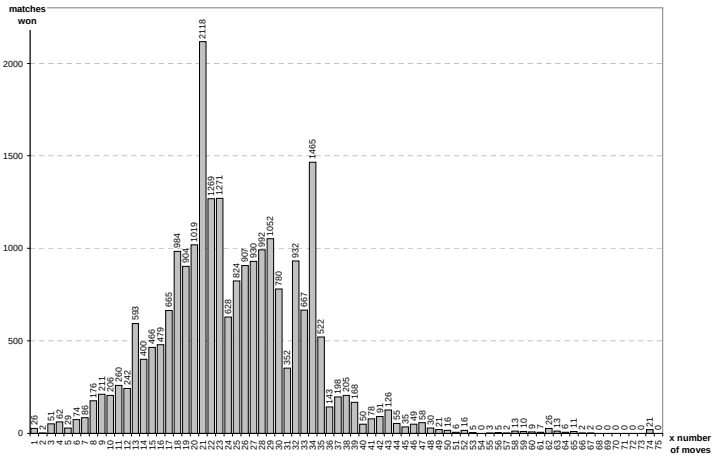


Fig. 11. Detailed histogram of ♔ ♖ ♚ ending game.

4.3 The queen ending (♔ ♕ ♚)

The evaluation function is similar to the one described in section 4.2 but we have to consider the Queen instead of the Rook.

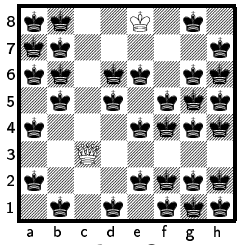


Table 2. The Queen cannot move.

In some initial experiments we noticed a problem in metapositions with the Queen far from the King and with more than one black King between them. This problem was caused by the choice of bringing the Queen closer to the King. For example, diagram 2 shows a metaposition where the white Queen cannot move without risking to be captured. Thus we introduced three more features with respect to the evaluation function in section 4.2. The first feature aims to avoid this problem and the other two intend to speed up the game by exploiting the power of Queen. So $n = 9$ and in the initial six rules the function is the same as in the Rook case[5], while in the last three:

7. it avoids metapositions where Queen risks to be captured: $w_7 = -100$ and f_7 is a boolean function that returns true if Queen is under attack;
8. it penalizes those metapositions with a big number of black Kings: $w_8 = -1$ and f_8 is equal to the number of black Kings on White's reference board;
9. it reduces the number of black Kings on the areas traced by the Queen's diagonals: $w_9 = -1$ and $f_9 = evalRhombArea(S)$ where

$$evalRhombArea(S) = c \cdot (a_0 + a_1 + a_2 + a_3) \tag{3}$$

[5] Notice that rule 7 differs only in weight from rule 1 used for rook ending; these rules could be combined into a single rule, but for the moment we keep them separated in order to maintain strategies for different endings divided.

and $c \in \{1, 2, 3, 4\}$ is a constant which counts the areas that possibly contains a black King and a_i $(i = 0, ..., 3)$ counts the number of possible black Kings on i^{th} area.

Figure 12 shows a graphic description of *evalRhombArea*() function. For the miniature on the left the function returns $4 \cdot (12 + 12 + 3 + 6) = 132$.

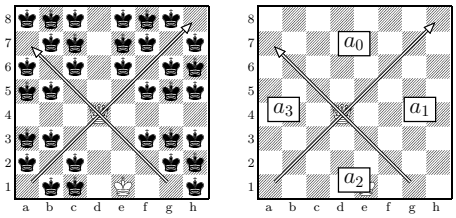


Fig. 12. Graphic description of evalRhombArea()

Now we show some examples of games played by the program. From a starting metapositions where ♕ is on d7, ♔ on c7, and ♚ on a6, ♚ on a7, ♚ on a8, the program correctly plays the move ♕a4#.

From a starting metapositions where ♕ is on h7, ♔ is on d7, and ♚ on a7, ♚ on a8, ♚ on b8, the game goes in accordance with the initial positions of Black as follows:

Suppose ♚ on a8:	Suppose ♚ on a7:	Suppose ♚ on b8:
♔c7; ♚a7	♔c7; ♚a6	♔c7 I, ♔c6; ♚c8
♔b6 I, ♕d3; ♚a8	♔b6 I, ♕d3+; ♚a5	♕h6; ♚d8
♕a6#	♕b3; ♚a6	♔c7 I, ♕f8#
	♕a4#	

Figure 13 shows an histogram analogous to the one for the rook ending. It represents the number of moves needed to win each game, starting from metapositions with greatest uncertainty. The number of matches won is on the ordinate and the number of moves needed is on the abscissa.

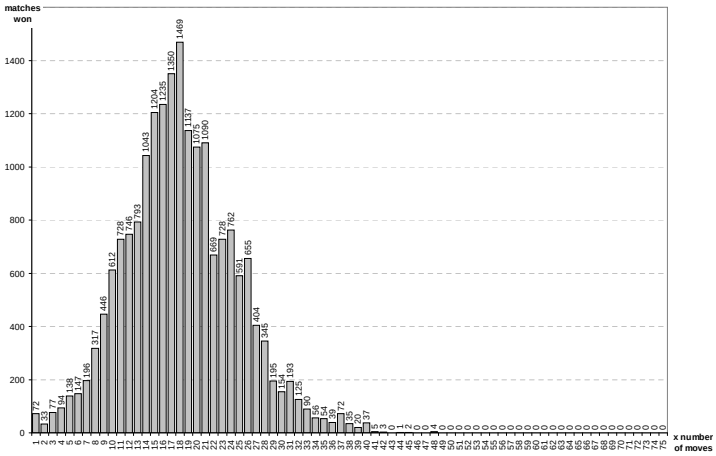


Fig. 13. Detailed histogram of ♔ ♕ ♚ ending game.

4.4 The ending with two Bishops (♔♗♗♚)

In this ending we have to deal with two White pieces besides the King. The evaluation function exploits the same subfunctions previously analyzed, but it assigns different weights.

1. it avoids jeopardizing the Bishop: $w_1 = -1000$ and f_1 is a boolean function which is true if white Bishop is under attack;
2. it brings the two Kings closer: $w_2 = -1$ and f_2 returns the distance (number of squares) between the two Kings;
3. it avoids the black King to "pass through" the border controlled by the Bishops: $w_3 = -500$ and f_3 is a boolean function which returns true if the black King is inside the rectangle formed by King and Bishop row or King and Bishop column;
4. it keeps close the white Bishops: $w_4 = +2$ and f_4 is a boolean function which returns true if the Bishops are adjacent to each other;
5. it pushes the black King toward the corner of the board: $w_5 = +1$ and $f_5 = \sum_{i=0}^{63} b[i]$, where b is a numerical 64-element vector, shown in figure 14, that returns a grade for each squares which possibly holds the black King or returns 0 otherwise.

$$\begin{pmatrix} 0 & -10 & -50 & -100 & -100 & -50 & -10 & 0 \\ -10 & -10 & -40 & -40 & -40 & -40 & -10 & -10 \\ -50 & -40 & -40 & -40 & -40 & -40 & -40 & -50 \\ -100 & -40 & -40 & -50 & -50 & -40 & -40 & -100 \\ -100 & -40 & -40 & -50 & -50 & -40 & -40 & -100 \\ -50 & -40 & -40 & -40 & -40 & -40 & -40 & -50 \\ -10 & -10 & -40 & -40 & -40 & -40 & -10 & -10 \\ 0 & -10 & -50 & -100 & -100 & -50 & -10 & 0 \end{pmatrix}$$

Fig. 14. The numerical matrix $b[]$

6. it keeps white King on the Bishop's row or column: $w_6 = +1$and f_6 is a boolean function which returns true if the King and the Bishop are on the same row or column;
7. it penalizes the metapositions where the Bishop risks to be captured: $w_7 = -100$ and f_7 is a boolean function that returns true if Bishops are under attack;
8. it penalizes those metapositions with a big number of black Kings: $w_8 = -1$ and f_8 is equal to the number of black Kings on White's reference board;
9. it reduces the number of black Kings on the areas traced by the Bishop's diagonals: $f_9 = evalRhombArea(m)$, described with equation 3, and
   if $evalRhombArea(m) < -600$ $w_9 = -4$;
   otherwise $w_9 = \frac{1}{6}$
10. it prefers some particular positioning (we will refer to with the term *key Bishops' positions*) for the white King and Bishops, highlighted in figure 15; for example ♔c7, ♗c4 and ♗c5. Therefore $w_10 = +30$ and f_10 is a boolean function which is true if the Bishops and the King are arranged in one of the key positions.

Now we propose some examples of endings in order to show the behavior of program with two Bishops.

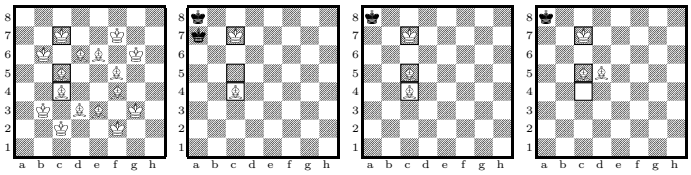


Fig. 15. Key Bishops' positions

Starting from the metaposition depicted on figure 16 on the left, where ♗ is on c4 and ♗ is on h8 and ♔ is on c7, the game continues as follows:

Suppose ♚ on a8:
1. ♔b6; ♚b8 2. ♗e6; ♚a8
3. ♗f6; ♚b8 4. ♗e5+; ♚a8
5. ♗d5#.

Suppose ♚ is on a7:
1. ♔b6 I, ♗d4+; ♚a8 2. ♗d5#.

Starting from the second metaposition depicted on figure 16 from the left, where ♗ is on c4 and ♗ is on d6 and ♔ is on c7, the game continues as follows:

Suppose ♚ on a8:
1. ♗b4; ♚a7 2. ♔b6 I, ♗c5+; ♚a8
3. ♗d5#;

Suppose ♚ on a7:
1. ♗b4; ♚a8 2. ♔b6; ♚b8
3. ♗e6; ♚a8 4. ♗e7; ♚b8
5. ♗d6+; ♚a8 6. ♗d5#.

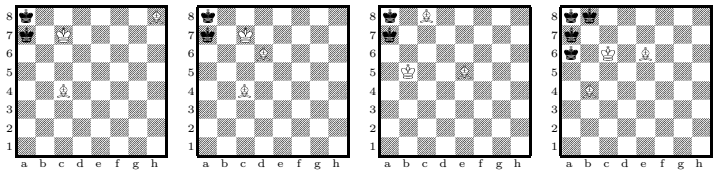


Fig. 16. Bishops ending examples.

Starting from the third metaposition depicted on figure 10 from the left, where ♗ is on c8 and ♗ is on e5 and ♔ is on b5, the game continues as follows:

Suppose ♚ on a8:
1. ♔c6; ♚a7 2. ♔c7; ♚a8
3. ♔c6; ♚a7 4. ♔c7; ♚a8
5. ♗f6; ♚a7 6. ♔b6 I, ♗d4+; ♚a8
7. ♗b7#.

Suppose ♚ is on a7:
1. ♔c6; ♚a8 2. ♔c7; ♚a7
3. ♔c6; ♚a8 4. ♔c7; ♚a7
5. ♗f6; ♚a8 6. ♔b6; ♚b8
7. ♗e6; ♚a8 8. ♗e7; ♚b8
9. ♗d6+; ♚a8 10. ♗d5#.

Starting from the metaposition depicted on figure 10 on the right, where ♗ is on b4 and ♗ is on e6 and ♔ is on c6, the game continues as follows:

Suppose ♚ on b8:	Suppose ♚ is on a8:	Suppose ♚ is on a7:	Suppose ♚ is on a6:
1. ♔b6; ♚a8	1. ♔b6; ♚b8	1. ♔b6 I, ♗c4; ♚a8	1. ♔b6 I, ♗c4+; ♚a7
2. ♔c7; ♚a7	2. ♔c7 I, ♗d6+; ♚a8	2. ♔c7; ♚a7	2. ♔c7; ♚a8
3. ♗b4; ♚a8	3. ♗d5#.	3. ♗c5+; ♚a8	3. ♗e7; ♚a7
4. ♗e7; ♚a7		4. ♗d5#.	4. ♗c5+; ♚a8
5. ♗d5+; ♚a8			5. ♗d5#.
6. ♗d5#.			

Figure 17 shows an histogram analogous to the one for the rook ending. It represents the number of moves needed to win each game, starting from meta-positions with greatest uncertainty. We can notice that for the ♔♗♗♚ ending the game is won in a bigger number of moves than those required to win for the ♔♕♚ or the the ♔♖♚ ending. Sometimes the program wins with more than 80 moves.

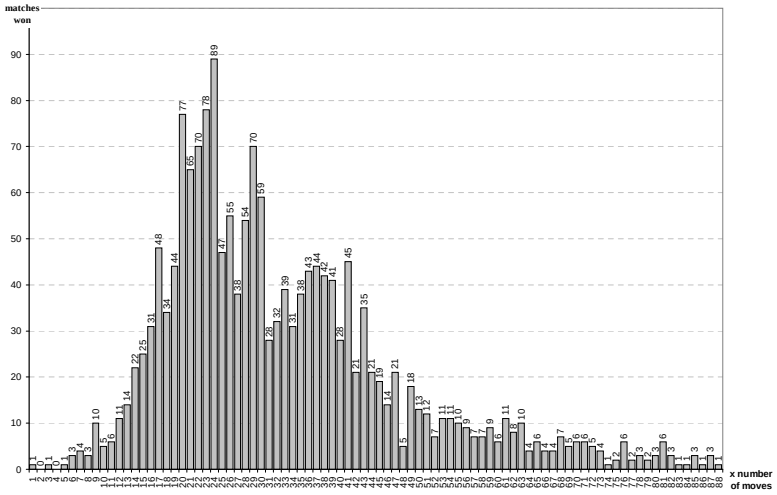


Fig. 17. Detailed histogram of ♔♗♗♚ ending game.

4.5 The pawn ending (♔♙♚)

The evaluation function for the ♔♙♚ ending takes the discussion in 3.2 as starting point. It considers $n = 5$ different features.

1. it brings the Pawn adjacent to the King: $w_1 = -1$ and f_1 calculates the distance between King and Pawn;
2. it pushes the Pawn: $w_2 = +1$ and $f_2 = 2 \cdot (Pawn's row)$;
3. it let the King above the Pawn: $w_3 = +1$ and $f_3 = (King's row) - (Pawn's row)$;
4. **If** $(Pawn's\ row == seventh\ row)$
   if $(Pawn's\ row) > (King's\ row)$ $w_4 = -1000$;
   otherwise $w_4 = +100$
5. **If** $(Pawn's\ row == sixth\ row)$

if $(King\ is\ on\ the\ right\ of\ the\ Pawn)\ w_5 = +5 + rand();$
if $(King\ is\ on\ the\ left\ of\ the\ Pawn)\ w_5 = +5 + rand();$

The fifth condition implements the stochastic choice and forbids the use of hash techniques.

In order to implement a 1/2 likelihood, it uses a random number generator indicated here with the function *rand*(). Figure 18 shows an histogram which represents the number of moves needed to win each game, starting from random metapositions.

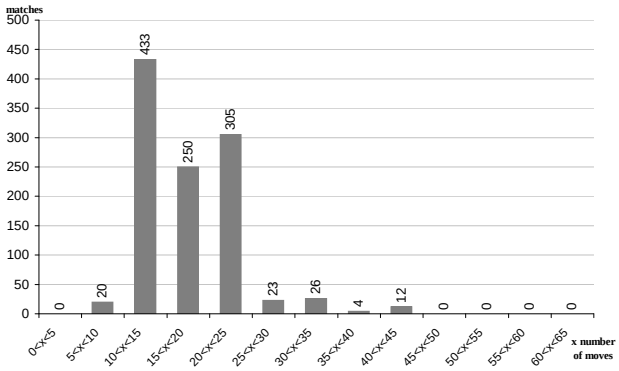


Fig. 18. Histogram of ♔ ♙ ♚ ending game.

5 Conclusions

In our knowledge this is the first time that an evaluation function including a notion of progress has been defined for Kriegspiel. We have devoted special care to implement progress inside such an evaluation function. We have tested such a function on some simple endings, with good results except for the KBN vs K case. Future work will lead us to adapt the program to more complex endings, where both players have a larger number of pieces on the board. Our aim consists in writing a complete program for the whole game of Kriegspiel.

References

1. A. Bolognesi and P. Ciancarini. Computer Programming of Kriegspiel Endings: the case of KR vs K. In J. van den Herik, H. Iida, and E. Heinz, editors, *Advances in Computer Games 10*, pages 325–342. Kluwer, 2003.
2. J. Boyce. A Kriegspiel Endgame. In D. Klarner, editor, *The Mathematical Gardner*, pages 28–36. Prindle, Weber & Smith, 1981.
3. A. Bud, D. Albrecht, A. Nicholson, and I. Zukerman. Information-theoretic Advisors in Invisible Chess. In *Proc. Artificial Intelligence and Statistics 2001 (AISTATS 2001)*, pages 157–162, Florida, USA, 2001. Morgan Kaufman Publishers.
4. P. Ciancarini, F. Dalla Libera, and F. Maran. Decision Making under Uncertainty: A Rational Approach to Kriegspiel. In J. van den Herik and J. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 277–298. University of Limburg, Maastricht, The Netherlands, 1997.
5. T. Ferguson. Mate with Bishop and Knight in Kriegspiel. *Theoretical Computer Science*, 96:389–403, 1992.
6. T. Ferguson. Mate with two Bishops in Kriegspiel. Technical report, UCLA, 1995.
7. M. Sakuta and H. Iida. Solving Kriegspiel-like Problems: Exploiting a Transposition Table. *ICCA Journal*, 23(4):218–229, 2000.