Moving in the Dark: Progress through Uncertainty in Kriegspiel

Andrea Bolognesi and Paolo Ciancarini

Dipartimento di Scienze dell'Informazione
University of Bologna, Italy

Abstract. Kriegspiel is a wargame based on the rules of Chess. A player only knows the position of his own pieces, while his opponent's pieces are "in the dark", ie. they are invisible. A Kriegspiel player has to guess the state of the game and progress to checkmate being in the dark about the history of moves (but he can exploit the messages of a referee).
Thus, computer playing of Kriegspiel is difficult, because a program has to progress in the game even with scarce or no information on its opponent's position. It is especially interesting to study the progress of computer play of simple Kriegspiel endings. We show how we tested a program able to play simple Kriegspiel endings to see how it progresses through uncertainty.

1 Introduction

Kriegspiel is a Chess variant similar to wargames. Each player uses a normal chessboard with normal pieces and normal rules, except that he cannot see his opponent's pieces. Both players are not informed of their opponent's moves. Each move is tried "in the dark", ie. knowing nothing about the position and strategy of the opponent.

Since Kriegspiel is based on Chess, a normal Chess program can be easily adapted to play Kriegspiel, for instance trying random moves until the referee accepts one. If we want to have a playing quality better than random, however, a special problem has to be addressed. Most computer chess evaluation functions compute a score evaluating both armies, whose position is well known, and then the search procedure progresses maximizing or minimizing the difference of score assigned to each army. In Kriegspiel this optimization is not possible, so progress (namely improving the program's army position with respect to the position of the adversary) becomes a problem. A player has to choose a "good" move being in the dark about the position of the enemy army.

In order to build a complete program able to play a good Kriegspiel game, the study of simple endings is useful because these endings are quite common in the practice of the game between humans. There is also a game-theoretic interest: in fact, a number of papers discuss abstract, rule based procedures suitable to solve the simplest endings from any initial position. A first study on the ♔♖♚ ending was published by Boyce, who proposed a complete procedure to solve it [3]. Also two Italian researchers studied this ending, more or less at the same time than Boyce's [8]. Then Ferguson analysed the endings ♔♗♘♚ [6] and

♔♗♗♚ [7]. A rule-based program to play the ♔♙♚ ending according to the principle of Bound Rationality was discussed in [4].

In this article we study and evaluate instead a search-based algorithm first exposed in [1] and generalized in [2] . It explores a game tree made of nodes which are metapositions [9], and uses an evaluation function in order to implement a progress heuristic.

We will deal with the basic endgames, i.e. those where Black has left the King only. This paper completes the work described in our papers [1,2]: there we discussed some partial results on the most common Kriegspiel endings, that we here consolidate and generalize. More precisely, in [1] we developed a basic program for the classic ending ♔♖♚; in [2] we extended our approach, developing a more general evaluation function useful also for other endings.

The goal of this paper is to study how our approach is effective, namely we aim at showing that our algorithm progresses even if it "moves in the dark". We initially compare our algorithm with an abstract algorithm proposed by Boyce [3]. The Boyce algorithm is rule-based and in our knowledge has been never implemented before. Our algorithm instead is search-based: our goal is to compare the two different algorithms from a practical, agonistic viewpoint. In fact, we have recently developed a complete playing program named Darkboard [5]: when we let it to play on the Internet Chess Club (ICC), human Kriegspiel players are very clever in exploiting its weaknesses in dealing with simple endings.

This paper has the following structure. In section 2 we evaluate our approach comparing it with the algorithm proposed by Boyce. In Section 3 we discuss the completeness of our algorithm. In Section 4 we draw our conclusions.

2 Tests and comparisons

In order to evaluate the quality of the algorithm described in [2], we have implemented another, different, rule based program which plays the procedure proposed in [3] to win the rook ending.

2.1 The Boyce Algorithm

Boyce showed a way to force checkmate by considering positions where both Kings are in the same quadrant of the board as seen from the Rook, or where the black King is restricted to one or two quadrants of the board.

The procedure applies when

1. both Kings are in the same quadrant as designed by the Rook; see Fig. 1;
2. the black King cannot exit from the quadrant;
3. the white Rook is safe.

Basically, the algorithm first ensures that the rook is safe from capture. Next White plays to a position where all the possible squares for the black King are in a rectangle where one corner is at the rook. White will put its King in that

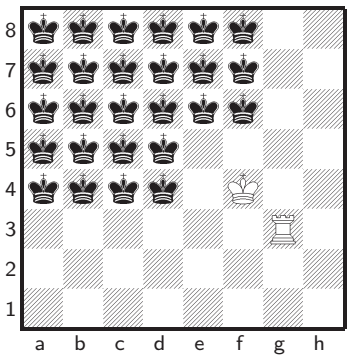


Fig. 1. Initial position of the Boyce's procedure.

rectangle to keep the black King away from its Rook. White then forces the black King back until it can occupy only those squares on a single edge. The final phase to checkmate is then fairly simple.

We have implemented a program which uses a search algorithm and a special evaluation function with the aim to obtain an initial position similar to that shown in Fig. 1. Then we apply the Boyce rules, and count how many moves are necessary in order to give mate.

2.2 Our Algorithm

Our search-based algorithm has been presented in [2]. Here we summarize only the main ideas in its evaluation function. The function includes six different heuristics.

1. it avoids jeopardizing the Rook;
2. it brings the two Kings closer to each other;
3. it reduces the size of the quadrant where the black King should be found;
4. it avoids the black King going between the white Rook and the white King;
5. it keeps the white pieces close to each other;
6. it pushes the black King toward the corner of the board.

These features are evaluated numerically and added to obtain the value for a given metaposition; a search program then exploits the evaluation function to visit and minimax a tree of metapositions [2].

2.3 Comparing the Two Programs

Figure 2 shows a graph which depicts the result of all the 28000 matches which can occur considering all the possible initial metapositions for the rook ending from the White's point of view, starting with greatest uncertainty, that is starting 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

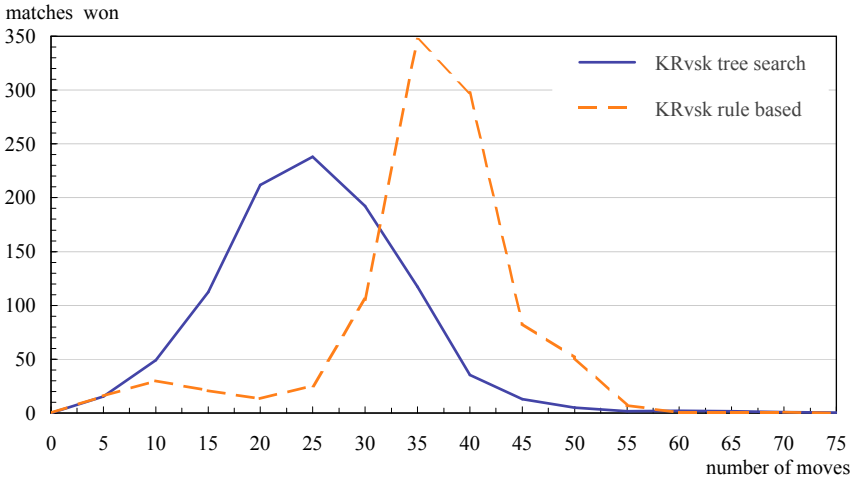


Fig. 2. Comparison of the rule-based program with the search-based program.

of moves needed to win each match is on the abscissa. The graphic shows the distribution of the matches won normalized to 1000.

The rule based program spends the first 25 moves looking for one of the initial positions; when it reaches one of these positions the checkmate procedure is used and the program wins very quickly. However, the average of moves needed is around 35. Our program based entirely on the search of game tree wins with a better average, around 25 moves.

This is due to the fact that the program analyzes from the beginning each position trying to progress to checkmate. On the other hand, the rule-based program is faster in deciding the move to choose, with respect to the tree-searching program. In fact, the rule-based program has a constant running time, whereas the second one has a running time exponential on the game tree depth.

We remark, however, that from a practical viewpoint the Boyce approach is useless because on the ICC Kriegspiel is played with the normal 50-moves draw rules derived from chess.

2.4 Evaluating the Search Algorithm with Other Endings

Figure 3 shows the results obtained with our search-based algorithm when analyzing some different basic endings. We performed a test choosing random meta-positions with greatest uncertainty for ♔♕♚, ♔♗♗♚, and ♔♗♘♚ endings; then we normalized the results to 1000 and we merged them to produce the ♔♖♚ figure.

In figure 3 we see that the program wins the ♔♕♚ ending quicker than the ♔♖♚ ending. This result was expected, because the Queen is more powerful

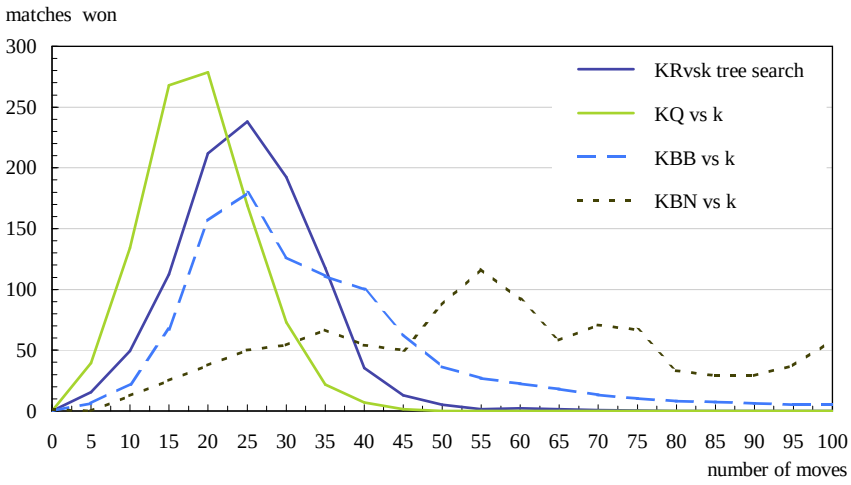


Fig. 3. Comparing the behavior of the search-based algorithm on different endings.

than the Rook: the Queen controls more space so metapositions have a lesser degree of uncertainty.

The case ♔♗♗♚ is instead more difficult with respect to ♔♖♚. In fact, the former is won on average in a larger number of moves: sometimes our program needs more than 100 moves.

Finally, we see that the behavior of our program in the ♔♗♘♚ ending is not good at all. The program often spends more than 100 moves to win and the distribution of victories does not converge to zero, meaning that sometimes it takes an unbound number of moves to win. We conclude that in this ending our program is not really able to progress.

2.5 Progress through Uncertainty

An effective way to analyze the progress toward victory consists in considering how the value of White's reference board changes after playing each pseudomove. The reference board is the metaposition which describes all positions were the opponent King can be, compatibly with the past history of the game.

Figure 4 shows the trend of evaluations assigned to each reference board reached during a whole match for the ♔♕♚ ending. The number of attempts needed during the game is shown on the abscissa, while the grades assigned by the evaluation function are on the ordinate.

We see that, at each step, the value of metapositions increases. From White's point of view, this means that the state of the game is progressing and this is actually a good approximation for the real situation.

We have performed the same test for the case of ♔♗♘♚ ending, whose result is depicted in Figure 5. Here the progress is not satisfactory for White, in

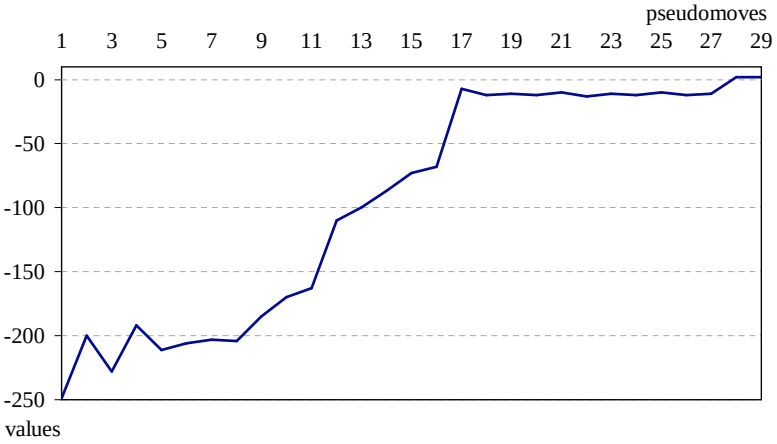


Fig. 4. Trend of evaluations assigned to metapositions crossed during ♔♕♚ ending.

fact he does not improve the state of the game at each step. The graph shows how the evaluations of the reference board change during a match which ends with the win of White: the value of metapositions does not increase at each pseudomove, but at some lucky situation for White. Thus the program basically wins this game by chance, that is by exploiting either some lucky metapositions or its opponent's errors.

We conclude that our program is able to progress to victory when we deal with pieces able to divide the board in separate areas, which can then be reduced to trap the black King; whereas when we have a piece which does not offer this feature, like the Knight, the behavior of the program is not fully satisfactory.

3 Optimality

In this section we deal with the issue of the optimality of our approach. We will show that our program is not able to win against an opponent using an oracle. If it won against an omniscient opponent it would be able to give optimal solutions, due to the search algorithm properties. We will point out that problems arise when the possibility of an illegal answer is considered into the tree of moves.

We remark that, according to its author, the algorithm given in [3] (or the (different) one given in [8]) can win against an omniscient opponent. However, these abstract algorithms do not always chose the optimal [1] move for each position the player may be in. Alas, these solutions follow a strategy fixed in advance that let the player to win even against an opponent using an oracle.

We showed in [2] that with a search on the tree of moves our program can guarantee the optimality if a checkmate position is among the visited nodes. In

[1] i.e. the move which leads to victory with the minimum number of moves

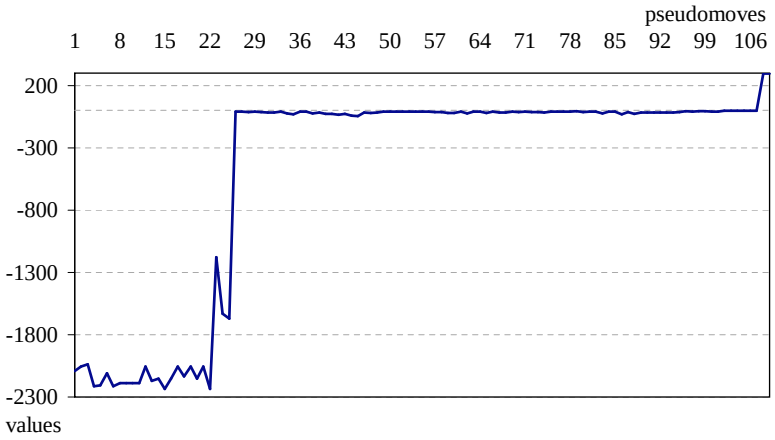
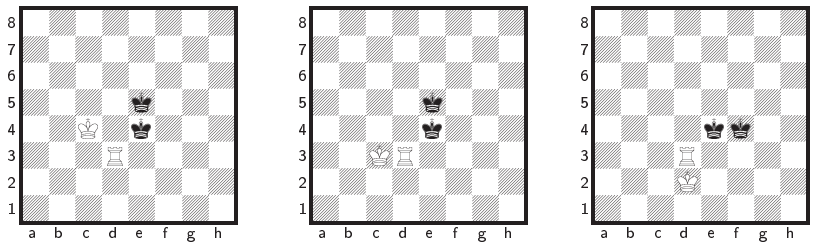


Fig. 5. Trend of evaluations assigned to metapositions crossed during ♔♗♘♚ ending.

all the other cases what we can say is that the move chosen by the program depends on the evaluation function which judges the positions. In these cases we cannot claim optimality.



a) *b*) *c*)

Fig. 6. Difficult positions against an omniscient opponent.

We will see in this section that the program with the evaluation function proposed in [2] does not always win against an omniscient opponent.

If the program ends up in the position *a*) depicted in figure 6 no checkmate state is found during the search on the tree of moves, so the program entrusts the evaluation function with the task to judge for sub-optimal positions. Since the evaluation function for the ♔♖♚ ending tries to reduce the uncertainty about the black King by decreasing the number of black Kings on his reference board, it tries to push the white King inside the quadrant controlled by the Rook. ♔d5 and ♔d4 are illegal moves, so the program plays ♔c3: the program plays subsequently ♔d2, then ♔e3, trying to pass through the row controlled by the Rook.

If ♔e3 is a legal move, then the referee remains silent and White can proceed, but if the referee says 'illegal', then the game reaches the position c) depicted in Figure 6. In such a case the program acts as if it would play for time: it chooses ♔e2 or ♔c3. This behavior depends on a random choice among the moves with same value. So if it plays ♔c3 then the game comes up again as in the initial position where we started, otherwise if it plays ♔e2, then at the subsequent turn it will try ♔e3 again.

We note that the program's behavior is not optimal: it entrusts its choices to the chance of having a silent referee after its move. In other words, if White plays ♔e3 and the referee is silent then the program progresses; actually it decreases the uncertainty about the black King's position. If it plays ♔e3 and the referee's answer is "illegal", then it does not progress.

If Black has no hint about White's moves, then a good strategy consists in centralizing his King to the middle of the chessboard. For instance, the black King could move from e5 onto e4 and viceversa, as highlighted in miniature d) in figure 7 on the left. In this case, the program with the evaluation function given in [2] progresses and its choices lead him to win the game.

On the contrary, if Black gets an oracle to help him, it can halt White's progress. In this case, White faces an omniscient opponent and the program fails by going forwards and backwards from position b) in Figure 6 to position c) in Figure 6.

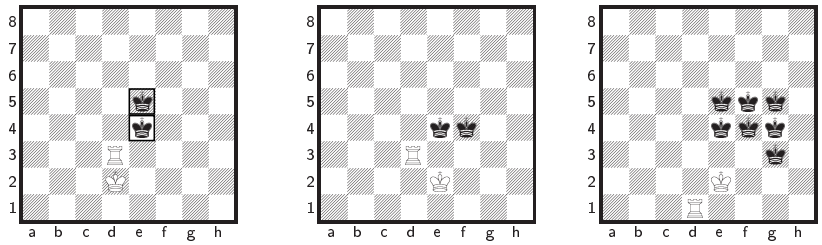


Fig. 7. Difficult positions against an omniscient opponent

From [3] we know that a good move for White consists in playing ♖d1, when the King is in e2 square (position e) in Figure 7). In fact this move allows White having his King inside the quadrant controlled by his Rook. With this move White reaches a position from which he can progress.

Writing an evaluation function which considers the position f) of figure 7 better then the position b) in figure 6 is quite a complex matter.

This problem can be expressed observing that from position a) in Figure 6, moving ♔c3 White reaches the position g) in Figure 8, then the move ♔d4 may receive two different answers from the referee: silent which leads to position h) in Figure 8, or illegal which leads to position i) in Figure 8. The worst answer

is the silent one[2], so the value for ♔d4 will be considered as the value of the position reached with a 'silent' answer.

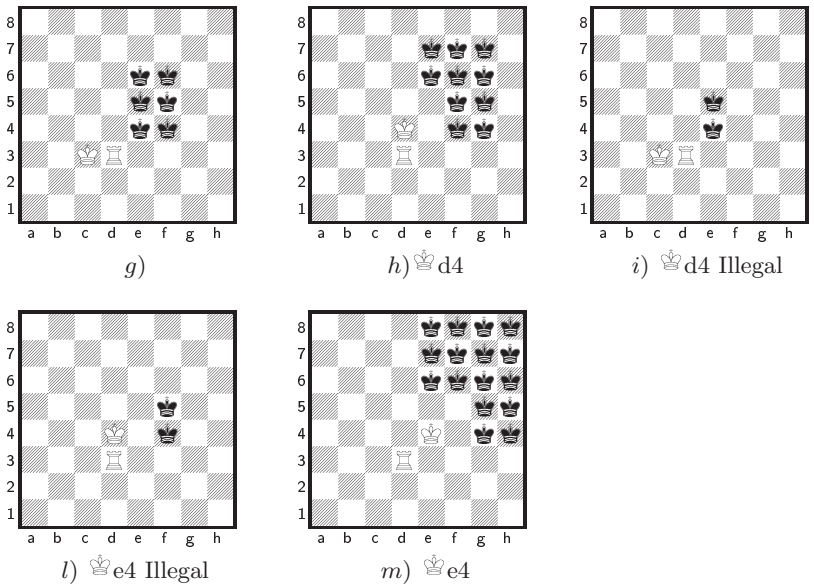


Fig. 8. Difficult positions against an omniscient opponent.

Again, starting from position h) in Figure 8 if White plays ♔e4 he may receive two kind of answer: silent or illegal. Also in this case the illegal answer is better because offers more information.

As a result, during the search each time the program analyzes moves of the King in situations similar to the position g) or the position h) of Figure 8, the illegal answer is not taken into account.

In Figure 9 is proposed an example of game tree. Starting from the position depicted on the root of the tree we have only represented three moves that White can choose.

We want to focus on the reasons that lead the program to prefer ♔c3 rather than ♔e2. Each time that a move can be illegal, this kind of answer leads to a node in the tree that is better valued with respect to the node resulting from a silent referee. Since the program considers the value of a move equal to the value of the worst position reached considering all the possible referee's answers, if a move may be illegal or silent then the worst one is the silent one.

Figure 9 shows that the leaf of the leftmost branch represents a better position than the leaf of the rightmost branch. In the middle there is a branch

[2] In order to progress, the number of black Kings in the reference board computed after each try should decrease: the best answer from the referee is 'illegal', because we can infer that the opponent King must be close to our King.

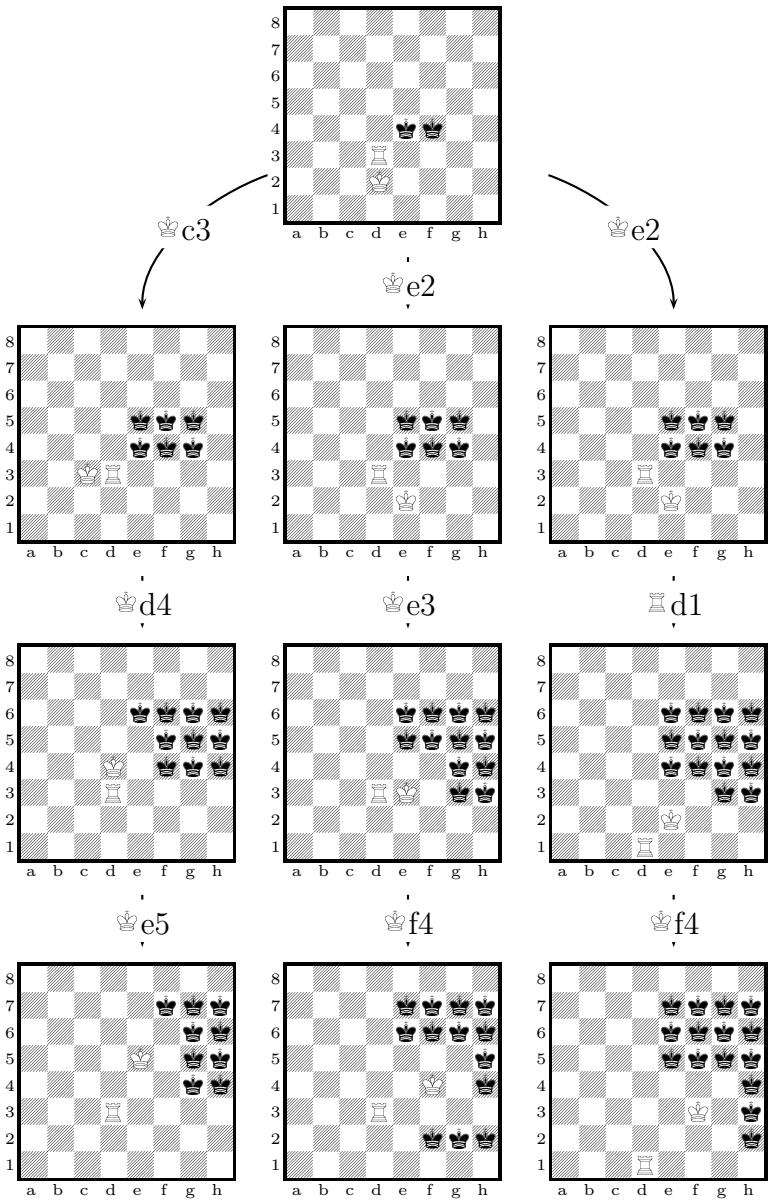


Fig. 9. Example of tree of metapositions.

concerning the ♔e2;♔e3 moves which lead to bad positions. The problem is due to a program blindness: it acts as if it was not distinguishing a potentially illegal branch from a certainly legal one.

An attempt to solve this problem consists in decreasing the value of illegal moves. If we do it too heavily we run the risk of letting the program to play only moves that are legal. In order to overcome this problem, we can try to discard from the search on the tree of moves the positions previously reached during the game. This is a trick that can avoid infinite loops, but it does not lead us to formulate an optimal solution to the problem.

4 Conclusions

The work done so far on a program which plays Kriegspiel endings exploiting the search on a tree of moves lets us to obtain optimal result for endings in which the checkmate is technically achievable in few moves. In all the other cases we delegate the judgement on moves to an evaluation function, that we proposed in [1] and we have generalized in [2].

The approximation due to this kind of evaluation lets the program to play reasonably against a fair opponent, which does not use an oracle. However the program plays not optimally in positions with larger uncertainty or against an omniscient opponent.

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. A. Bolognesi and P. Ciancarini. Searching over Metapositions in Kriegspiel. In J. van den Herik, Y. Bjornsson, and N. Netanyahu, editors, *Revised papers from 4th Int. Conf. on Computer and Games*, number 3846 in LNCS, pages 246–261. Springer, 2006.
3. J. Boyce. A Kriegspiel Endgame. In D. Klarner, editor, *The Mathematical Gardner*, pages 28–36. Prindle, Weber & Smith, 1981.
4. P. Ciancarini, F. DallaLibera, 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. Univ. of Rulimburg, 1997.
5. P. Ciancarini and G. Favini. Representing Kriegspiel States with Metapositions. In *Proc. Int. Joint Conf. on Artificial Intelligence (IJCAI 07)*, pages 2450–2455, India, January 2007.
6. T. Ferguson. Mate with Bishop and Knight in Kriegspiel. *Theoretical Computer Science*, 96:389–403, 1992.
7. T. Ferguson. Mate with two Bishops in Kriegspiel. Technical report, UCLA, 1995.
8. M. Leoncini and R. Magari. *Manuale di Scacchi Eterodossi*. Tipografia Senese, Siena, 1980.
9. M. Sakuta and H. Iida. Solving Kriegspiel-like Problems: Exploiting a Transposition Table. *ICCA Journal*, 23(4):218–229, 2000.