Moving in the Dark: Progress through Uncertainty in Kriegspiel

Andrea Bolognesi[1] and Paolo Ciancarini[2]
Dipartimento di Scienze dell'Informazione
University of Bologna, Italy

[1] : abologne@cs.unibo.it

[2] : cianca@cs.unibo.it

[1]Contents

1. Introduction

1.1. Complete vs partial information in games

- Board Games based on complete information:
  - The current state of the game is fully accessible to each player
  - Examples: Chess, Go.
- Board Games based on partial information:
  - Players have partial (and different) knowledge about the state of the game
  - Examples: Battleship, Stratego, Kriegspiel

1.2. Kriegspiel

- The players cannot see the opponent pieces
- All other rules of Chess still apply
- The players never communicate directly with each other, but instead interact with a referee.
- Only the referee knows the full state of the game.
- Players try moves which can be either accepted or rejected by the referee; if a move is rejected another move can be tried
- The referee announces checks and captures. These messages are sent to both players.

1.3. Research works on Kiegspiel

- Modeling and implementing a Kriegspiel referee program (Burger, Wetherell and others)
- Abstract algorithms for simple endings (Boyce, Ferguson, Ciancarini and others)
- Planning based on MonteCarlo Sampling
- AND-OR search of belief-state trees
- Reasoning about partially observed actions
- Algorithms for Kriegspiel variants
- Development of a Kriegspiel player program (Parker, Favini, others - see ICC)

1.4. Why do we study Kriegspiel?

- **Complex**: extremely large belief state makes an explicit representation of it computationally intractable
- **Challenging**: currently, the best humans are still far ahead of computer players at this game
- **Convenient**: same rules as Chess: this allows for reuse of a certain amount of game theory and software

Why do we study simple endings in Kriegspiel?

- 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.

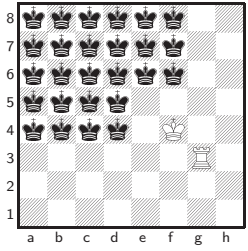
1.5. Our Aim

We have recently developed a complete playing program named Darkboard [CiancariniFavini 2007]: 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.

- Our goal 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 [Boyce 1981].
- Our algorithm instead is search-based.

2. 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;
  2. the black King cannot exit from the quadrant;
  3. the white Rook is safe.



2.1. Boyce Algorithm Implementation

- 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 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 paritcular evaluation function with the aim to obtain an initial position similar to that shown in the previous figure.
- Then we apply the Boyce rules in order to give mate.

3. Our algorithm

3.1. Metapositions

- Our program plays building a tree of metapositions
- A metaposition groups several game states together to provide the illusion of complete information
- The states with the same strategy space (set of moves available to the player) may be merged together and a game tree can be built
- Concept introduced in [Sakuta 2001] to deal with a Shogi equivalent of Kriegspiel, used to solve endgame positions

3.2. Kriegspiel Metapositions

- Simple structure; a metaposition is a board with both allied pieces and enemy pseudopieces on it
- Each square has a set of boolean flags telling which types of enemy pseudopieces can be on it.
- Each square has an integer number that serves as history, telling how many moves ago the square was last explored (not strictly required, but coding the history of the game helps prioritize some game states).
- Other information such as castling data.

3.3. Building a metaposition tree

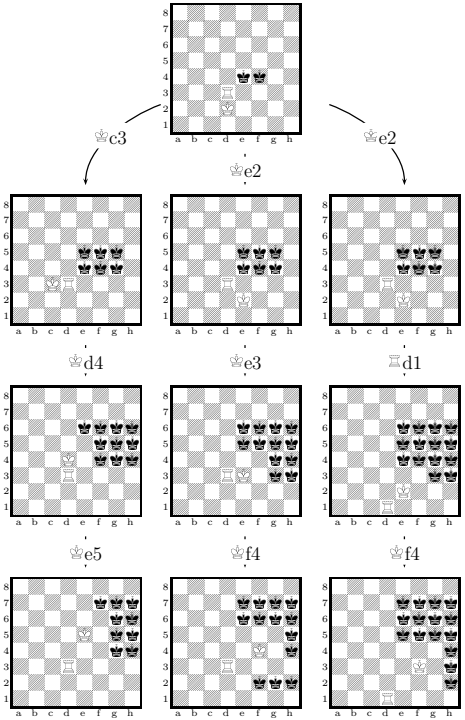
- A metaposition can be used to take incomplete information out of the picture. Instead of playing the original game, we can play a complete information game whose states are metapositions.
- If we can build a game tree of metapositions, we can then apply a minimax-like algorithm to evaluate moves. An evaluation function would be used that examines metapositions as a whole instead of single game states.
- Updating metapositions is the crucial step here

3.4. Pseudomoves and Metamoves

- Pseudomoves
  - represent the players move.
  - May be rejected by the referee in Kriegspiel.
  - A pseudomove and the referees responses uniquely define the resulting metaposition.
- Metamoves
  - represent the opponents hidden move.
  - The player does not know what the move is, so the new metaposition is uniquely defined by the referee's messages

3.5. Tree Structure

- 2-ply example.
- As many pseudomoves as there are possibly legal moves; the assumed referee response is generated with a set of rules.
- Only one metamove, generated in the same fashion.
- The agent is playing against the environment.


♔c3
♔e2
♔e2
♔d4
♔e3
♖d1
♔e5
♔f4
♔f4

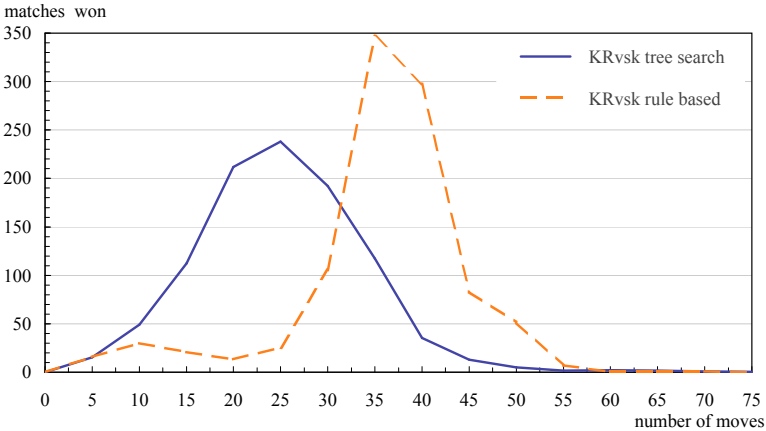
3.6. The evaluation function

Our search-based algorithm has been presented in [BolognesiCiancarini 2006]. 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.

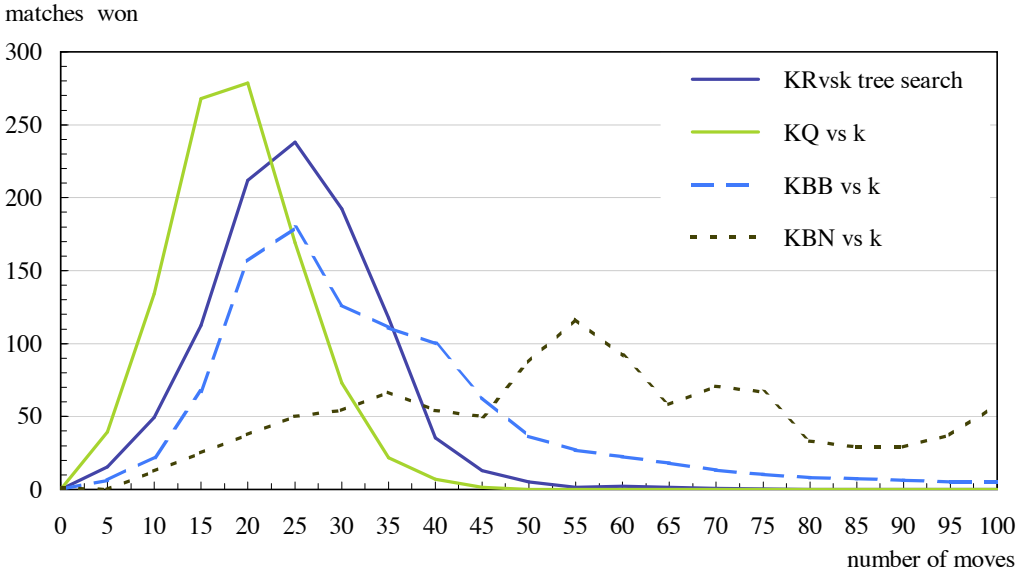
These features are evaluated numerically and added to obtain the value for a given metaposition; a search program then exploits the evaluation function to visit and minimax a tree of metapositions.

4. Results: Comparing the Two Programs



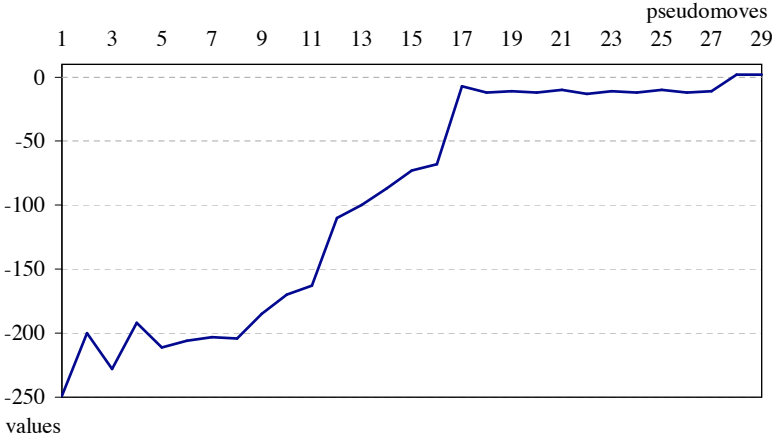
This graph depicts the result of all the 28000 matches which can occur considering all the possible initial metapositions for the rook ending from the White's point of view, starting with greatest uncertainty.

4.1. Evaluating the Search Algorithm with Other Endings



4.2. Progress through Uncertainty

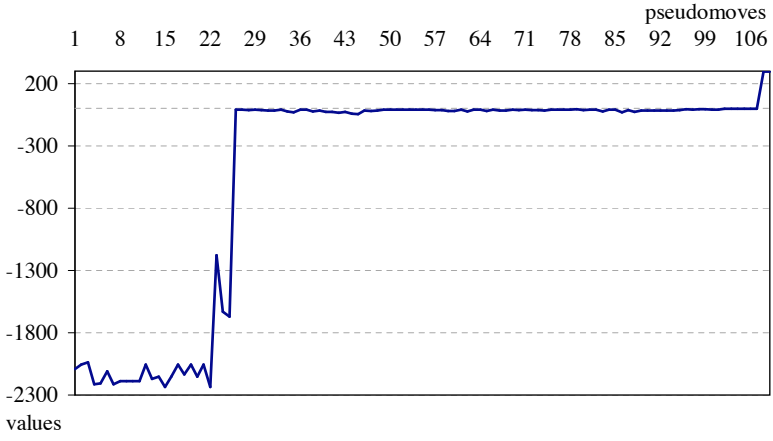
An effective way to analyze the progress toward victory consists in considering how the value of White's reference board changes after playing each pseudomove.



This graph shows the trend of evaluations assigned to each reference board reached during a whole match for the ♔♕♚ ending.

4.3. When progress is not satisfactory

We have performe



Here the progress is not satisfactory for White, in fact he does not improve the state of the game at each step.

The value of metapositions does not increase at each pseudomove, but at some lucky situation for White.

5. Conclusions and Future works

- Our program (with the evaluation function proposed in [BolognesiCiancarini 2006]) does not always win against an omniscient opponent
- Very difficult to implement the abstract algorithms, and possibly non convenient for practical goals
- Study of more complex endings