



Monte Carlo tree search techniques in the game of Kriegspiel

Paolo Ciancarini and Gian Piero Favini

University of Bologna, Italy

22 IJCAI, Pasadena, July 2009



Agenda

- Kriegspiel as a partial information game
- MonteCarlo in Kriegspiel
- Experimenting different MonteCarlo approaches for Kriegspiel



Referee's messages

- **Illegal**: the move is retracted and another move must be tried
- **Silent** (no message): the move is legal and stays
- **Capture**: the move is legal and “something” has been captured
- **Check**: the move is legal and a direction of check is also given (row, column, diagonal, knight)

NB: there exist different variants of Kriegspiel where more or less information is given by the referee; we use the variant played on the ICC



Uncertainty in Kriegspiel

- The nature of Kriegspiel uncertainty is completely dynamic: information is scarce and ages quickly.
- Unlike Phantom Go or Stratego, information does not increase over time.
- In Kriegspiel we do not know what pieces the opponent has, where they are, and what he knows about us.
- Interestingly, most Chess strategies remain valid, for instance in the endgame
(Exercise: define a winning strategy for KR vs K)



Progress

- In Chess or Go “progress” is crucial: programs rely upon minimax and evaluations which compare White and Black positions
- How can we measure the progress if we do not see the opponent’s position?
- Strategic victory condition: not based on the accumulation (and comparison) of an overall score, like in computer chess or go.
- Material advantage helps, but it is not enough to win.



Research works on Kriegspiel

- Definition of game-theoretic algorithms for simple endings (Shapley, Boyce, Ferguson, Ciancarini and others)
- Planning based on MonteCarlo sampling (Parker and Nau and Subrahmanian, IJCAI 2005)
- AND-OR search of belief-state trees (Russell and Wolfe IJCAI 2005)
- Reasoning about partially observed actions (Rance Vogel Amir AAI 2006)
- A full program to play Kriegspiel is described in Ciancarini Favini, IJCAI 2007.
This program won the Kriegspiel Championship at the Computer Olympics in 2006 and 2009



MonteCarlo in Kriegspiel

- Monte Carlo Tree Search (MCTS) has been extensively used to play a variety of complex board games such as Go.
- It has also been used to play imperfect information games such as Phantom Go and Kriegspiel (Parker)
- Our aim is to adapt MCTS so it can play Kriegspiel better than our past program based on minimaxing a tree of metapositions



Monte Carlo Tree Search (1/2)

- MCTS builds a game tree and usually consists of four steps that are repeated iteratively as long as time allows.
- 1) Selection:** the algorithm selects a leaf node from the tree according to some policy (eg. UCT: Upper Confidence-bound applied to Trees); it has similarities to an exploration/exploitation problem
 - 2) Expansion:** optionally, the algorithm expands a leaf to the next depth level (for example, after x visits).



Monte Carlo Tree Search (2/2)

- 3) **Simulation:** A full game (or several) is simulated from a leaf node in the tree. Moves are random, but preferably guided by some heuristics.
 - 4) **Backpropagation:** The value of the simulated game(s) is propagated to the node's ancestors up to the root, usually by averaging it. This will affect subsequent selection steps.
- ... but how does a textbook application of this algorithm perform in Kriegspiel?



Problems with approach A

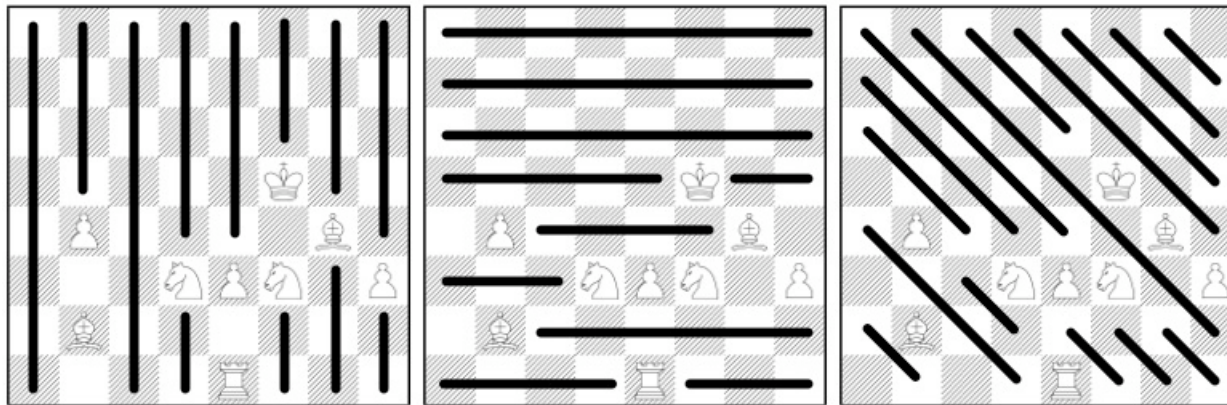
- It is difficult to create good random layouts for the opponent's pieces.
- It is also time-consuming, which is very harmful to a Monte Carlo method.
- Simulating the game with random moves makes for very long games that usually result in a draw regardless of the starting position.
- Except in very specific scenarios, approach A turns out to be about as strong as the random player.



Approach B (1/2)

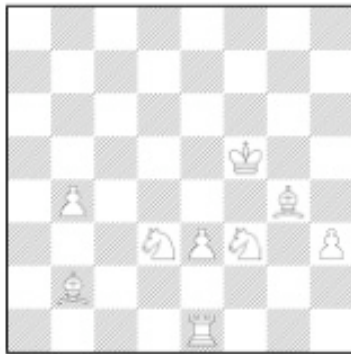
- Approach B tries neither to generate layouts for the enemy pieces, nor to move them on the board.
- Only the player's moves and their consequences are simulated.
- **The referee's messages are simulated.**
- The algorithm estimates the probability of each message being given by the referee (can be upgraded with opponent modeling data).
- Even if the probabilities are not very accurate, they are more reliable than generating random layouts - not to mention much faster.

Spreading probabilities



- Every horizontal, vertical or diagonal sequence of 2 or more squares is considered.
- For each sequence, the total probability of a piece (other than King and Pawn) being there is unchanged, but the probabilities for the individual squares are adjusted so they are closer to the average.
- We ignore Knight moves for performance reasons.

Approach B (2/2)



- After estimating that, for example, Bb2-a1 has a 20% chance of capturing something...
- ... we run our Monte Carlo simulations as before, but we simulate our own moves and update the board according to the referee's simulated messages (as defined by a probability board).
- There is one more addition from approach A: a *simulation cutoff* after k moves.



Simulation cutoff

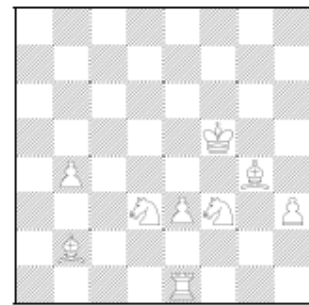
- Checkmate is approach A's major problem. Progress to checkmate happens very seldom with random moves, adding too much noise to the evaluation.
- To remedy this, we add a little game-specific knowledge to the algorithm.
- Instead of running each simulation to the end, we stop it after k moves and adjudicate the game to the player that seems to be winning.
- This function is much simpler than a true "evaluation function" and just counts the number of pieces each player has.



Approach C

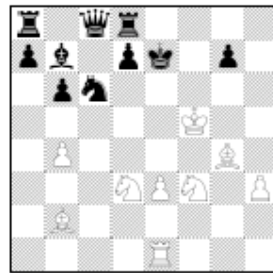
- The final approach is the same as approach B, with $k = 1$.
- Simulation is stopped after only one move.
- Since there is only one move to simulate, the result can be computed as a weighed average of the possible referee messages for that move.
- Every node is computed only once, saving time. Also, simulation is very accurate in the short range, though short-sighted (but the algorithm can use quiescence).

The three approaches

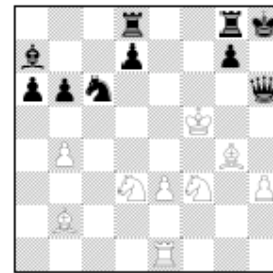


Value of b4-b5?

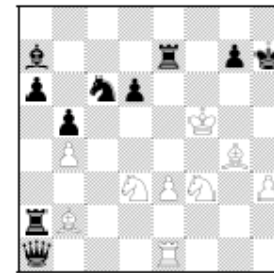
A



Umpire is silent



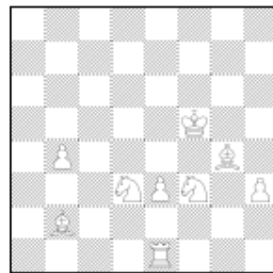
Pawn try



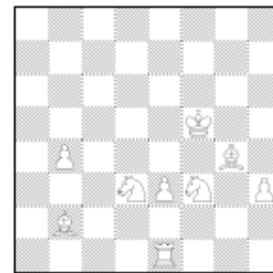
Illegal move

Full game simulations

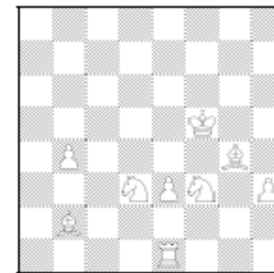
B



Silent (35%)



Pawn try (30%)



Illegal (35%)

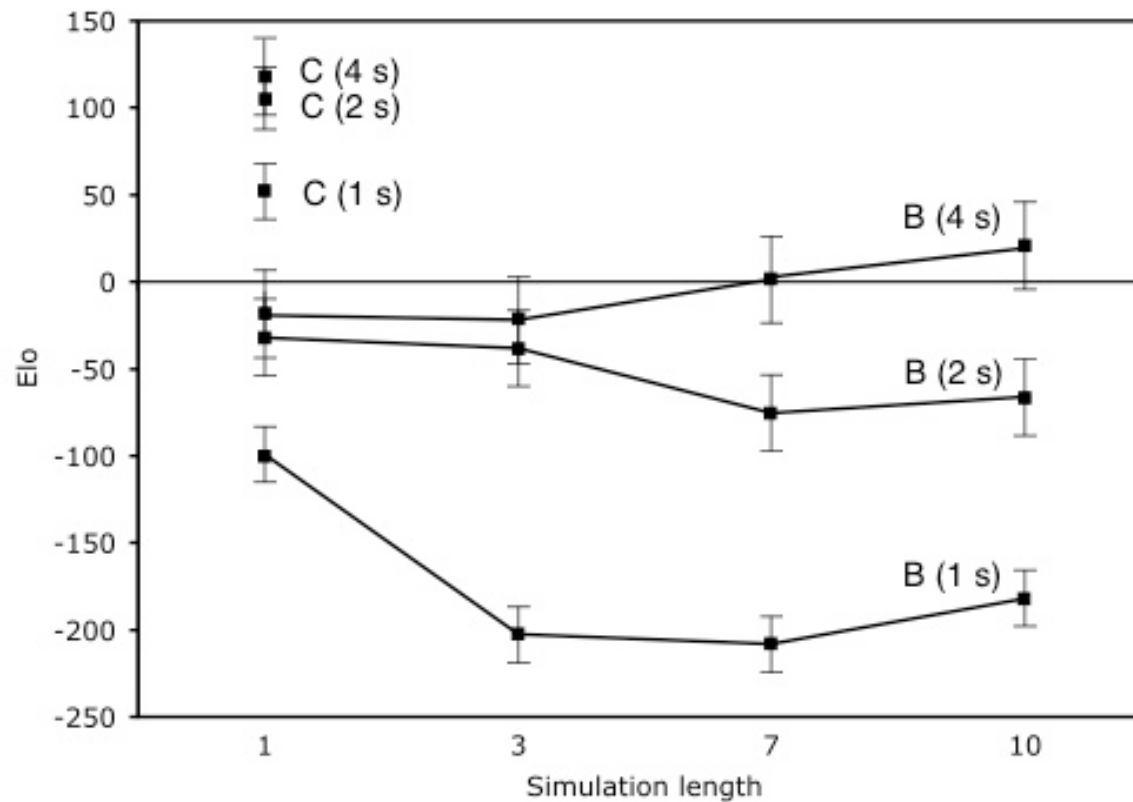
k-move simulations

C

$$\text{value} = 0.35 * v(\text{silent}) + 0.3 * v(\text{pawn_try}) + 0.35 * v(\text{illegal})$$

Weighed average of B (k=1)

Experimental results



$k = 1 \div 10$

- We test our B and C programs against an existing Kriegspiel player based on minimax search.
- We test a 1,2,4 seconds per move
- Surprisingly, short-sighted C performs best.



Conclusions

- C performs better because it simulates better in the short range and can explore more nodes, but...
- ... on higher time settings, B seems to be catching up. Eventually we expect B to be able to beat C.
- Longer simulations perform better as soon as they can explore enough nodes.
- The minimax player is clearly defeated by the Monte Carlo approach.



Question Time

- Questions?