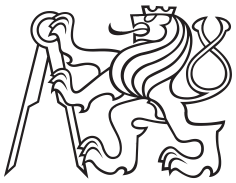
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

A Heuristic Algorithm for Kriegspiel

Bc. Vojtěch Foret

Supervisor: Ing. Michal Šustr
May 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Foret Vojtěch** Personal ID number: **456991**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master's thesis details

Master's thesis title in English:

A heuristic algorithm for Kriegspiel

Master's thesis title in Czech:

Heuristický algoritmus pro Kriegspiel

Guidelines:

1. Make a literature overview of algorithms for solving large games of imperfect information, with the emphasis on Kriegspiel [1].
2. Implement the game and reimplement a state-of-the art algorithm [2] as a baseline in OpenSpiel [3].
3. Implement a new algorithm, based on sampling compatible boards [4] and evaluating them with a traditional chess engine [5] as a heuristic for the value function.
4. Evaluate the algorithms in pair-wise matches to compute a statistically significant result.

Bibliography / sources:

[1] Li, David Hsiang-fu. Kriegspiel: Chess Under Uncertainty. Premier Publishing Company, 1994.
[2] Paolo Ciancarini, Gian Piero Favini, Monte Carlo tree search in Kriegspiel, Artificial Intelligence, Volume 174, Issue 11, Pages 670-684, 2010.
[3] Lanctot M, Lockhart E, Lespiau JB, Zambaldi V, Upadhyay S, Pérolat J, Srinivasan S, Timbers F, Tuyls K, Omidshafiei S, Hennes D. OpenSpiel: A framework for reinforcement learning in games. arXiv preprint arXiv:1908.09453. 2019 Aug 26.
[4] Ciancarini P, Favini GP. Representing Kriegspiel States with Metapositions. InIJCAI 2007 Jan 6 (pp. 2450-2455).
[5] Acher, Mathieu, and François Esnault. "Large-scale analysis of chess games with chess engines: A preliminary report. arXiv preprint arXiv:1607.04186 (2016).

Name and workplace of master's thesis supervisor:

Ing. Michal Šustr, Department of Computer Science, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **21.02.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **19.02.2023**

Ing. Michal Šustr Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

First, I would like to thank my supervisor Michal Šustr for his guidance. During our regular consultations and discussions he provided me with many useful comments and valuable insights.

I would also like to thank my family for their support and help. Especially, I thank my wife for her never-ending patience and supply of encouragement, and my little son for finally learning to sleep the whole night.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

In Prague, 21. May 2021

Abstract

Games are a simple model of real-world problems. Games with imperfect information extend this model by adding hidden information and thus being closer to real problems. However, these games are much more difficult for computer programs to play, especially if there is a significant amount of hidden information. Kriegspiel is an imperfect information variant of a well-known game of Chess, in which the player does not know the positions of the opponent's pieces. Helpful information is scarce in Kriegspiel, which makes it difficult for computer programs. Moreover, very little research has been done on computer programs playing the game. This thesis tests general algorithms for playing imperfect information games and the strongest known Kriegspiel playing program. It improves them by integrating advanced and well-tested algorithms for playing perfect information Chess. The final algorithm is indeed stronger than the original version. This shows that determinization evaluation is a powerful tool that might also be useful other imperfect information games with well-researched perfect information versions.

Keywords: Kriegspiel, Game Theory, Imperfect Information, Monte Carlo, Monte Carlo Tree Search

Supervisor: Ing. Michal Šustr

Abstrakt

Hry představují jednoduchý model problémů reálného světa. Hry s neúplnou informací vylepšují tento model přidáním skryté informace, čímž se přibližují skutečným problémům. Tyto hry jsou však pro počítačové programy náročnější na hraní, obzvlášť pokud mají velmi silnou skrytou informaci. Kriegspiel je varianta známé hry Šachy, ve které hráč nezná pozice nepřátelských figurek. Užitečné informace jsou v Kriegspielu vzácné, což z něj dělá velmi složitou hru pro počítačové programy. Navíc bylo provedeno jen velmi málo výzkumu na poli hraní celé hry Kriegspielu. Tato práce testuje obecné algoritmy pro hry s neúplnou informací a také dosud nejsilnější známý algoritmus hrající Kriegspiel. Tyto algoritmy vylepšuje využitím pokročilých a dobře vyzkoušených algoritmů pro hraní šachů s úplnou informací. Výsledný algoritmus skutečně předvedl lepší výkony než původní verze. Toto ukazuje, že evaluace determinizací je důležitý a mocný nástroj, který by mohl mít budoucnost i v dalších hrách s neúplnou informací, u kterých jsou dobře prozkoumané verze s úplnou informací.

Klíčová slova: Kriegspiel, teorie her, neúplná informace, Monte Carlo, Monte Carlo Tree Search

Překlad názvu: Heuristický algoritmus pro Kriegspiel

Contents

Figures

Tables

ctuthesis t1606152353

Chapter 1

Introduction

1.1 Motivation

Games are a great model of real-world problems. They provide an environment much simpler than the real world with much fewer possible actions to take. It is used for developing and testing new algorithms and techniques that can be later generalized and used in artificial intelligence for agents operating in the real world. Games with imperfect information are especially interesting because they generalize the environment by adding hidden information, coming closer to the real world, as agents in the real world do not have all the information about the environment they find themselves in.

Among games with imperfect information, Kriegspiel stands out as it is an imperfect information version of a popular and, from a game-theoretical standpoint, very well researched game of Chess. Kriegspiel offers strong imperfect information - the information gained by players is scarce and loses most of its relevance after only a few moves. This makes it extremely difficult to play well not only for humans, but especially for computer programs. This has been proved several times throughout the decades of research of Kriegspiel. It is so difficult for programs to play that decades of research have been done only on solving specific problems within Kriegspiel, e.g., endgames. Only recently serious attempts to play the whole game have been made, with only one known algorithm being able to play against average human players decently.

This thesis aims to improve and test existing algorithms for playing games

with imperfect information, especially those tested on Kriegspiel, by adding Chess domain knowledge. Chess has been the most popular game for computer programs to play for a long time. This resulted in Chess having many advanced computer programs that play on a super-human level. As the game of Kriegspiel builds on the rules of Chess, we can use these advanced algorithms to provide a heuristic for algorithms playing Kriegspiel.

Chapter 2

Background

This chapter provides a brief introduction to the background needed to understand the concepts described later in this thesis. I introduce notations for imperfect information extensive-form games and define used expressions.

2.1 Imperfect Information Games

Definition 2.1. An extensive form game (EFG) with imperfect information as defined in [Moravčík et al., 2017] consists of:

1. $\mathcal{N}$ is the set of players including the nature player (c) that represents the dynamics of the game,
2. for each $i \in \mathcal{N}, \mathcal{A}_i$ is the set of actions available to player i,
3. $\mathcal{H}$ is a set of possible states of the game, where each state corresponds to a history of actions performed by all players,
4. $\mathcal{Z} \subseteq \mathcal{H}$ is a set of terminal game states,
5. $\mathcal{P} : \mathcal{H}\ \mathcal{Z} \to \mathcal{N}$ is a function assigning each non-terminal state a player that selects an action in the state,
6. $\mathcal{A} : \mathcal{H}\ \mathcal{Z} \to \mathcal{A}_i$ is a function assigning each non-terminal game state the actions applicatble by the acting player,

7. $\mathcal{T} : \mathcal{H} \times \mathcal{A}_i \to \mathcal{H} \cup \mathcal{Z}$ is the transition function realizing one move of the game,
8. $u_i : \mathcal{Z} \to \mathbb{R}$ gives the utility of player 1,
9. $\mathcal{I}_i$ for player i represents player's imperfect, information about the game. It is a partion of $\mathcal{H}_i = \{h \in \mathcal{H} : \mathcal{P}(h) = i\}$ called information sets. Each information set $I \in \mathcal{I}_i$ represents the set of histories that are indistinguishable for player i,
10. $\triangle_c(h) \in \triangle(\mathcal{A}(h))$ is the commonly known probability distribution of nature player's actions.

These games can be represented by a tree, where nodes represent the game states (histories), leaves the terminal states and the root node the empty history.The edges represent actions and the transitions from one state to another after applying the actions.

The expression **determinization** is in literature used for two different things. One meaning of determinization is a particular history of the imperfect information game compatible with a player's information set. Therefore, the information set can be represented as a set of determinizations compatible with all the player's observations in the game so far. Another meaning of determinization is is Perfect Information Monte Carlo - an algorithm based on sampling histories compatible with the current information set [Cowling et al., 2012]. In this thesis, I am going to use the expression in the first mentioned meaning.

Chapter 3

Imperfect information chess variants

Chess has a legendary status among board games and games in general. For a long time, the state of artificial intelligence research was measured by how well computers could play Chess. A significant step forward took place in 1997 when IBM's Deep Blue [Campbell et al., 2002] defeated the, at the time, chess world champion Kasparov. Since Chess is such a popular game, there are many widely spread chess variants. The most popular chess variant is probably Chess960 proposed by Fisher, but many other variants differ much more from the original game of Chess. For example, Four Player Chess, King of the Hill, and imperfect information variants. This section will take a closer look at the imperfect information variants of Chess and especially Kriegspiel.

3.1 Kriegspiel

Kriegspiel is a chess variant where the player cannot see the enemy pieces. When playing this game in person, three chessboards and three sets of pieces are needed. The game needs a referee (called umpire) who is the only one having complete information about the state of the game. He has a chessboard with all the pieces. Players have chessboards with only their own pieces on the chessboard. They can also have a set of enemy pieces, but their purpose is only to visualize certain possibilities to the player and have no impact on the game. Players move their pieces on their chessboards, the umpire updates his chessboard accordingly and tells *messages* to the players.

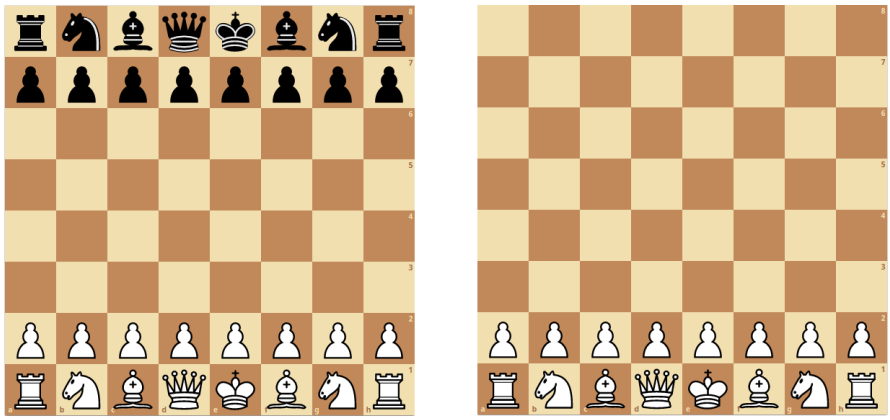


Figure 3.1: Comparison between a chessboard in the game of Chess (left) and in Kriegspiel (right) from the perspective of white player

Generally, all the messages can be heard by both players, but each player infers different information from them. These messages can be:

- **Illegal** - Since the player cannot see enemy pieces, this is not an uncommon scenario. The player can attempt to move through enemy pieces or move into check. In this case, the player is notified that his chosen move was illegal and can try again.
- **Nonsense** - Since both players can hear all the messages, one could try to confuse the opponent by trying obviously illegal moves like moving a horse as a bishopmoves, or moving through allied pieces. This is avoided by using a message different from Illegal.
- **White to move** - The move was successful, and it is white's turn
- **Black to move** - The move was successful, and it is black's turn
- **Piece captured** - The move was successful, and a piece was captured. Both players are notified at which square the capture happened. Optionally, some information about the captured piece can be given, but never about the capturing piece.
- **Check** - Players are notified when check occurs. Additionally, the type of check is given:
  - file,
  - rank,
  - long diagonal,
  - short diagonal,

- knight.

As Kriegspiel has a long history, there exist many different variations. The first notable one is the English Kriegspiel. The most significant rule in this ruleset is called "Are there any?". The player can pose this question to the umpire to determine whether there are any possible pawn captures. The umpire answers *yes* or *no*. If the answer is *no*, the player knows that there are no available pawn captures. If the answer is *yes*, the player knows that there is at least one possibility for capturing a piece with a pawn. The only price he has to pay for this information is that he has to try playing at least one pawn capture. This rule is widely spread because it saves plenty of time that would be wasted trying all kinds of pawn captures at each turn. Attempting all possible pawn captures is a popular strategy because:

1. Capturing a piece is not easy in Kriegspiel.
2. Trying to capture with a pawn can result in either capture or illegal move, which results in playing again.
3. Capturing with a pawn carries little to no risk.

The ruleset we are interested in is the one used in Internet Chess Club (ICC). Kriegspiel is played in ICC under the name Wild 16 (Chess variants in ICC are called Wild Chess and Kriegspiel happens to be 16th on their list). This ruleset is widely spread and right now is the closest one to being standardized. It was also adapted in the 11th and 14th Computer Olympiad. Under these rules, players do not receive messages about the opponent's illegal moves. It also goes further with the "Are there any?" rule than the English variant. At the beginning of each turn, the player receives a message about the number of pawn tries (possible pawn captures), if there are any. Additionally, when a piece is captured, besides the location of the capture, players receive information whether the captured piece was a pawn or another piece.

3.2 Imperfect Information Chess Variants

Besides different variants of Kriegspiel, there are imperfect information chess variants with rules sufficiently different to Kriegspiel that they have a different

Figure 3.2: Dark Chess chessboard from the perspective of the white player

name. The two most spread variants are Dark Chess and Reconnaissance Blind Chess.

Dark Chess gained popularity thanks to being available on a famous free server *chess.com* under the name Fog of War Chess. Dark Chess offers more information about the state of the game to the player and is easier to win than Kriegspiel. Each player can see their own pieces and all the squares they can legally move to. This offers much more information about the chessboard and the opponent's pieces, especially with a queen or a rook on an open file. In Fog of War Chess, even at the beginning of the opponent's move, one can see squares where he could potentially move if it were their turn. This allows for sending self-sacrificial scouts to the opponent's territory. However, the most critical change to the rules of Chess is the complete removal of the concept of a check. The player is not notified about a check, and the king can move in and out of check. Checkmate is no longer the objective, and the game ends when the king is captured.

Another imperfect information chess variant gaining popularity is Reconnaissance Blind Chess (RBC) [Newman et al., 2016]. Similarly to Dark Chess, check is also removed from the game, and the objective is to capture the opponent's king. The player can only see his own pieces and nothing more, just like in Kriegspiel. It would seem that this game is just a crossing of these two games, but Reconnaissance Blind Chess adds an entirely new feature to the game. At the beginning of each turn (except for the first), the player can select an area of 3x3 squares on the board. The player learns the positions and types of all pieces in this area. This adds a new layer of strategy complexity to this game.

Darkchess and RBC are interesting for us because they are present a simpler version of Kriegspiel. Since the player does not have to check-mate a king and the games offer much more information about the state of the game, they are much easier to win. With more information about the game state, one can also more easily represent the information set. This makes them easier to play for humans and computers. However, Kriegspiel being a challenging game is why it is very interesting from a game-theoretical point of view. This thesis will focus on algorithms playing Kriegspiel, using Darkchess and RBC as examples where similar algorithms were used and how successfully.

Chapter 4

Algorithms

4.1 Information Set Representation

Uncertainty is a big part of what Kriegspiel is about and representation of the information set is very important. Non-trivial ways to represent the belief state in Kriegspiel are needed because it gets too large to store in memory explicitely in a matter of a few moves. In this chapter I describe existing ways to represent the information set and what algorithms have been applied to Kriegspiel in the past.

4.1.1 Statistical Sampling

Since the information set tends to be too large, statistical sampling can be used to sample random states consistent with the current information set. It was proven successful in games like Bridge [Ginsberg, 2001] and Scrabble [Sheppard, 2002]. Four different statistical sampling algorithms were defined and tested on Kriegspiel in [A. Parker, 2005].

All Observation Sampling (AOS) generates random states consistent with all the observations the player received since the beginning of the game. This algorithm can sample all states within the information set when given enough

time. However, the required time grows exponentially with the branching factor of the underlying perfect information game. The branching factor in Chess is relatively high, on average around 30. The game of Chess is already long, but Kriegspiel games, challenging to win as they are, tend to be even longer. This makes All Observation Sampling time-consuming and infeasible in practice.

All Observation Sampling with Pool (AOSP) keeps a pool of states consistent with all previous observations. When a new observation is received, the pool is updated. When the pool grows too large, random samples are removed, so the pool remains acceptable in size. When the information set grows large enough, it becomes highly likely that none of the states in the pool is the current actual state of the game. Observations may reveal this fact, and the pool ends up empty. It is then necessary to make new samples and fill the pool again. This approach was succesfully used in StrangeFish [**?**], a computer program playing RBC. However, StrangeFish was able to find a strategy for using RBC's sensing so effectively, that the information set never grew too large to store in memory. This is not possible in Kriegspiel, so in our case, the pool size has to be limited.

Last Observation Sampling (LOS) generates random states consistent with the last observation available to the player. Ignoring all previous observations can lead to a loss of information. However, information obtained from observations quickly loses its significance in Kriegspiel and all previous observations may not be needed. On the other hand, the Kriegspiel observations rarely give much information to guide the sampling. The information set is so large that, later in the game, Last Observation Sampling becomes more useful because the samples are more likely to be consistent with the current information set.

Hybrid sampling (HS) is a combination of AOSP and LOS. It keeps a pool of states consistent with all observations at the beginning of the game. When an observation proves some of these states wrong and the pool thins under the desired size, it is filled with samples retrieved by Last Observation Sampling. This combines the early game accuracy of AOSP and the fact that LOS provides fast samples that are mostly consistent with the information set in the late game. LOS also provides diversification to the pool, enhancing the AOSP approach.

4.1.2 Metapositions

Metapositions were introduced in [Sakuta, 2001] to solve endgame positions for an imperfect information game based on Shogi. As described, metaposition merges different moves into one state. A metaposition represents a set of states with the same actions available to the player.

Another definition of metapositions that better suits the needs of Kriegspiel was formulated in [P. Ciancarini, 2007]:

Definition 4.1. If $\mathcal{S}$ is the set of all possible game states and $\mathcal{I} \subseteq \mathcal{S}$ is the information set comprising all game states compatible with a given sequence of observations (referee's messages), a metaposition $\mathcal{M}$ is any *opportunely coded* subset of $\mathcal{S}$ such that $\mathcal{I} \subseteq \mathcal{M} \subseteq \mathcal{S}$.

As such, metapositions provide a compact way to represent a superset of an information set. Having a state representing a set of states instead of sampling them presents an opportunity to treat games with imperfect information as games with perfect information and apply well-known algorithms to such games.

Metapositions applied to the game of Kriegspiel were introduced in [Bolognesi and Ciancarini, 2004] for solving simple endgame problems and in [Ciancarini and Favini, 2010b] they were used to find strategies with guaranteed victory in KRK (King and Rook vs. King), KQK (King and Queen vs. King), KBBK (King and two Bishops vs. King), and KBNK (King, Bishop and Knight vs. King) endgames. A great accomplishment was finding a deterministic strategy for winning the KBNK endgame, which was previously thought impossible [Ferguson, 1992]. Later, metapositions were also used in a computer program playing the whole game of Kriegspiel [P. Ciancarini, 2007].

A metaposition is represented as a chessboard with standard pieces and new entities called pseudopieces. The standard pieces represent the player's pieces, so if there is a piece at a given square, we know that there is nothing else at the square. However, a pseudopiece represents the possibility of an opponent's piece being at a given square. As such, there can be multiple pseudopieces on one square. Figure 4.1 shows a metaposition, where the opponent is left with only the king. Two types of moves can be applied to these metapositions: pseudomoves and metamoves. A pseudomove updates a metaposition based on the player's attempted move and the umpire's response to it. A metamove updates a metaposition after the opponent makes a move and the player receives the umpire's message. A metamove puts a clone of every pseudopiece to all his possible move destinations consistent with the umpire's message (e.g., putting the player's king in check or capturing a piece). Pseudopieces

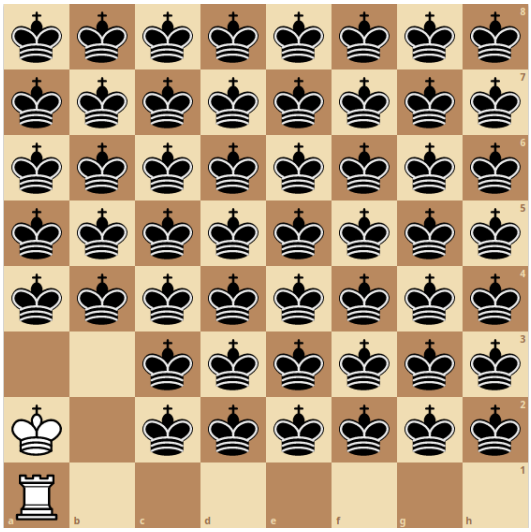


Figure 4.1: Metaposition for a KRK endgame scenario

cannot move through pieces, but they can move through other pseudopieces, because there is a possibility that the square is empty. This is the reason why metaposition is a superset of the information set.

Let us say that a pseudopiece moves from a square A to a square B, but there is another pseudopiece at square B. The metaposition does not know whether the square B can be empty under the condition that A i occupied by the piece represented by the pseudopiece. This leads to metapositions containing also states that are not in the current information set.

4.2 Darkboard

Darkboard algorithm was introduced in [P. Ciancarini, 2007]. It uses metapositions described in the previous section to represent a superset of the information set. The algorithm is similar to a min-max algorithm and uses a custom evaluation function for Kriegspiel represented by metapositions.

Darkboard uses some optimizations in building the game tree as the actual game tree would be too large. After the opponent's unknown move and after each of the player's possible moves, several possible umpire messages are compatible with the current metaposition. If the game tree contained all these possibilities, the branching factor would be too large. Instead, only one reasonable umpire message is generated in each node, according to a prediction heuristic.

The umpire prediction heuristic has two sets of rules. The first set of rules determines the umpire messages generated after the player's pseudomove:

- Every move is legal.
- The move does not capture anything unless it is a pawn try, a move to a square that is certainly not empty or retry of an illegal move but one square shorter.
- The captured entity is always a pawn if there is a possibility that there is a pawn on the square. Otherwise, a piece is captured.
- Opponent pawn tries are generated always when there is a possibility that the moved piece is a target of a pawn try.

The second set of rules determines the umpire messages generated after the opponent's metamove:

- The opponent never captures anything.
- The opponent never threatens the player's king.
- Pawn tries are never generated.

The reason for the messages surprisingly never considering the opponent capturing a piece or threatening player's king is that the evaluation function considers these risks already. The game tree does not need to simulate this because the risks tied with exposing pieces are heavily punished in the evaluation function.

These heuristics generate reasonable umpire messages. They will also never generate messages that imply a lucky move, like revealing the opponent's king or capturing a piece. However, with the growing tree depth, the accuracy of these heuristics drops very quickly. It is also less accurate during the middle game when silent umpire messages are less likely.

The goal of using metapositions in Kriegspiel is to allow use of traditional perfect information techniques like Min-Max in Kriegspiel. However, as mentioned above, only one possible umpire message commenting opponent move is generated at each state when it is the opponent's turn. This makes the MIN layer redundant, and therefore each node in the search tree represents two plies (one complete turn), and the algorithm resembles a weighed maximax.

It is weighed by a prediction coefficient, reflecting the level of confidence in the heuristic generating umpire messages.

The evaluation function uses three main components that the algorithm aims to maximize throughout the game: material safety, position, and information.

Material safety evaluates how well protected each piece is. It prefers pieces being protected by other pieces. It also takes into consideration that some squares are more dangerous than others and that a recently moved piece is more likely to be captured.

The position component evaluates the material rating with dynamic values for pieces as the game advances. It uses factors like pawn advancement bonus, having pawns on files without opponent pawns, and the number of controlled squares. When Darkboard is considering checkmating the opponent, it also aims to push the opponent's king pseudopieces to the board's edges.

Evaluating the amount of known information is a great advantage of using metapositions in contrast to statistical sampling information set representation. When sampling states compatible with the information set, we cannot properly evaluate the size of the information set. When using metapositions, the algorithm aims to reduce the size of the metaposition's position set by reducing the number of pseudopieces on the board. This component of the evaluation function also encourages the algorithm to explore squares that have not been visited for a longer time.

Using metapositions with techniques for playing perfect information games in Kriegspiel had a great success. The algorithm consistently ranked among the top 20 players on ICC and won the Gold medal at the Eleventh Computer Olympiad in Turin. However, this approach has several disadvantages. The heuristic for generating umpire messages quickly becomes very inaccurate as we search deeper in the game tree, which leads to the algorithm not improving its play after 2 seconds of computing. Another issue with this approach is that the author has to invent a representation of metaposition and an evaluation function for using it, both domain-specific. A good evaluation function requires a great deal of domain knowledge about the underlying perfect information game and the imperfect information game itself. It should also consider that the metaposition represents a superset of the information set, so a lot of information might get lost.

4.3 Perfect Information Monte Carlo

Perfect Information Monte Carlo (PIMC), sometimes also called determinization, is a simple approach to deal with uncertainty in games with imperfect information. It was first used in the game of Bridge together with other approaches to improve the gameplay and managed to become the strongest Bridge computer program, even being on the same level as top humans [Ginsberg, 2001].

Rather than dealing with the uncertainty, it could be said that the algorithm avoids it. The approach is simple. We sample a random determinization compatible with the information set and evaluate it with techniques for games with perfect information. We can repeat this until we run out of determinizations or until time runs out. Results of these evaluations are then combined, and the best action is chosen. Some of the advantages of this approach are that it can be easily parallelized and that we can use techniques for games with perfect information, which have been known and tuned for some time.

The disadvantages of Perfect Information Monte Carlo are less obvious than the advantages. One of the problems with PIMC is strategy fusion [Long et al., 2010]. When evaluating a specific determinization, the algorithm does not realize that it plays a game with imperfect information. It believes that it can use a different strategy in different states within one information set. For example, it can choose an action that leads to a state with an excellent response to any of the opponent's moves available in every state from the information set. However, it does not know in which state it is, so it might as well be impossible to choose the desired response, as it could be different in each state of the information set.

Another problem with PIMC is non-locality [Long et al., 2010]. In games with perfect information, the value of a node depends on the subtree starting with this node. However, in games with imperfect information, it can also depend on other regions of the game tree, outside of the node's subtree. This is caused by the opponent's ability to choose actions that lead to regions of the tree that he believes are more advantageous for him, based on his private information.

Another problem is that the algorithm does not work with uncertainty and hidden information. Since the evaluation has no concept of hidden information, it will never consider hiding information or revealing information as a factor in decision making. This can potentially have a significant impact in games like Kriegspiel, where our actions directly influence how much information the opponent gets or can get after his move.

Despite these flaws, Perfect Information Monte Carlo shows great results in games like Bridge. Results presented in [Long et al., 2010] show that PIMC has a serious disadvantage in games with low leaf correlation (probability that each sibling pair of terminal nodes will have the same payoff value) and with low disambiguation (how much a player's information set shrinks each time the player is to move). As mentioned, the disambiguation factor is extremely low in Kriegspiel because the player gets very little information about the state of the game.

Perfect Information Monte Carlo was tried in [A. Parker, 2005] and together with Hybrid Sampling and GNU Chess engine. It showed promising results, beating a random player in 65% of games. This algorithm was later improved, and it competed at the Eleventh Computer Olympiad in 2006, where it lost to DarkBoard with the result of 6-2. However, chess engines have made tremendous progress since then, and now there are much stronger engines with optimized search and better evaluation function using neural networks. This can leverage the apparent advantages of this approach and significantly improve its performance. Notably, PIMC with Stockfish was used in a computer program Strangefish [**?**], that won NeurIPS 2019 Tournament in Reconnaissance Blind Chess [Escalante and Hadsell, 2019].

4.4 Monte Carlo Tree Search Algorithms

4.4.1 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) was first introduced in [Coulom, 2006]. Former approaches used Monte Carlo techniques for the evaluation of leaves in a Min-Max tree. MCTS combines tree search with Monte Carlo evaluation and does not separate between a Min-Max phase and a Monte Carlo phase. This idea lends MCTS several key attributes that give it an edge over traditional Min-Max algorithms in several games.

Monte Carlo Tree Search is an **aheuristic** algorithm. Since Monte Carlo Tree Search requires little to no domain-specific knowledge, it can be beneficial in games where it is challenging to formulate reliable and useful heuristics (e.g., Go). However, using some domain-specific knowledge can significantly improve performance of Monte Carlo Tree Search [S. Gelly, 2011].

It is an **anytime** algorithm. This means that it can return an action at any moment during strategy computation. In general, it is expected to give better results the longer it is running. This is an advantage over Min-Max

algorithm, although iterative deepening can also make Min-Max into an anytime algorithm.

Finally, MCTS builds an **asymmetric** game tree, where each node represents a state of the game. With the current state being the root of the tree, MCTS iteratively adds more nodes, building a bigger and bigger tree. Although Min-Max can use alpha-beta pruning to prune branches that are sure not to yield the best result, Monte Carlo Tree Search takes this a step further, exploiting better branches and exploring these parts of the tree more than the less preferable ones.

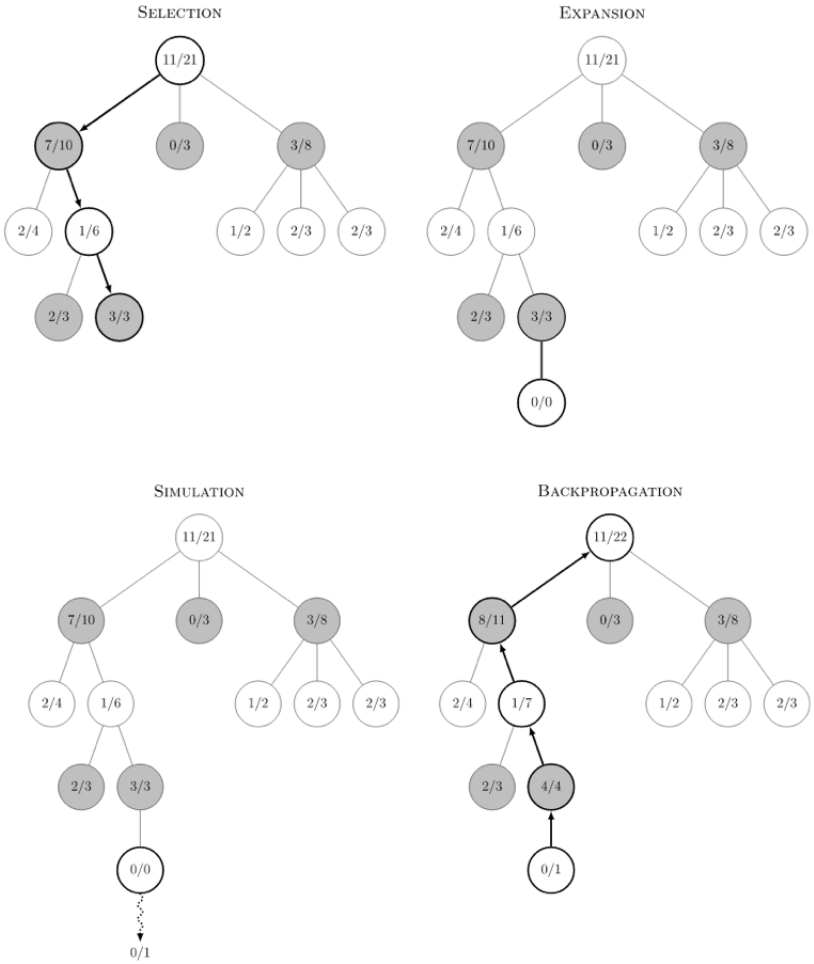


Figure 4.2: The four steps of Monte Carlo Tree Search.

Each iteration of building the tree can be split into four steps:

- **Selection**
  In this step, the algorithm decides which node in the existing tree to expand. Starting in the root, we descend in the built game tree, selecting an appropriate child in every node until we enter a leaf node.

- **Expansion**
  After selection, the algorithm adds one or more sub-tree to the built tree. The most common approach is to add one random child node that is not yet in the tree. However, there are more possibilities, including adding all the missing child nodes or searching a random child node to a certain depth and adding a whole sub-tree at once.

- **Simulation**
  Once a new node is added to the tree, we need to evaluate it. The most common approach is to run a random rollout from the expanded node. This ensures that MCTS is an aheuristic algorithm, and we do not need any domain-specific knowledge whatsoever. However, the rollout can be guided - we can add domain-specific knowledge to it, preventing it from making obvious blunders. This can help to make the simulation result more relevant.

- **Backpropagation**
  We backpropagate the result of the simulation up the tree to the root, updating all the nodes on the path. The result will be added to the information stored in the nodes so it can be used in the next iteration's selection step.

Selection is a task similar to the multi-armed bandit problem. The program has to find a balance between exploration and exploitation - exploring new parts of the game tree versus exploring the parts that look promising so far. A standard way to address this problem is Upper Confidence Bound applied to Trees (UCT) first used in [Kocsis and Szepesvári, 2006]. UCT chooses child nodes maximizing the following expression:

$$U_i = v_i + c\sqrt{\frac{lnN}{n_i}},$$

where v_i is the value of the node i, N is the number of times the parent node has been visited, n_i is the number of times the considered node has been visited, and c is a constant that has to be found experimentally. Low values of c favor exploitation, while higher values favor exploration.

Although Monte Carlo Tree Search is an aheuristic algorithm, a simple heuristic to guide the simulation step can significantly improve this step's precision - for example, not walking into a mate in one in chess or recapturing in Kriegspiel.

4.4.2 Information Set Monte Carlo Tree Search

Information Set Monte Carlo Tree Search (ISMCTS) is a variant of Monte Carlo Tree Search for games with imperfect information. Instead of a tree where nodes represent states, here nodes represent information sets. The reason behind it is that since the player cannot distinguish between the states, we should group evaluations of these indistinguishable states into one node.

At the beginning of each iteration, we choose a determinization, just like in Perfect Iteration Monte Carlo. This sample from the information set will guide the iteration.

- **Selection**
  Selection is the same as in MCTS, except the nodes represent whole information sets instead of specific states. Determinizations from one information set can have different sets of actions available. This is caused by positions of opponent pieces influencing the actions that the player can take in Chess. This means that we now have to deal with a subset-armed bandit problem. This can be solved by replacing the number of times the parent was visited with the number of times the parent was visited and the considered node was available in the UCT formula [Cowling et al., 2012].

- **Expansion**
  There is a slight difference in the expansion step too. When we come across a state whose one or more child states do not have a corresponding information set in the tree, we add the node representing the whole information set.

- **Simulation**
  Simulation runs from the added state as in MCTS.

- **Backpropagation**
  During backpropagation, we need to do a few extra things compared to simple MCTS. We propagate the result of the simulation up the tree back to the root, updating the nodes. However, in this case, it is not enough to know the number of times the parent node was visited, but we need to know how many times an action to a state in the node's corresponding information set was available. This means we also need to update information for sibling nodes of the visited nodes, if there is an action available in the current determinization, that would lead to the information set represented by the sibling node.

The fact that the nodes represent information sets rather than states means that the evaluations of the indistinguishable states are grouped together. This way, the algorithm realizes that there is some information hidden from the player. With this approach, the algorithm can choose actions that reveal information because then we will find ourselves in an information set that allows for more informed decisions.

In the selection step, we discussed only the part when we are selecting the successor of the node where it is the deciding player's move. What about the nodes when the opponent plays? We cannot use the information stored in the nodes of the built tree because this information is not available to the opponent. The simple solution is to use a random action and see to which information set it takes us. This means that the model for the opponent is a random player. It also means that our algorithm will never choose an action to hide information because it has no way of knowing that there is some information hidden to the opponent.

The solution to the problem mentioned above is an algorithm called Multiple Observer Monte Carlo Tree Search (MO-ISMCTS) [Cowling et al., 2012]. In this algorithm, we build two game trees - one from the perspective of our player and one from the opponent's. We move through both trees simultaneously. When it is the deciding player's turn, the action of the determinization is decided according to the information in the node of the deciding player. When it is the opponent's turn, it decides according to its information. Note that the information sets in these trees will be different. Each action chosen by a player will lead to a different node in this player's tree. This does not apply to the other player's tree - the moves will be grouped depending on the information they get from this move.

Information Set Monte Carlo Tree Search supplied with Last Observation Sampling was tested in [Ciancarini and Favini, 2010a] under the name "Approach A". Authors tried to improve the sampling with statistics about pieces locations gathered from more than 12 000 games played on Internet Chess Club. However, these statistics considered piece locations as independent events, thus wholly ignoring relationships such as piece protecting. This resulted in generating improbable boards. To ensure a more realistic simulation, both players did not play entirely random moves in the simulation step. They always tried to recapture and use paws tries when possible since this is a standard strategy used by human players. However, this approach failed to deliver satisfactory results, and it was indistinguishable from a random player.

4.4.3 DarkBoard 2.0

After the poor results of Approach A mentioned in the previous section, authors of [Ciancarini and Favini, 2010a] introduced two more MCTS approaches playing Kriegspiel. Approach B also uses the same statistics collected from ICC games as Approach A. However, this approach never samples specific determinizations from them. Instead, it works with probabilistic umpire messages to guide the selection and simulation. This adds abstraction similar to how human players think about the game. This approach is similar to an abstraction used in real-time strategy games [M. Chung, 2005], where only high-level decisions and high-level responses are simulated because detailed simulation over continuous time would be impossible.

Nodes in the game tree are represented by a triplet of matrices, similar to the probability matrices collected from ICC games. These matrices give us probabilities of a king, pawn, or a piece being on a certain square for a given player at given time in the game. These probabilities and the statistics collected from ICC games drive the prediction of umpire messages.

Simulation is not run untill the end of the game. Instead, it ends after k moves, and the board is evaluated as the number of player's pieces minus the number of the opponent's pieces.

Although the selection is still driven by UCT, the generation of probabilistic umpire messages decides outcomes of player's and opponent's moves. When generating umpire messages, authors make two sets of assumptions.

The first set models the probabilities of outcomes of the player's moves:

- The probability that the opponent controls a square equals the sum of probabilities that the square is controlled by king, pawn, or another piece. However, the presented model does not use a probability, that the square is controlled by the opponent, but rather an expected number of opponent's pieces controlling the square. The expected number of kings and pawns controlling the square is simply calculated as a sum of probabilities that the king or pawn is on squares from which they control the desired square. The expected number of pieces controlling the square is a sum of probabilities that a piece is at a square attacking the desired square, but the probability is multiplied by a coeficient decreasing with distance from desired square. The result is multiplied by a coefficient saying what portion of the opponent's current pieces can actually control a square at once.

- The probability of a move to be legal is a sum of probabilities of the squares on the piece's path to be empty (for a pawn's advance move, this includes the destination square) minus a pin probability (a probability that the move would cause the player's king to be in check). If the king moves, the pin probability is the probability that the opponent controls the destination square. If the piece to move is another piece and it is possible that the piece is protecting the king, the pin probability is approximated as the probability that the piece's starting square is controlled by opponent.
- The probability of capturing a pawn is equal to the probability that the move is legal multiplied by the probability that the destination square is occupied by a pawn.
- The probability of capturing a piece is equal to the probability that the move is legal multiplied by the probability that the destination square is occupied by a piece.
- The probability of the move causing a check is equal to the sum of probabilities that a king is on a square over squares threatened by the moving piece. This probability is again multiplied by a coefficient decreasing with the distance from the moving piece.
- When a square is moved through by a player's piece, all probabilities of enemy pieces on that square are set to 0. When a square is found to be occupied (opponent's capture or player's illegal move), the sum of probabilities of enemy pieces on that square is brought to 1.

The second set of assumptions represents the opponent model. It updates the probabilities of enemy pieces based on umpire messages after the opponent's move:

- When the player captures something, there is a high chance of the moved piece being captured back. The probability that the destination square is controlled is used to determine the chance of retaliation.
- When the player puts the opponent king in check, there is a chance that the moved piece is going to be captured. This probability is assumed to be a constant.
- When the opponent moves, there is a probability that he will capture something. This probability is considered a constant. The victim is chosen depending on the probability of the square being controlled by the opponent. So there is a higher probability that more exposed pieces will be captured than less exposed ones.

- All opponent's pieces are equally likely to be moved. Since the player knows the number of the opponent's pawns and other pieces, the probability of the moved piece being of a certain type can be easily calculated.
- King's movement is modeled as a random walk over squares that are possible destinations for a given square.
- Pawns are modeled as one-way Markov chains.
- The generic piece movement is a random walk over a group of neighboring squares in a certain direction. The probability that a piece ends up on a certain square is lower with a larger number of squares in the square group. There is also a coefficient simulating that not all pieces can move in the analyzed direction.

Approach C is very similar to Approach B, and it capitalizes on the notion already noticed in [A. Parker, 2005] that short-sighted algorithms tend to outperform long-sighted algorithms in Kriegspiel. Approach C assumes that the simulation immediately converges through a weighted average after one move - it simulates all possible umpire messages and their probabilities and computes a weighted average of values of the results. The same evaluation function as in Approach B is used. The assumption of instant convergence is underlined by ditching the average value as a backup operator, replacing it with the maximum node value.

Experiments showed that Approach C is the strongest of the MCTS approaches and even defeats the metaposition player three times more often than it loses to it. This approach won the gold medal with a perfect score at the 14th Computer Olympiad in Pamplona in 2009. It also reached an average of 1750 Elo points after playing more than 7000 games on ICC, being cca 200 points below top human players. The authors praise the algorithm's abstract model of the game and ability to simulate short-term tactical opportunities accurately.

Even with all this success, the authors point out that Approach C can be significantly further improved. MCTS is often improved with game-specific heuristics guiding the selection and simulation steps, while the presented MCTS uses very little domain knowledge. There is also a possibility to use a hybrid simulation between approaches B and C - to simulate the first move like the Approach C does and then continue with a longer B-like simulation for the next moves. Genetic algorithms or other adaptive methods could also improve the assumptions about referee messages.

Unfortunately, the paper [Ciancarini and Favini, 2010a] does not mention any implementation details of Darkboard 2.0. Moreover, many aspects of the algorithm are not described in sufficient depth and some important details

seem to be missing. Here, I list the aspects of the described algorithm, that I find ambiguous:

- The probabilistic model uses many coeficients, that are not described.
- Probabilistic model of the opponent models opponent's pieces' movement along the file, rank and diagonals but does not mention how it models knight movement.
- The probabilistic model of an umpire does not mention the "Pawn tries" message, although the authors use the ICC set of rules.
- The evaluation function takes into account the number of pieces. However, the authors sometimes use the word *piece* in the standard meaning of any Chess piece and sometimes in the meaning of a Kriegspiel piece, meaning any piece except for king and pawn.
- The probabilistic umpire and the evaluation function take into account check-mating the opponent's king. This would suggest that the algorithm has no notion of the actual goal of the game.

Chapter 5

Implementation

5.1 OpenSpiel: A Framework for Reinforcement Learning in Games

OpenSpiel [Lanctot et al., 2019] is an open-source framework for implementing games and algorithms in reinforcement learning and search and planning in games. It supports n-player games, zero-sum and general-sum games, one-shot and sequential games, perfect and imperfect information games, turn-taking, and simultaneous-move games.

OpenSpiel contains implementations of more than 40 different games of all mentioned types, many different algorithms for playing games, and tools for common evaluation metrics. It provides convenient documentation for installation and contributing to the code base - either implementing new games or new algorithms. The community developing OpenSpiel is alive, and new contributions are added every week.

The framework uses a combination of C++ and Python. Games are implemented in C++ and wrapped in Python. Algorithms can be implemented in C++ and Python, with the APIs in both languages being almost identical.

5.2 Stockfish

Stockfish [Sto, 2004] is a strong chess engine. According to Computer Chess Rating List (CCRL) [CCR, 2015], Stockfish is currently the strongest computer Chess player in the world. Stockfish uses Min-Max algorithm with alpha-beta pruning and transposition tables. It also has advanced heuristics for recognizing uninteresting branches. For board evaluation, it can use either a classic evaluation function handcrafted by chess experts or an Efficiently Updatable Neural Network (NNUE).

Computer Chess rating List measures chess engine strength in several time control settings. All the games are played with pondering turned off (the engine cannot think while it is the opponent's turn). They can use up to 12-move opening books and up to 6 piece endgame tablebases. With the repeating time control 40/15 (15 minutes for the first 40 moves, then 15 minutes for the next 40 moves and so on) Stockfish ranks first in the list with elo of 3554, followed by Fat Fritz with an elo of 3528.

Stockfish uses UCI (Universal Chess Interface), which is a stateless protocol for chess engines. It allows all sorts of GUIs and web servers to use Stockfish to play against humans or other computers. UCI supports things like time control with increment or time per move, pondering on a selected move, and setting engine-specific options. I implemented this protocol in OpenSpiel to support any UCI compliant engine. I decided to use Stockfish for being an open-source project and for its strength.

5.3 Kriegspiel

Although OpenSpiel contained the implementation of dozens of different games, unfortunately it did not contain the implementation of Kriegspiel. As a part of the master thesis, I had to provide a Kriegspiel implementation of the game and a non-trivial implementation of observations in C++. I did the same for another mentioned imperfect information chess variant - Dark Chess.

Both implemented games passed through vigorous code reviews to comply with the OpenSpiel repository's rules and practices. Both implemented games were merged in the repository and can be used by the research community. I believe this will help in future research concerning imperfect information chess variants and draw the attention of other researchers to them.

5.4 Information Set Representation

5.4.1 Last Observation Sampling

Implementation of this information set representation is similar to the one mentioned in [Ciancarini and Favini, 2010a]. Exact locations of the sampling player's pieces' locations are known. The opponent pieces are randomly placed on the board so that they follow the distribution for the player's color and the turn number. These statistics are collected from more than 8000 games of Kriegspiel played on ICC. The sample tries to comply with the last umpire message received after the opponent's move. It also remembers how many pieced and pawns the player has captured, so that the generated states have the correct amount of them.

5.4.2 All Observation Sampling with Pool

This implementation keeps a pool of possible states, and at the beginning of the player's turn, it updates the pool with the received umpire messages. The pool is given maximum size. When the pool has more elements after the update than the maximum size, it will get rid of random samples until the size is again in the limit.

If the algorithm finds out that none of the samples in the pool was complying with the current information set, it will use the LOS algorithm from the previous section to sample new states and fill the pool. This new pool is compatible with the last observation only. Initial testing showed, that this pool has a low quality and it very often does not last more than a few moves.

5.4.3 Hybrid Sampling

Hybrid sampling uses the implementation of AOSP from the previous section to store a pool of compatible states. Additionally, it uses the implementation of LOS to refill the pool when its size drops below half of the pool's maximum size, as recommended in [A. Parker, 2005].

I added two additional checks that have to be satisfied to refill the pool

so that the states provided by LOS do not unnecessarily lower the quality of the pool:

1. The AOSP already removed some of the compatible states from the pool because of the pool getting too large. This way, it is certain that no new states that are compatible only with the last observation are unnecessarily added to a pool that contains the whole information set.
2. LOS is not used when the opponent has only a few pieces left. With fewer opponent pieces on the board, the maximum size of the information set gets lower. When it is low enough, it does not make sense to generate samples compatible with only the last observation because there is a good chance that the pool contains the whole information set.

This approach wa evaluated to perform the best of the three approaches in [A. Parker, 2005]. Therefore, all tested implementations using statistical sampling use Hybrid Sampling.

5.5 DarkBoard

This thesis contains an implementation of the second, stronger version of DarkBoard using Information Set Monte Carlo Tree Search. All the information about implementation was taken from the paper [Ciancarini and Favini, 2010a]. Unfortunately, the paper does not contain all the information needed for the exact implementation that reached the results presented in the paper.

For extraction of the ICC Kriegspiel games statistics, I used the games that are in the PGN format in the attachment. There are more than 8000 games of Kriegspiel (Wild 16) played by players of different strength.

5.6 Random Player

As a baseline algorithm, I use uniformly random action sampling algorithm. It chooses a random move from the moves that seem legal to the player.

Another implementation uses a simple heuristic already mentioned in this thesis and in [Ciancarini and Favini, 2010a] and [Bolognesi and Ciancarini, 2004]. The simple heuristic uses a very common strategy in Kriegspiel, and that is recapturing whenever possible. Thanks to umpire messages, the algorithm knows when its piece was captured, and it will always recapture if a recapture move is possible.

5.7 PIMC with Stockfish

Perfect Information Monte Carlo was implemented as described in Section 4.3. It needs an information set representation that allows sampling random states from the set. Therefore, the implementation supports any of the statistical sampling information set representations.

The algorithm evaluates each sampled state with Stockfish using the UCI protocol. Unfortunately, the protocol has limitations concerning evaluating particular legal moves in the state. We can either fully evaluate each available move or we can evaluate the sampled state receiving only the best move and not an evaluation for every move. Evaluating each move would eliminate Stockfish's advanced branch pruning, which would significantly slow down the evaluation. Therefore, this implementation only evaluates a particular move based on the number of sampled states in which the move was selected as the best move. This is also how it was done in [A. Parker, 2005].

The algorithm is given a parameter determining the minimal state evaluation time. The algorithm first checks whether the information set is small enough to evaluate all the states. Then it allocates equal time for each state so that all of them can be evaluated. A second approach has to be used when there are too many states Then it samples random states until time runs out and evaluates each of them for the minimal evaluation time. In LOS, we do not have a pool of states to sample from, so we always have to use the second approach.

5.8 ISMCTS

Although OpenSpiel already contains an implementation of ISMCTS, the implementation's representation of information sets and choosing an opponent

model is not general enough to allow non-trivial custom implementations. Therefore, I implemented the algorithm basically from scratch, only reusing a method selecting child nodes with the UCT formula.

The basic ISCMTS implementation uses the heuristic random player for simulation and for the opponent model. I also implemented the more sophisticated version of ISMCTS - Multiple Observer Information Set Monte Carlo Tree Search. Although the opponent model in MO-ISMCTS is more sophisticated, the simulation still uses a random rollout guided by the mentioned simple heuristic.

Chapter 6

Experiments

6.1 Metacentrum

To play large amounts of games to evaluate the algorithms, I used computational cluster Metacentrum for my experiments. Each algorithm is given 1 machine with 1 CPU and 4GB RAM for its computations. Using Metacentrum allowed me to run many games at once on its large clusters, therefore simulating in total tens of thousands of games to ensure statistical significance of the results.

6.2 Statistics

To show statistical significance of the results of following experiments, I compute confidence intervals. I use 95% confidence intervals, meaning that the true result lies in the interval with 95% probability.

The formula for confidence interval is:

$$\overline{X} \pm Z\frac{s}{\sqrt{n}},$$

where X is the mean of the measured results, s is the standard deviation, n is the number of measured results and Z is the Z-value (for 95% confidence

intervals it is 1.96). The formula for computing standard deviation is as follows:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{X})^2},$$

where N is the number of measured results and X is the mean of measured results.

In order to apply this formula, I assume the measured outcomes of the games can be approximated by the normal distribution.

6.3 Results

In this section, I present the results of the experiments I ran on Metacentrum. First, I try to find hyperparameters for all the implemented algorithms, that yield the best results against the random player. All the algorithms are then tested with the best found settings against each other. Each pair of algorithms, in the round-robin tournament and in finding hypermarameters, plays with the time settings of 2 seconds per move.The expected outcome is presented with a 95% confidence interval.

In Tables 6.1, 6.2, 6.4, 6.5, 6.3, 6.6 I look for optimal parameters by letting the algorithms play against a random player. The parameter with the highest expected outcome has its expected outcome written in bold. The parameter is then used in the final round-robin tournament 6.7.

UCT c	0.25	0.5	1.0	2.0	5.0
Wins	823	844	803	815	810
Draws	1083	1064	1101	1082	1096
Losses	94	92	96	93	94
Expected outcome	0.6822	**0.6880**	0.6768	0.6805	0.6790
Confidence Interval	±0.0125	±0.0125	±0.0125	±0.0125	±0.0125

Table 6.1: Finding UCT exploration constant for the ISMCTS algorithm. The best expected outcome is written in bold.

UCT c	0.25	0.5	1.0	2.0	5.0
Wins	785	752	734	724	711
Draws	1148	1146	1171	1168	1184
Losses	67	102	95	104	105
Expected outcome	**0.6795**	0.6625	0.6598	0.6528	0.6515
Confidence Interval	±0.0119	±0.0124	±0.0123	±0.0123	±0.0123

Table 6.2: Finding UCT exploration constant for the MO-ISMCTS algorithm. The best expected outcome is written in bold.

In Table 6.1 and 6.2 I look for the UCT exploration constant to guide the selection step in ISMCTS algorithms.

Board Eval Time [ms]	1	5	10	20	50
Wins	933	946	966	968	949
Draws	67	54	33	32	51
Losses	0	0	1	0	0
Expected outcome	0.9655	0.9730	0.9825	**0.9840**	0.9745
Confidence interval	±0.0077	±0.0070	±0.0059	±0.0055	±0.0068

Table 6.3: Finding evaluation time for each determinization in PIMC. The best expected outcome is written in bold.

In Table 6.3 I try to find the optimal time for evaluation of each sampled state in Perfect Information Monte Carlo algorithm.

Eval Time [ms] | UCT c		0.25	0.5	1.0	2.0	5.0
1	Wins	508	397	469	488	473
	Draws	1468	1576	1525	1502	1518
	Losses	24	27	6	10	9
	Expected outcome	0.6210	0,5925	0.6158	0.6195	0,6160
	Confidence Iterval	±0.0098	±0.0092	±0.0094	±0.0096	±0.0095
5	Wins	550	668	619	635	625
	Draws	1423	1318	1367	1305	1311
	Losses	27	14	14	60	64
	Expected outcome	0.6308	0.6630	0.6513	0.6438	0,6402
	Confidence Iterval	±0.0103	±0.0106	±0.00104	±0.0113	±0,0113
10	Wins	633	673	655	642	617
	Draws	1338	1310	1311	1351	1326
	Losses	29	17	34	54	57
	Expected outcome	0.6510	0.6640	0.6553	0.6443	0,6400
	Confidence Iterval	±0.0107	±0.0107	±0,0106	±0.0111	±0,0111
20	Wins	828	803	744	732	720
	Draws	1150	1178	1234	1242	1250
	Losses	22	19	22	26	30
	Expected outcome	**0.7015**	0.696	0.6825	0,6765	0.6725
	Confidence Iterval	±0.0112	±0.0111	0,0110	±0.0111	±0.0111
50	Wins	678	628	616	605	593
	Draws	1240	1321	1317	1327	1342
	Losses	82	51	67	69	75
	Expected outcome	0.6490	0.6442	0.6373	0.6338	0.6293
	Confidence Iterval	±0.0118	±0.0111	±0.0113	±0.0113	±0.0113

Table 6.4: Finding UCT exploration constant and evaluation time for each state in ISMCTS with Stockfish. The best expected outcome is written in bold.

In Tables 6.4 and 6.5 I perform a grid search on two hyperparameters of ISMCTS algorithms improved by using Stockfish in simulation step. Each cell of the table contains the number of wins, draws, losses and the expected outcome. Rows contain results for a parameter saying how much time is spent on a single board evaluation in simulation step. Columns contain results for a given value of UCT exploration constant.

Eval Time [ms]	UCT c	0.25	0.5	1.0	2.0	5.0
1	Wins	1106	1092	1141	1086	1074
	Draws	853	860	811	858	863
	Losses	41	48	48	56	63
	Expected outcome	0.7663	0.7610	0.7733	0.7575	0.7528
	Confidence Interval	0.0118	0.012	0.0119	0.0121	0.0123
5	Wins	1204	1240	1228	1190	1110
	Draws	780	734	728	768	845
	Losses	16	26	44	42	45
	Expected outcome	0.7970	**0.8035**	0.7960	0.7870	0.7657
	Confidence Interval	±0.0111	±0.0113	±0.0117	±0.0114	±0.0117
10	Wins	1087	1163	964	988	762
	Draws	871	755	982	967	995
	Losses	42	82	54	45	52
	Expected outcome	0.7613	0.7703	0.7275	0.7358	0.6965
	Confidence Interval	±0.0118	±0.0126	±0.0120	±0.0119	±0.0119
20	Wins	880	833	986	886	896
	Draws	1089	1047	930	1021	982
	Losses	93	120	84	93	93
	Expected outcome	0.6813	0.6783	0.7255	0.6983	0.7040
	Confidence Interval	±0.0125	±0.0130	±0.0126	±0.0126	±0.0127
50	Wins	568	576	514	615	765
	Draws	1324	1222	1306	1228	1090
	Losses	108	202	180	157	145
	Expected outcome	0.6150	0.5935	0.5835	0.6145	0.655
	Confidence Interval	±0.0123	±0.0130	±0.0124	±0.0127	±0.0131

Table 6.5: Finding UCT exploration constant and evaluation time for each state in MO-ISMCTS with Stockfish. The best expected outcome is written in bold.

UCT c	0.5	1.0	2.0	5.0	10.0
Wins	470	522	411	387	365
Draws	1484	1438	1523	1545	1562
Losses	46	40	66	68	73
Expected outcome	0.6055	**0.6205**	0.58625	0.5798	0.5730
Confidence Interval	±0.0102	±0.0103	±0.0100	±0.0099	±0.0097

Table 6.6: Finding UCT exploration constant for Darkboard 2.0. The best expected outcome is written in bold.

The final Table 6.7 contains expected outcomes of games between each pair of implemented algorithms using the best found parameters. The value is the expected outcome for the row player with a 95% confidence interval.

	Uniform Random	Random with Heuristic	ISMCTS	ISMCTS with SF	MO-ISMCTS	MO-ISMCTS with SF	PIMC with SF	Darkboard 2.0
Uniform Random	X	0.4650 ±0.0069	0.3178 ±0.0125	0.2985 ±0.0118	0.3205 ±0.0119	0.1965 ±0.0113	0.0160 ±0.0055	0.3795 ±0.0103
Random with Heuristic	0.5350 ±0.0069	X	0.4530 ±0.0097	0.4312 ±0.0092	0.4490 ±0.0098	0.4270 ±0.0092	0.0457 ±0.0064	0.3566 ±0.0096
ISMCTS	0.6822 ±0.0125	0.5470 ±0.0097	X	0.4107 ±0.0102	0.5038 ±0.0149	0.3987 ±0.0128	0.0335 ±0.0059	0.5019 ±0.0112
ISMCTS with SF	0.7015 ±0.0118	0.5688 ±0.0092	0.5893 ±0.0102	X	0.5945 ±0.0121	0.4125 ±0.0111	0.0769 ±0.0081	0.5224 ±0.0063
MO-ISMCTS	0.6795 ±0.0119	0.5510 ±0.0098	0.4962 ±0.0149	0.4055 ±0.0121	X	0.4068 ±0.0121	0.0380 ±0.0061	0.5416 ±0.0102
MO-ISMCTS with SF	0.8035 ±0.0113	0.5730 ±0.0092	0.6013 ±0.0128	0.5875 ±0.0111	0.5932 ±0.0121	X	0.1080 ±0.0099	0.5374 ±0.0113
PIMC with SF	0.9840 ±0.0055	0.9543 ±0.0064	0.9665 ±0.0059	0.9231 ±0.0081	0.9620 ±0.0061	0.8920 ±0.0099	X	0.9468 ±0.0062
Darkboard 2.0	0.6205 ±0.0103	0.5434 ±0.0096	0.4981 ±0.0112	0.4776 ±0.0063	0.4584 ±0.0102	0.4426 ±0.0113	0.0532 ±0.0062	X

Table 6.7: Round-robin Tournament of all implemented algorithms

6.4 Discussion

6.4.1 Darkboard 2.0

The performance of Darkboard using ISMCTS against the random player, although better than all the previous ISMCTS algorithms, is still lower than expected. The only information about the performance of the original implementation described in [Ciancarini and Favini, 2010a] is that it ranks 100 elo higher than the first version of Darkboard (meaning an expected win rate of 66%), which had a 95% win rate against a random player.

The reasons for the worse performance of my implementation might be some of the following:

1. The original implementation used a kind of root parallelization and was tested running on 4 CPUs.
2. I used a different database of games from ICC. This means a slightly different opponent model.
3. The description of the algorithm in the paper does not mention any implementation details, and in many places, the description of the algorithm is vague or misinterpretable. This left room for some changes that might have widened the gap between these implementations.

The algorithm performed onlyslightly better than heuristic random player. One of the reasons could be the already mentioned missing concept of a check-mate in the probabilistic umpire model and in the evaluation function. The algorithm does a good job capturing opponent pieces, but fails in actually check-mating the opponent's king. The higher performance than random players is probably caused by a higher probability of check-mating the opponent's king when one has a material advantage, even when one plays randomly.

6.4.2 Information Set Monte Carlo Tree Search

All Information Set Monte Carlo Tree Search algorithms perform best with low values of the UCT exploration constant, meaning they favor exploitation

over exploration.

Although [Ciancarini and Favini, 2010a] stated that ISMCTS's performance was indistinguishable from the random player, the results in 6.1 show that it plays better than the random player. The reason might be that I used a more sophisticated information set representation - a combination of hybrid sampling from [A. Parker, 2005] and the sophisticated LOS using statistics from ICC.

With the ISMCTS algorithms performing only slightly better than a random player, the performance is not satisfactory even on its best settings. The reason why random rollout performs this poorly is that Kriegspiel is a challenging game to win. The random player (although guided by a heuristic) has a minimal chance to win a game, even if it has a significant material and positional advantage. Therefore, most of the simulations end with a draw, not offering any information for the ISMCTS to work with.

When comparing the results of MO-ISMCTS and ISMCTS with random rollout simulation, we can notice that ISMCTS performs better. This seems counter-intuitive, since MO-ISMCTS uses a more sofisticated opponent model. When playing against a random player, ISMCTS has a more accurate opponent model, because the opponent model is a random player. One explanation for MO-ISMCTS's poor performance might be that it solely relies on the simulation step to evaluate moves. As I mentioned, these simulations are not accurate. Meanwhile, ISMCTS uses heuristic random player as opponent model in selection phase. This means that ISMCTS expects the opponent to make somewhat reasonable actions already in selection phase, while MO-ISMCTS must converge to it through many simulations.

ISMCTS algorithms using Stockfish instead of a classical random rollout simulation improved the performance of a simple ISMCTS and MO-ISMCTS. In the case of ISMCTS, a relatively high value of board evaluation time performed the best, meaning that realies more on the board evaluation that on the search. MO-ISMCTS performed best with a lower value, since the search is more sophisticated.

MO-ISMCTS with Stockfish evaluation performed better than ISMCTS with Stockfish. This shows that a more accurate state evaluation in the simulation step causes the algorithm to converge more quickly to a reasonable solution. Since the opponent model in MO-ISMCTS also relies on the results of simulations, this makes it stronger, resulting in much better overall results.

6.4.3 Perfect Information Monte Carlo

In Perfect Information Monte Carlo (PIMC) using Stockfish for state evaluation, we need to decide how much time we want the chess engine to spend on each state. The search for this hyperparameter is shown in the table 6.3. Higher times have better results than lower times. This means that it is preferable to have a more precise evaluation of the sampled states at the cost of only evaluating a handful of them. This makes sense when we consider the actual size that the information set can grow to in the game of Kriegspiel. No matter how short the evaluation time for each state would be, we would never be able to evaluate a significant portion of the states in the information set.

When choosing the best move from the statistics gathered, the used version chooses the one evaluated by Stockfish as the best move the most times. Although the implementation allows for more sophisticated statistics to be gathered, I discarded them after not performing well during some initial testing. Evaluation of all moves in a state is computationally very difficult due to limitations of the used UCI protocol. To gather the score of every move, we need to generate all legal moves, apply each of them and evaluate the resulting board. This way we need to give Stockfish fixed time for evaluating each move, limiting Stockfish's ability to quickly prune obviously bad moves.

The results against the random player are surprisingly good since Kriegspiel is known for being a challenging game to win. In [A. Parker, 2005] a very similar version of PIMC player had a win rate of only 65%. This player used GNU-chess engine, which was one of the strongest open-source chess engines at the time. My PIMC player has significantly better results against the random player even though it plays under much harsher time management - I used 2 seconds per move. In comparison, the authors of the mentioned paper used 30 seconds per move. Although both algorithms use Hybrid Sampling, my implementation improves the LOS part with statistics from more than 8000 ICC games. The authors of the paper do not mention any heuristic for Last Observation Sampling, which might be one of the reasons why my algorithm performs better. Another, and probably more significant, improvement my algorithm has, is a stronger chess engine backing the PIMC algorithm. Chess engines came a long way since the original paper came out. This again shows that a solid and precise evaluation of particular states does wonders for the algorithm's strength.

However, the weaknesses of ignoring the problem of hidden information are apparent when playing against stronger opponents. PIMC plays too aggressively, does not cover its pieces, and tries to go for the checkmate. When analyzing the games closer, I saw that PIMC often attacks the opponent half of the board early in the game with the queen. This reckless behavior

works against weak players who are not able to use it to their advantage. This strategy is probably caused by the fact that since only a small fraction states are sampled. The algorithm often does not anticipate all the dangers of opponent's pieces being protected and looses the queen.

Chapter 7

Conclusion

This thesis examined using strong chess engine for evaluation as a heuristic for algorithms playing Kriegspiel, an imperfect information Chess variant, where information given to players is sparse and quickly vanishes.

First, I introduced and analyzed the game of Kriegspiel and implemented it in the OpenSpiel framework. I also mentioned other imperfect information Chess variants and implemented Darkchess, a game very similar to Kriegspiel, but it offers more information to players.

I reviewed information set representations introduced and used in different papers. I proposed a new way to represent information sets in Kriegspiel by combining two of the representations discussed and implemented a total of three information set representations.

I reviewed algorithms for playing imperfect information games that have been used to play Kriegspiel. I implemented them in the OpenSpiel framework and suggested a way to improve them by using Stockfish, a strong chess engine, for evaluation of determinizations. This resulted in having implemented five algorithms from the Monte Carlo family of algorithms. Moreover, I implemented three baseline algorithms. Two of them are random players - one playing uniformly random moves and one guided by a simple heuristic. As the last algorithm, I implemented the strongest known algorithm for playing Kriegspiel.

Experiments showed that the proposed information set representation performs better than the ones found in the literature so far. Adding the evaluation of determinizations by a strong chess engine shows promise since the algorithms performed better than their versions with traditional domain-

independent techniques. Moreover, Perfect Information Monte Carlo performs much better with modern Stockfish than with a weaker chess engine years ago. This shows that precise evaluation of determinizations plays a significant role in the performance of algorithms playing imperfect information games, even in games with very little information like Kriegspiel.

Appendix A

Bibliography

[Sto, 2004] (2004). Stockfish. `https://stockfishchess.org`. [Online; accessed 18-May-2021].

[CCR, 2015] (2015). Computer chess rating list. `http://ccrl.chessdom.com/ccrl/4040/`. [Online; accessed 18-May-2021].

[A. Parker, 2005] A. Parker, D. Nau, V. S. (2005). Game-tree search with combinatorially large belief states. *roc. 19th Int. Joint Conf. on Artificial Intelligence(IJCAI05)*, pages 254–259.

[Bolognesi and Ciancarini, 2004] Bolognesi, A. and Ciancarini, P. (2004). Searching over metapositions in kriegspiel. *Computer and Games 04*, Lecture Notes in Artificial Intelligence.

[Campbell et al., 2002] Campbell, M., Hoane, A., and hsiung Hsu, F. (2002). Deep blue. *Artificial Intelligence*, 134(1):57–83.

[Ciancarini and Favini, 2010a] Ciancarini, P. and Favini, G. P. (2010a). Monte carlo tree search in kriegspiel. *Artificial Intelligence*, 174(11):670–684.

[Ciancarini and Favini, 2010b] Ciancarini, P. and Favini, G. P. (2010b). Playing the perfect kriegspiel endgame. *Theoretical Computer Science*, 411(40):3563–3577.

[Coulom, 2006] Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *5th International Conference on Computer and Games*.

[Cowling et al., 2012] Cowling, P., Powley, E., and Whitehouse, D. (2012). Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and Ai in Games*, 4:120–143.

[Escalante and Hadsell, 2019] Escalante, H. J. and Hadsell, R. (2019). Proceedings of the neurips 2019 competition and demonstration track. 123.

[Ferguson, 1992] Ferguson, T. S. (1992). Mate with bishop and knight in kriegspiel. *Theoretical Computer Science*, 96(2):389–403.

[Ginsberg, 2001] Ginsberg, M. L. (2001). Gib: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 14:303–358.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. *Machine Learning: ECML*, 2006:282–293.

[Lanctot et al., 2019] Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., Hennes, D., Morrill, D., Muller, P., Ewalds, T., Faulkner, R., Kramár, J., Vylder, B. D., Saeta, B., Bradbury, J., Ding, D., Borgeaud, S., Lai, M., Schrittwieser, J., Anthony, T., Hughes, E., Danihelka, I., and Ryan-Davis, J. (2019). OpenSpiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453.

[Long et al., 2010] Long, J., Sturtevant, N. R., Buro, M., and Furtak, T. (2010). Understanding the success of perfect information monte carlo sampling in game tree search.

[M. Chung, 2005] M. Chung, M. Buro, J. S. (2005). Monte carlo planning in rts games. *Kendall, G., Lucas, S. (eds.) Proc. IEEE Symposium on Computational Intelligence and Games, Colchester, Essex*, (2):117–124.

[Moravčík et al., 2017] Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., and Bowling, M. (2017). Deepstack: Expert-level artificial intelligence in heads-up nolimit poker. *Science*, 14356(6337):508–513.

[Newman et al., 2016] Newman, A. J., Richardson, C. L., Kain, S. M., Stankiewicz, P. G., Guseman, P. R., Schreurs, B. A., and Dunne, J. A. (2016). Reconnaissance blind multi-chess: an experimentation platform for ISR sensor fusion and resource management. 9842:62 – 81.

[P. Ciancarini, 2007] P. Ciancarini, G. F. (2007). Representing kriegspiel states with metapositions. *roc. 20th Int. Joint Conf. on Artificial Intelligence(IJCAI-07)*, pages 2450–2455.

[S. Gelly, 2011] S. Gelly, D. S. (2011). Monte-carlo tree search and rapid action value estimation in computer go. *Artificial Intelligence*, 175:1856–1875.

[Sakuta, 2001] Sakuta, M. (2001). Deterministic solving of problems with uncertainty. *PhD thesis*.

[Sheppard, 2002] Sheppard, B. (2002). World-championship-caliber scrabble. *Artificial Intelligence*, 134(1):241–275.