Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Heuristic evaluation functions for imperfect-information chess

Matěj Šesták

Supervisor: Dominik Andreas Seitz, MSc.
May 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Šesták Matěj** Personal ID number: **465549**

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

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Heuristic Evaluation Functions for Imperfect-Information Chess

Bachelor's thesis title in Czech:

Heuristické evaluační funkce pro šachy s neúplnou informací

Guidelines:

Depth-limited look-ahead search in conjunction with value functions has had large success both in perfect- as well as in imperfect-information games. Representing game states in imperfect-information chess variants pose a significant challenge due to their sparse information and hence increase the difficulty for finding optimal strategies significantly. As part of the thesis the student is expected to:
- Review and embed preexisting research around value functions.
- Implement imperfect-information chess variants and a method for representing their game states.
- Train heuristic evaluation functions for a strategically interesting set of endgame situations.
- Experimentally showcase the quality of the strategies produced by the depth-limited equilibrium solving algorithm using it.

Bibliography / sources:

[1] Dominik Seitz, Vojtěch Kovařík, Viliam Lisý, Jan Rudolf, Shuo Sun, Karel Ha, Value Functions for Depth-Limited Solving in Imperfect-Information Games beyond Poker. arXiv 1906.06412, 2019.
[2] Moravčík, Matej et al. "DeepStack: Expert-Level Artificial Intelligence in Heads-up No-Limit Poker." Science 356.6337 (2017): 508–513. Crossref. Web.
[3] Paolo Ciancarini, Gian Piero Favini, Monte Carlo tree search in Kriegspiel, Artificial Intelligence, Volume 174, Issue 11, Pages 670-684, 2010.

Name and workplace of bachelor's thesis supervisor:

MSc. Dominik Andreas Seitz, Department of Computer Science, FEE

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

Date of bachelor's thesis assignment: **10.01.2020** Deadline for bachelor thesis submission: _____

Assignment valid until: **30.09.2021**

MSc. Dominik Andreas Seitz	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. Assignment receipt

The student acknowledges that the bachelor'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 bachelor'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 Dominik Seitz for regular and useful consultations, helpful comments, and discussions, that helped me to finish this thesis. Whenever I ran into a problem, he was ready to provide valuable insights or steer me in the right direction.

Last but not least, I would like to thank my family for their constant support during my studies and work on this thesis.

Thank you.

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 theses.

Prague, May 22, 2020

Abstract

Using value function in depth-limited solving of perfect information games produced super-human play in several domains, like chess or Go. In the domain of large imperfect information games, research mainly focused on the game of poker, which possesses nice properties, and new methods were not used and tested on other domains.

In this thesis, we present the use of value functions in imperfect information chess. We design an abstraction model for imperfect information chess variants, Kriegspiel and Darkchess, and empirically show its usability by computing approximate Nash equilibrium by a modified CFR-D algorithm. We test the use of a convolutional neural network to reduce the necessity of a domain-specific model. Finally, we take a step towards a real-time playing by using multiple neural networks at different depths as value functions.

Keywords: game theory, imperfect information games, chess, kriegspiel, darkchess, value function, CFR, depth-limited solving

Supervisor: Dominik Andreas Seitz, MSc.

Abstrakt

Využití evalučních funkcí v řešení her s úplnou informací vedlo k dosažení nadlidských výkonů v několika hrách, jako šachy nebo Go. V oblasti velkých her s neúplnou informací, výzkum mířil především na poker, který má dobré vlastnosti, a nové metody nebyly využity a testovány na jiných hrách.

V této práci předvedeme použití evaluačních funkcí v šachách s neúplnou informací. Navrhli jsme abstrakční model pro dvě varianty těchto šachů, Kriegspiel a Darkchess, a empericky předvedli jeho využití dosažením přibližné Nashovy rovnováhy minimalizováním upraveným CFR-D algoritmem. Otestovali jsme použití kovolučních neuronových sítí, které snižují potřebu herně specifického modelu. Nakonec jsme učinili krok k hraní v reálném čase použitím několika neuronových sítí v různých herních úrovních.

Klíčová slova: teorie her, hry s neúplnou informací, šachy, kriegspiel, darkchess, evaluační funkce, CFR, řešení s omezenou hloubkou

Překlad názvu: Heuristické evaluační funkce pro šachy s neúplnou informací

Contents

Figures

Tables

Chapter 1

Introduction

The use of games to measure advances in the research of artificial intelligence has been around for a long time. With the rise of computing power, computer agents for many perfect information games already surpassed the human grandmaster level. The first notable accomplishment came in the classical board game Chess when IBM's DeepBlue [9] defeated world champion Garri Kasparov, only with pure search power. The next milestone was the game Go: the algorithm AlphaGo [25] convincingly beat the strongest grandmaster at the time, Lee Se-Dol. It was built around a system using a neural network as an evaluation function, which proved to be crucial in this large game. These games and others share the property that all players know all information about the game, aside from the opponent's strategy.

In recent years there have also been significant breakthroughs in solving large imperfect information games. Most research focus went into different variants of poker and recently achieved superhuman play. In the domain of two-player Heads-up No-Limit Texas Hold-em, DeepStack [19] algorithm defeated 30 professional poker players, and recently agent called Pluribus [4] became first to consistently win against human players in a six-player version of poker. Outside of poker, there has been advancement in cooperation games with unknown and uncertain teammates, in which an algorithm DeepRole [22] excelled.

These algorithms were able to achieve such a good result by combining the established method of counterfactual regret minimization [28] and neural network used as evaluation functions for depth-limited search. In this thesis, we analyzed this approach for two variants of imperfect information chess, Kriegspiel and Darkchess. Both games have unique properties to previously explored domains; for example, in both is very sparse common knowledge, which quickly loses value, and in Kriegspiel, some action players can choose from are illegal.

We designed a novel abstraction for both games, which allowed depth-limited search and the use of value functions. We tested our method in several endgames and were able to approximate Nash equilibrium by a modified CFR-D algorithm [8]. We tested the use of convolutional neural networks to reduce the need for a domain-specific model, a concept previously only used for perfect information games. Finally, we provide a step towards

real-time play by stacking multiple neural networks at different depths as value functions.

This thesis is divided into six chapters. In Chapter 2, we provide an overview of relevant concepts in game theory and neural networks. In Chapter 3, we discuss previous work done in solving imperfect information games, both chess, and others. In Chapter 4, we provide a detailed analysis of the properties of Kriegspiel and Darkchess and provide our designed abstraction for them. In Chapter 5, we provide an empirical study of the use of value functions for depth-limited search in imperfect information chess. Finally, in Chapter 6, we provide an overview of the thesis and suggest ideas for future work.

Chapter 2

Background

This chapter gives a brief introduction to the background of this thesis. First, we introduce notations for imperfect information extensive form games. Then, we discuss the method of counterfactual regret minimization and game solving using decomposition. Finally, we provide a brief overview of neural networks.

2.1 Extensive form game

An *extensive form game* (in the rest of the text is used abbreviation EFG) represents a game with a sequence of action taken by players in turns in oppose to actions taken by players simultaneously. Game is then model as a game tree. A node in the tree represents a chance event (a die roll, card deal, etc.) or state of the game (player's turn, information available to players, board position). An edge represents an outcome of a chance event or an effect of playing action by a player resolving in a new game state. The root node of the tree represents the start of the game, and each leaf node represents the termination of a game.

An *imperfect information extensive form game* G is represented as a tuple $(N, H, Z, A, \rho, u_i, I)$ where:

- $N = \{1, \ldots, n\}$ is the set of all players;
- H is a set of histories, h is particular sequence of action, starting from root of the tree, we denote the fact, that for $g, h \quad H$, g is equal to or prefix of h as g @ h;
- Z is a set of terminal histories, $Z \quad H$ (those $z \quad H$ that are not a prefix of any other history);
- A is a set of actions, $A(h) = \{a|(h, a) \quad H\}$ are the actions available after non-terminal history $h \quad H$, $A(I)$ are actions available in information set $I \quad I_i$;
- $\rho : H \setminus Z \rightarrow N$ is a player function, it assigns an acting player $i \quad N$ to each non-terminal history;
- $u_i : Z \rightarrow \mathsf{R}, i \quad N$ is a real-valued utility function for player i in terminal nodes Z;

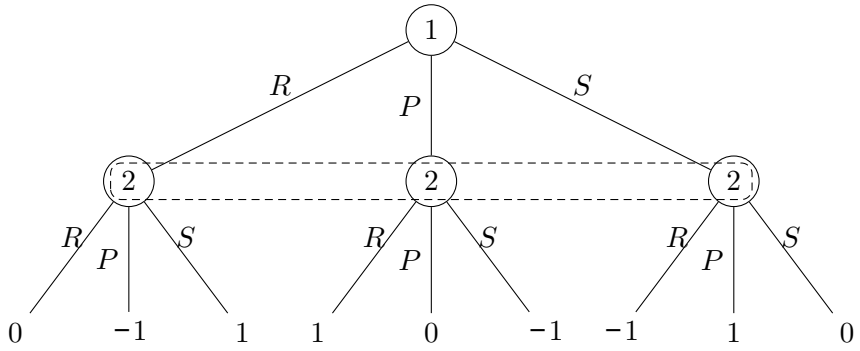

Figure 2.1: Simple game of rock, paper, scissors as imperfect information extensive form game.

- $I = (I_i)$ for $i \quad N$ is the information-partition, $I_i \quad H_i$, if $g, h \quad H_i$ and $g, h \quad I_i$, than player i cannot distinguish between those two states.

A perfect information extensive form game is the same tuple as imperfect information, but without the I as in every information set, there is only one state. A *zero-sum game* is a game which for every terminal node of EFG satisfies $\sum_{i \ N} u_i = 0$.

Example of is Rock, Paper, Scissors in Figure 2.1, which can be modeled as a zero-sum imperfect information game even though the play between humans is simultaneous. Number in a node represents the player taking action in that state; each edge represents an action taken, here R for rock, P for paper, and S for scissors. Leaf numbers are payoffs from the perspective of player 1. The dashed rectangle around multiple nodes represents an information set of given nodes.

EFG has *perfect recall* if $i \quad N, \ I \quad I_i : h, h \quad I \quad X_i = X_i(h)$, where $X_i(h)$ is a sequence of player i's pairs of information set and action that were taken to reach history $h \quad H$ in the same order as in h. In other words, it means that player i remembers all discovered information during its play up to h. In the rest of the thesis, all games will be two players zero-sum imperfect information in extensive form, and players will have perfect recall.

Pure strategy s_i in a EFG for player i is assignment of some action a for each information set where player i acts. S_i is the set of all pure strategies of the player i, formally $S_i = \prod_{I \ I_i} A(I)$. The probability distribution over pure strategies is *mixed strategy*. Space of all possible strategy profiles is denoted $\Sigma = \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_n$.

A is a function $\sigma_i : I_i \to \delta(A_i)$ which for each information set assigns a probability distribution over available actions. A tuple of all players' behavior strategies is called *strategy profile*, formally $\sigma = (\sigma_i)_{i \ N}$. The probability of history h occurring if players behave according to strategy profile σ is $\eta^\sigma(h)$. Probability of reaching a particular information set I given σ is defined similarly, $\eta^\sigma(I) = \sum_{h \ I} \eta^\sigma(h)$.

Best response is a strategy profile σ_i which satisfies $\sigma_i \quad \sum_i, u_i(\sigma_i, \sigma_{-i}) \geq u_i(\sigma_i, \sigma_{-i})$. It means that strategy profile σ_i of player i results in maximum possible payoff against opponent strategy profile σ_{-i} compare to all other

available profiles. Set of all best responses for player i to strategy profile σ_{-i} is $BR_i(\sigma_{-i})$. The strategy profile such that all players' strategies are best responses to other players' strategies is *Nash equilibrium*, $i \quad N, \sigma_i \quad RB_i(\sigma_{-i})$. The common sense behind equilibrium is that no player can gain any advantage by playing a different strategy than equilibrium. In general, games may have more than one Nash equilibrium. In zero-sum games, every Nash equilibrium expected outcome is the same.

To evaluate Nash equilibrium profile σ employed by players in zero-sum game, we define game value as expected payoff to the first player, $v = u_1(\sigma)$. If player 1 makes some adjustment to their strategy and gets a new strategy σ_1, then $\sigma_2 \quad \sum_2, u_1(\sigma_1, \sigma_2) = min_{\sigma_2} u_1(\sigma_1, \sigma_2) = v - \epsilon_1$. Intuitively, player 1 becomes exploitable by amount $\epsilon_1 \geq 0$, because player 2 can change its strategy to exploit the error player 1 makes and achieve $u_2(\sigma_1, \sigma_2) = -v + \epsilon_1$. Similarly, player 2 can become exploitable by ϵ_2. *Exploitability* value $\epsilon_\sigma = \epsilon_1 + \epsilon_2$ is used to get the distance of a σ to an equilibrium. If and only if a strategy profile has 0 exploitability is an exact Nash equilibrium.

2.2 Counterfactual regret minimization

In most iterative algorithms used for game solving, there is no guarantee to reach a Nash equilibrium in a finite number of steps.

An *ϵ-Nash equilibrium* is an approximate solution to finding Nash equilibrium, where the expected value for each player i is within ϵ of the value of best response, $u_i(\sigma) \geq u_i(\sigma_i, \sigma_{-i}) - \epsilon$. An ϵ-Nash has at most ϵ exploitability, strategy profile with exactly ϵ exploitability is a 2ϵ-Nash equilibrium.

Regret is a model used to quantify how an action taken during T trials is preferred versus a set of other actions that could have been taken instead. The average overall regret at time T is:

$$R_i^T = \frac{1}{T} \max_{\sigma_i \quad \sum_i} \sum_{t=1}^{T} (u_i(\sigma_i, \sigma_{-i}^t) - u_i(\sigma^t))$$

The average strategy for player i until step T is $\overline{\sigma}_i^t$, defined for every information set $I \quad I_i, \quad a \quad A(I)$ as:

$$\overline{\sigma}_i^t(I)(a) = \frac{\sum_{t=1}^{T} \eta_i^{\sigma^t}(I)\sigma^t(I)(a)}{\sum_{t=1}^{T} \eta_i^{\sigma^t}(I)}$$

An algorithm is regret minimizing, if player i's average overall regret goes to zero as $t \to \infty$. That means that such an algorithm can be used to approximate Nash equilibrium.

As first shown in [28], it is possible to decompose overall regret into a set of additive regrets, that can be minimized independently. The *counterfactual regret* of not playing $a \quad A(I)$ at I is the difference in the expected utility of playing a at I instead of $\sigma(I)$, weighted by the opponent's probability of reaching I. Counterfactual utility $u_i(\sigma, I)$ is defined as expected utility given that information set I is reached, and all players are playing according to

strategy profile σ except that player i plays to reach I. Then counterfactual utility is:

$$u_i(\sigma, I) = \frac{\sum_{h \in I, h \sqsubset z} \eta^{\sigma}_{-i}(h)\eta^{\sigma}(h, h')u_i(h')}{\eta^{\sigma}_{-i}(I)}$$

Additionally, let $\sigma|_{I \to a}$ be a strategy profile that differs from σ only in player i choosing action a in information set I. Then the *immediate counterfactual regret* at step T is:

$$R^T_{i,imm}(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \eta^{\sigma^t}_{-i}(I)(u_i(\sigma|_{I \to a}, I) - u_i(\sigma^t, I))$$

It is possible to use counterfactual regret minimization to minimize regret, because regret is upper-bounded by the sum of immediate counterfactual regret:

$$R^T_i \leq \sum_{I \in \mathcal{I}_i} R^{T,+}_{i,imm}(I)$$

where we use $R^{T,+}_{i,imm}(I) = max(R^T_{i,imm}(I), 0)$ as we are only interested in positive portion of the immediate counterfactual regret. Proof can be found in [28].

In this thesis, we use an improved algorithm to compute counterfactual regret minimization, CFR+ [26]. It uses an improved regret-matching technique to speed-up convergence of the algorithm. Let *cumulative counterfactual regret+* at information set I for taking action a at iteration T define as:

$$R^{+,T}_i(I, a) = \begin{cases} max\{v_i(\sigma|^T_{I \to a}, I) - v_i(\sigma^T, I), 0\} & T = 1 \\ max\{R^{+,T-1}_i(I, a) + v_i(\sigma|^T_{I \to a}, I) - v_i(\sigma^T, I), 0\} & T \geq 1 \end{cases}$$

Strategy profile is produced by regret-matching+:

$$\sigma^{T+1} = \begin{cases} \frac{R^{+,T-1}_i(I,a)}{\sum_{a \in A(I)} R^{+,T-1}_i(I,a')} & \text{if the denominator is positive} \\ \frac{1}{|I|} & \text{otherwise} \end{cases}$$

To see the implementation of CFR+, go to [26].

2.3 Decomposition

A common part of most algorithms used for solving perfect information games is decomposition. As all information sets in perfect information game contain only one state, it is easy to split the whole game into parts and solve those independently. In recent years, such algorithms reached a super-human level play in games of Go [25] or chess [24].

Using similar techniques for imperfect information games was until recently impossible. The problem is to decompose such a game into subgames as a strategy is defined on information sets, not separate states. Progress was made

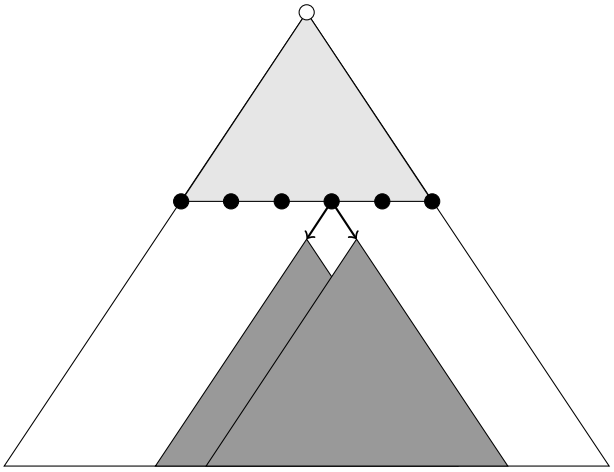

Figure 2.2: Game tree with subgames.

in [8], showing that imperfect information games can be safely decomposed into a trunk and subgames, which can then be re-solved when necessary.

To better understand the logic behind both trunk and subgame definition, we first define them for perfect information games and later extend the definitions to imperfect information games. A subgame in perfect information game is defined as a state and all of its descendants. In other words, the root of the subgame is some state of the original game, and the subgame is a set of states closed under the relationship "is descendent of". The trunk can then be defined for a complete game and a set of subgames as all states that are not in subgame. From this definition, it is evident that the trunk and subgames divide the whole set of states.

In Figure 2.2 is a visualization of the trunk and subgames in a perfect information game. The white node is the root of the game; the light grey triangle represents the trunk, the dark nodes are the states in the trunk border, and the dark shaded triangles are subgames (not all of the subgames are shown). To solve the trunk, we only need to summarize each subgame's strategy with our value against opponents' best response and keep this value in the corresponding state in the trunk border. Later, when choosing an action in that state, we need to compare the subgame values and pick the one that maximizes expected utility. Subgame can be resolved anytime necessary by the same algorithm as it was solved in the first place.

For imperfect information games, the definitions slightly differ. First, we define an *imperfect information subgame*: the root is a set of states, and the subgame contains a set of states which is closed under both the "is a descendent of" and the "is in the same information set" relationships. Meaning, if state s is in subgame S and belongs to information set I, $s \quad S \quad s \quad I$, then all descendants of s and all states in I are in the subgame. This condition is crucial as we have to guarantee that the subgame does not split up any information sets. The definition of *trunk* stays the same - all states that are not in a subgame. To denote trunk of a game G, we use G^T, where T is subset of $H \setminus Z$. Information partition of the trunk is I^T.

To compute the value of a subgame in a perfect information game, we only need to know the root state of the subgame. In comparison, imperfect information subgame has a probability distribution across its root states dependent on the trunk's strategy, meaning that the value of subgame changes as trunk strategy does too. To get a trunk strategy, we use an algorithm *CFR-D*, an extension of CFR. At each iteration, we have a complete trunk strategy and, therefore, well-defined subgames as well. We compute the counterfactual values of the information sets at the root of the subgames using the trunk strategy. With that, we can then update the regrets in the trunk and generate a new strategy profile to be used in the next iteration.

A trunk strategy is a special case of partially defined strategies. Those are part of a Nash equilibrium if we can extend them to a full Nash equilibrium of the game. Formally, σ^J, a profile of partial strategies on I I, is in Nash equilibrium only if exists a full Nash equilibrium strategy profile σ $\Sigma : \sigma^J(I) = \sigma(I)$, for I I.

The information available to player i in histories where he does not act is captured by *augmented information set* I^{aug}. Two histories g, h $H \setminus Z$ belongs to the same augmented information set of player i, if his observation history of g and h are the same, $\vec{\vec{O}}_i(g) = \vec{\vec{O}}_i(h)$. *Observation history* of player i in history h $H \setminus Z$ is sequence of the information sets visited and action taken by i on the path to h. We use to denote two histories indistinguishable by at least one player: g $h \Leftrightarrow$ i $N : \vec{\vec{O}}_i(g) = \vec{\vec{O}}_i(h)$.

We use I^{aug} as a way of generalizing information sets, so we only use the common knowledge available to all players. This approach will provide us with the correct definition for subgames in an imperfect information setting. The *common knowledge public tree* is partition S of H that forms a tree with respect to @ and is closed under . Its nodes S S are *common knowledge public states*, with $S_i = \{I$ $I_i^{aug} | I$ $S\}$ are augmented infosets of player i in public state S. Now a subgame with the root being public state S is the set of histories extending S. This is compatible with our previous definition of subgames but provides a better understanding of what histories need to be included in a subgame. We define a *public observation* O_p as the common knowledge information available to all players after a move was made.

Let us look at an example that summarizes this section. In Figure 2.3, is extensive form of rock, paper, scissors (RPS). The dashed bubble encircling the three nodes of player 2 marks his only information set $I_2 = 2_R, 2_P, 2_S$, with its index indicating action by player 1. There are also three augmented infosets of player 1, the same nodes as in infoset of 2, but he distinguishes between them: $I_1^R = 2_R, I_1^P = 2_P, I_1^S = 2_S$. The game has one apparent decomposition into a trunk and a subgame: let trunk be the decision of player 1; then there is one subgame - the endgame when player 2 selects his choice.

When using some fix trunk strategy, for example, playing rock with probability 0.4, paper also with 0.4, and scissors with 0.2, we cannot naively re-solve the subgame as in perfect information setting, as even when player 2 is playing best response against the trunk strategy, player 1 can exploit it, once he is allowed to adjust his strategy in the trunk [8, 14].

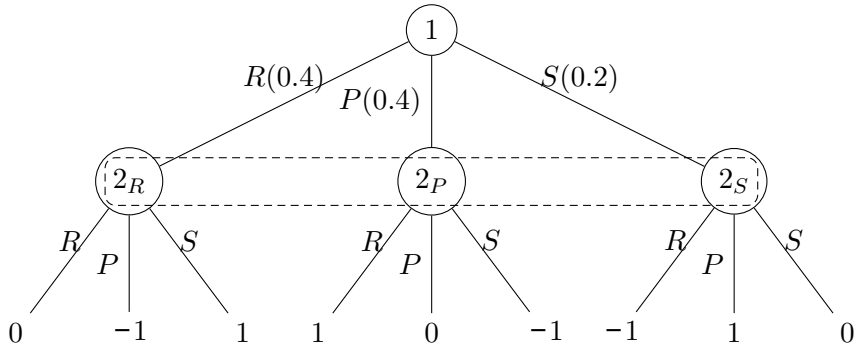

Figure 2.3: A trunk and a subgame of rock, paper, scissors.

However, using the CFR-D allows us to iteratively re-solve subgames and guarantees not to increase the exploitability of the resulting full-game strategy profile.

2.4 Neural networks

If not stated otherwise, this section is based upon [16].

A *neural network* is a function approximator consisting of layered linear functions whose outputs are transformed with a non-linear activation. In this thesis, we use feed-forward neural networks (FNNs) with fully connected layers and convolution neural networks (CNNs). A feed-forward neural network takes an input x and transforms it in predefined sequence with its layers so that the output of layer l is an input of the layer $l + 1$.

A *fully connected layer* is defined as:

$$y = f(W \cdot x + b)$$

where $x \quad \mathsf{R}^N$ is the input and $y = \mathsf{R}^M$ is the output. The matrix $W \quad \mathsf{R}^{M \times N}$ and the vector $b \quad \mathsf{R}^M$ are the parameters, sometimes called weights, of the layer, and the function f is an activation. A feed-forward neural network is solely made of several fully connected layers.

A convolution is an operation on two functions of a real-valued argument. In this thesis, we use two dimension convolution. We define it as:

$$S(i,j) = (K \quad I)(i,j) = \sum_m^M \sum_n^N I(i+m, j+n) K(m,n)$$

where $I \quad \mathsf{R}^{K \times L}$ is an input matrix, $K \quad \mathsf{R}^{M \times N}$ is called kernel and S is the output, sometimes referred to as the feature map. The dimension of the output is determined by several other parameters (stride, padding), which are in detail described in [16].

There are many activation functions used in neural networks. Commonly, a rectified linear unit (ReLU) [15] is used:

$$f(x) = \max\{0, x\}$$

In this thesis, for the output layer we use linear activation, which is an identity function, $f(x) = x$, as we use the neural network to predict a real-number value.

Neural networks are trained by minimizing a loss function. As shown in [19], for similar problem as ours, the L1 loss is often used. It is defined as:

$$L_1(y, f(x) = \sum_{i=1}^{n} |y_i - f(x)_i|$$

In [21], Huber loss was presented as showing better result when minimizing it during train. It is less sensitive to outliers. It is defined piecewise:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

Weights of the neural network are optimized with a gradient descent. We used an Adam optimizer [17], first-order gradient-based optimization algorithm.

Chapter 3

Related Work

Most of the previous research into large imperfect information games was focus on the domain of poker. First, with Heads-up Limit Texas Hold'em [27], later Heads-up No-limit Texas Hold'em [19, 7] and last year even multiplayer Texas Hold'em [5].

In this chapter, we will discuss related work in both solving imperfect information chess and generally of solving large imperfect information games. First, we present previous work done in the Kriegspiel domain. Next, we talk about the *Deepstack* algorithm, and finally, we discuss the newest approaches for depth-limited solving and value functions.

3.1 Solving Kriegspiel

From its invention in 1899, Kriegspiel quickly became popular amongst chess players for its complexity and surprises during play. Nevertheless, due to the game's enormous size, most of the research focused on solving particular endgames, for example, king - rook - king (KRK). In [3], Boyce analyzed properties of the KRK endgame and compared it to traditional KRK endgame in perfect information chess. Boyce showed a deterministic algorithm to force a win in this endgame in a bounded number of moves. It seeks to trap the black king in a single quadrant of the board by maneuvering white's king.

The first endgame implemented on a computer was king and pawn vs. king (KPK) [10]. Another finding of this paper was an example of Kriegspiel's position in which stronger side cannot win with probability 1, but can get as close to 1 as it desires. It showed that a position exists, which is equivalent to a recursive game called Blotto's problem. To win, the stronger player needs to take a small risk and is unable to achieve a full reward of 1.

Another improvement presented concept of *metaposition*, first discussed in solving chess-like game of Shogi [20] and later used for Kriegspiel [2, 11]. Metaposition allows the merging of a large set of possible game states into a single state and then applying perfect information solving algorithms. This method led to outperforming any other known computer agent.

In [13], an approach using a Monte Carlo Tree Search (MCTS) outperformed all opponents with a perfect score in the Kriegspiel tournament at the 14th Computer Olympiads.

Last published improvement in agent design is [12]. It combines MCTS with priority planning. They define two types of goals based on chess theory: priority goals (short term advantages) and long term goals (placing pieces in favorable positions, regrouping pieces). When an agent needs to pick an action, it tries to pick such action that leads toward reaching predefined goals; if no such action is available, it runs MCTS to pick the best possible one. With this method, they were able to beat the previous strongest player. This paper currently represents the last research done on the domain of Kriegspiel.

Previous research focused on producing the strongest possible agent and achieving a super-human level of play. In comparison, our approach goal is to approximate a theoretical perfect play - compute an ϵ-Nash equilibrium.

3.2 DeepStack

In 2017, *DeepStack* [19], an online playing general-purpose algorithm for sequential imperfect information games, was published.

It uses a concept of continual re-solving - reconstructing own strategy every time it needs to act and thus never needing a complete strategy for the whole game. This method is a theoretically sound approach but impractical as resolving is time-expensive except near endgame. DeepStack solves the challenge by decomposing the game and only solving the trunk strategy and approximate the counterfactual values of subgames with learned value function. It reduces the size of the game needed to be re-solved at the beginning of the game from 10^{170} decisions down to less than 10^{17}. With reducing the number of actions considered to six or seven (fold, call, 2-3 bets, and an all-in), the look-ahead tree needed to be re-solved was reduced to approximately 10^7 decision points and allowed solving the game with single GPU in less than 5s.

This reduction could be potentially exploited by opponents, as they could try to employ unfamiliar betting sizes. However, as shown in figure 3.1, with an increasing number of games played, DeepStack was able to defeat professional poker players consistently. The performance is measured in terms of mbb/g, which stands for milli-big blinds per game and represents how many big blinds (initial money player must put into the pot) a player wins on average over 1000 games. It is a standard win rate measure used in the literature. The red line represents a break-even level of 0mbb/g; the blue dashed line represents the level which professional players consider sizable margin, 50mbb/g.

DeepStack uses two separate neural networks: one for estimating counterfactual values after the flop is dealt (first three public cards) and second after the fourth card is dealt (turn). The input of the networks is the ranges of both players (distribution of possible private card holding), pot size, and public cards. This amount of information is sufficient to specify a public state of the game.

The networks need to be trained before play. Training data are generated by simulating random poker situations and then solving them. The turn

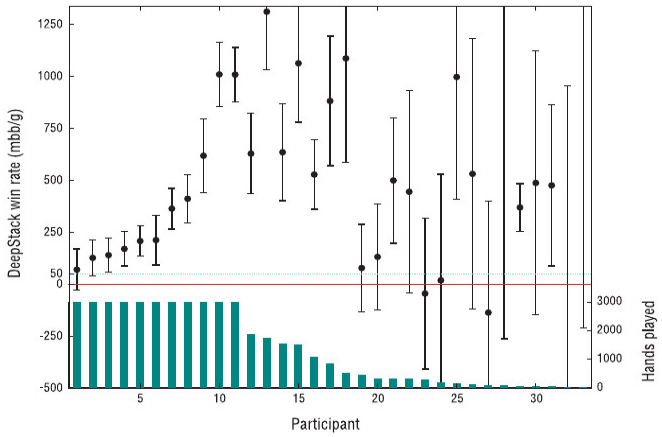

Figure 3.1: Performance of DeepStack against professional poker players[19].

network was trained first, and then it was used to compute training data for the flop network. This stacking of the neural networks speeds up the process of data generating.

Although DeepStack is presented as a general algorithm, its development was heavily influenced by poker, some of its design was tailor-made to poker properties and never was presented to play other imperfect information games.

3.3 Multi-Valued states

In [6], Brown et al. described a new method for depth-limited solving in imperfect information games referred to as *multi-valued states*. The main idea is associating terminal histories h of the trunk with values $v^{\sigma_1,\sigma_2}(h)$ opponent achieved playing different strategies against a fixed approximation of the equilibrium for the whole game.

Their main proposition is, that in order to calculate player 1 Nash equilibrium strategy in a trunk S, reached when player 1 played according to Nash equilibrium strategy σ_1 prior to reaching it, it is sufficient to know player 2 value playing best response against σ_1, $v_2^{\sigma_1,BR(\sigma_1)}(I)$ for its every root information set $I \quad S$, and player 1 value for every pure undominated strategy σ_2 and every leaf node $h \quad S$, $v_1^{\sigma_1,\sigma_2}(h)$.

In practice, there are numerous pure strategies σ_2 against σ_1 to know expected values in every state. Moreover, σ_1 is unknown. They propose a workaround to estimate these values and still achieve excellent performance.

The first step is to precompute an approximate equilibrium fro the whole game, which they call blueprint strategy $\hat{\sigma}$. It is computed first by abstracting the game by bucketing together similar situations and the running Monte Carlo Counterfactual Regret Minimization algorithm. This strategy is only used to estimate value at the trunk leaf $v^{\hat{\sigma}_1,\sigma_2}(h)$ and agent never uses it for playing decision.

Next is the selection of player 2 strategies σ_2. Brown et al. claim that it is

possible to use a small number of diverse, intelligent strategies, e.g., 10, as a choice of strategy is made at each leaf node independently. This leads to choosing between a much higher number of strategies (for a trunk with 100 leaves, it would be 10^{100} different strategies). The strategies can be generated in two different ways: *bias approach* or *self-generative approach*.

Bias approach utilizes the blueprint strategy and bias to it in different ways. For example, a new strategy σ_2 in poker could be obtained by multiplying the probability of choosing fold action by a given bias and then normalized all the probabilities.

The self-generative approach constructs the set of σ_2 via self-play iteratively. At the start, the only strategy is the blueprint $\hat{\sigma_2}$. Then the trunk of the whole game is solved to whatever depth is feasible; player 2 can only choose a strategy from the strategy set. Let the solution to this game be σ_1. The best response strategy to σ_1 is computed and added to the set of strategies of player 2. This process is repeated until there is a desired number of different σ_2 strategies in the set.

With this approach, they were able to defeat previous top heads-up No-Limit Texas Hold'em poker agents Baby Tartanian8 and Slumbot using only a 4-core CPU and 16 GB of RAM for real-time play.

Brown et al. thoroughly discuss the advantages and disadvantages of their method against DeepStack. They highlight that their approach is less computationally expensive and easier to scale up. On the other hand, DeepStack does not need an approximate Nash equilibrium in blueprint strategy, and it computes best-response against all possible opponent's strategies, not just selected few.

There has not been a direct comparison of head-to-head performance made between these two solving methods in any domain published so far.

3.4 Value functions beyond poker

The two previously mentioned methods, DeepStack and Multi-valued states, were built around the game of poker, which until recently was the main considered domain in the area of imperfect information games. However, poker has some properties other games do not have. For example, all information sets have the same size, and public observation always obtains the same size of information (number of cards dealt does not change, pot size is always visible).

To help extend research outside of the poker domain, Seitz et al. in [21] proposed a domain and algorithm independent definition of a value function for general extensive-form games and its representations. They proved that any optimal value function enables depth-limited solving. They experimentally showed on three fundamentally different domains (Generic Poker, Imperfect Information Goofspiel, Oshi-Zumo) that neural networks could be easily trained to approximate counterfactual value functions.

The important finding of this paper was Theorem 7 - Public state minimality. It shows that public state factorization is the smallest possible for solving,

and anything smaller provides insufficient information to evaluate states.

Next, they generalize the concept of range beyond poker. The range at public state S is defined as:

$$rng^{\sigma^T}(S) = \left(\left(\eta_i^{\sigma^T}(I)\right)_{I \quad S(i)} \right)_{i \quad N}$$

where σ^T is some trunk strategy profile, S is a public state in the trunk boarder, $S \quad T \quad Z^T$. It means, player's range at a public state S is set of player i's reach probabilities of information sets in S. They prove that this is a sufficient representation of the context for depth-limited solving. In this thesis, when we talk about range, we mean this definition.

Chapter 4

Problem Analysis

In this chapter, we introduce the games of Kriegspiel and Darkchess and discuss their differences and their implementation details.

4.1 Imperfect information chess

4.1.1 Kriegspiel

Kriegspiel is an imperfect information variant of chess designed by Henry Michael Temple in 1899. During the play, each player only sees his own pieces and only gets sparse observation of the moves the opponent made. For this reason, an umpire with complete information on the game progress is necessary.

Rules

The basic rule is that each player knows the exact positions of their own pieces only. Apart from that, there are several rules variants for Kriegspiel, mostly differing in what observation players are getting. In our implementation, each player received this observation after each move:

- White / black to move.
- Pawn gone, after a pawn was captured.
- Piece gone, after a piece was captured.
- Illegal move, after a player tried to perform a move, that is impossible to do in this game state, e.g., pinned figure tries to move away from pinned square or pawn advancing to an occupied square.
- Check, followed by all variation in effect: vertical, horizontal, long diagonal, short diagonal, by a knight.
- Checkmate, stalemate, draw.

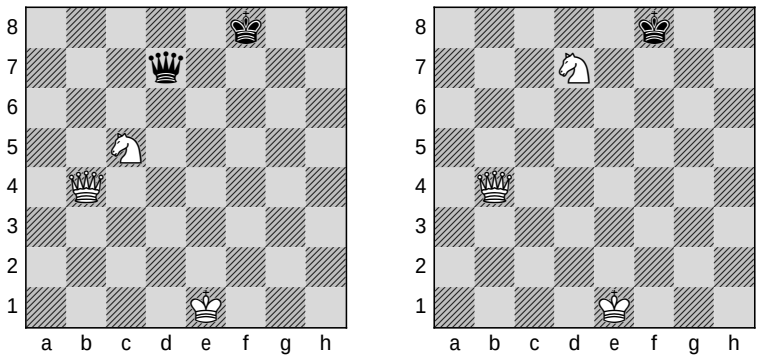

Figure 4.1: Multiple observations in Kriegspiel.

There can be multiple observations given to a player after a move, as is shown in Figure 4.1. After white captures queen with Nxd7+, both players receive a public observation from the umpire: "Piece gone, check long diagonal, check by a knight, black to move." Private observation for black adds the information that his queen was taken; white private observation is the same as the public.

Observation implementation

As the rules of the game strictly provide the observations, we implemented it as a one-hot encoding of thirteen bits. The first five bits represent checks, the next two encode if there was capture and if yes, it encodes if that was a piece or a pawn and the last six encode position on the board of the capture. The position is independent of the size of the board.

0	0	1	0	1	0	1	1	1	0	1	0	0

Table 4.1: Observation encoding

Example of encoded observation for move from Figure 4.1 is shown in Table 4.1.

As the rules of Kriegspiel allows us to make an illegal move, we implemented it only as private information for the player on the move as it gives nothing concrete to the non-acting player.

Difficulties with illegal moves

Making an illegal move gives extra information to the player on the move. For example, a pawn can only take diagonally, and it could be helpful to try all possible pawn captures first, as it may reveal empty squares on the board. This led to the invention of the "Are there any?" rule, when umpire provides all possible pawn, captures to a player at the beginning of its move.

Still, illegal moves make the game extremely complicated as the game tree is imbalanced in the depth of each player's nodes, the same player can

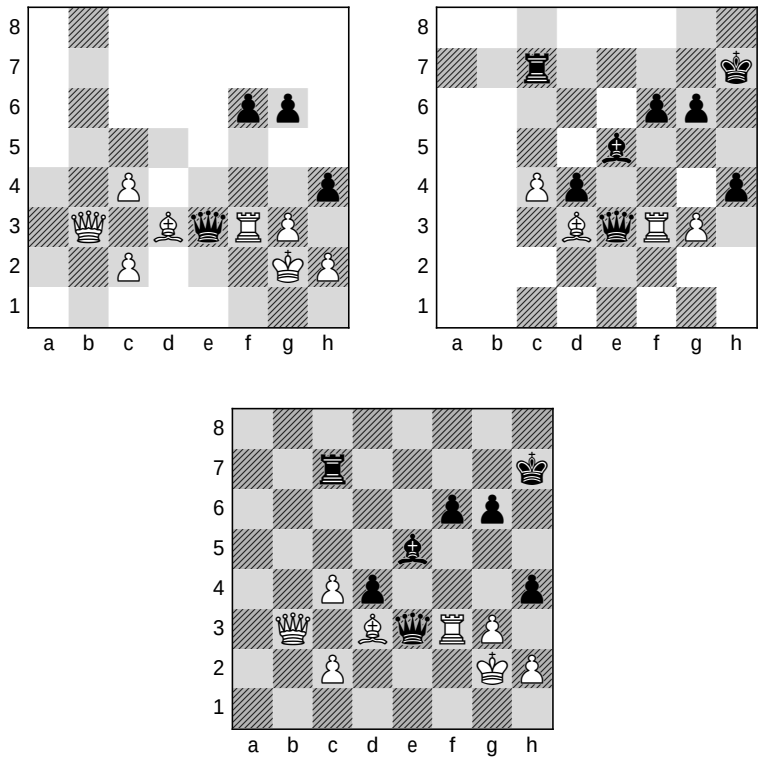

Figure 4.2: Position in Darkchess from white's and black's perspective.

play multiple times in a row, and the same board position can be reached in multiple different ways.

4.1.2 Darkchess

Darkchess, sometimes referred to as fog-of-war chess, was invented in 1989 as a version of correspondence chess by Torben Osted and Jens Nielsen. It was inspired by a version of Kriegspiel, where at the beginning of a move player can ask the umpire, "Are there any?" meaning if it is possible to make any capture with a pawn. During the play, players see all of their pieces but also sees all the squares they can legally move and opponents pieces on those squares.

Rules

The goal of the game is to capture the opponent's king. This change leads to removing the concept of a check from the game, and it is legal to move into check or fail to move out of check, a player also does not receive information that his king is under attack. During the play, players see all of their pieces but also sees all the squares they can legally move and opponents pieces on those squares. It is explicitly indicated to each player which squares are hidden, which, compared to Kriegspiel, leads to no illegal moves.

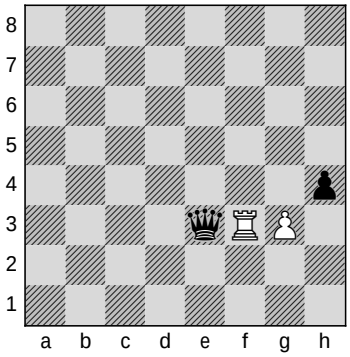

Figure 4.3: Example of public observation in Darkchess.

The difference in information available to each player is demonstrated in Figure 4.2. The whole game is shown on the bottom chessboard. The top left shows the board from the white perspective, and the top right chessboard is from the perspective of black. The plain white squares indicate squares that are out of player's reach.

Observations

In Kriegspiel, private observation was almost the same as a public observation. On the other hand, players in Darkchess rely on private information to determine available moves. However, what is a public observation in Darkchess?

We decide to implement it as a chessboard with only the pieces that can capture each other next move. So for the position in Figure 4.2, the public observation would look like Figure 4.3. There could be multiple levels of "I know this, so you have to know that" situations, which would mean that there is more information in the public observation, but we show that this together with private observation is enough.

4.1.3 Game complexity

There are two critical measures of game complexity: state-space size (number of distinct legal states allowed by the game's rules) and game-tree size (number of distinct games that can be played). For chess, several estimates were given, all around 10^{46} for state-space size and 10^{120} for game-tree size [1, 23].

For Darkchess, the game complexity will not be much different: only legal moves are allowed, so nothing can blow-up the game tree or introduce new game states.

It is a different story for Kriegspiel. The number of possible games (tree-size) is enormous when illegal moves are taken into account: between any two moves, there can be any sequence of illegal moves. If the average branching factor is 40 and from that 10 are for illegal moves, there is an average of 1000

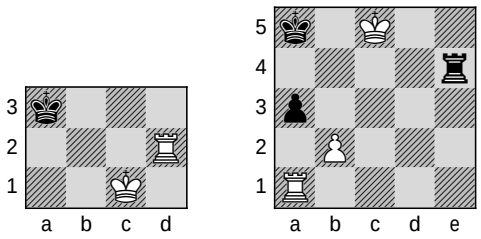

Figure 4.4: Used chessboard configurations.

combinations of illegal moves in between any two moves of any chess games [13].

What sets Kriegspiel apart from other imperfect information games is the possible size of information sets. As stated previously, perfect information games, like chess or Go, have only one state per information set. The game of Heads-up No-Limit Texas Holdem has information sets up to the size of 10^3. Krigspiel's are much bigger: up to 10^{14} states per information set in the later part of the game.

This thesis's goal is to show that this method of depth-limited solving works for the imperfect information chess domains and not complete game solving. We use two smaller board configuration and endgame setting to run our experiments.

4.1.4 Board configurations

Two chessboard configurations used are shown in Figure 4.4: Minimal4x3 on the left and minimized endgame for 5x5 board on the right (this configuration is referred to as Mate2). Minimal4x3 is good for testing the correctness of used algorithms, as in Kriegspiel, there is a pure strategy for white (1. Kb1 Kb3 2. Rd3#), and slightly different for Darkchess (1. Rb2 Ka2).

Still, both configurations provide a big game tree for both domains (Minimal4x3 has 40k nodes, Mate2 has 3.7m nodes).

4.2 Note on AlphaZero

DeepMind recently showcased the strongest chess agent in AlphaZero [24]. It outperformed the best chess engine at the time, Stockfish, winning 155 games and losing 6 out of 1000. It uses a general-purpose Monte Carlo tree search, selecting moves with high move probability and value according to a neural network. It is trained during a self-play period.

Their approach differs from ours in core principles: we are using a game theory approach which, in its core, can guarantee certain properties (Nash equilibrium, exact exploitability). On the other hand, AlphaZero uses reinforcement learning with the goal of achieving a superhuman level of play. As

this approach leads to great results against humans or other computer agents, it does not guarantee that it leads to theoretical perfect play.

4.3 Implementation

Code of this thesis was implemented as part of the GTLib2, Game Theoretic library written in C++, developed by AI Center in the Department of Computer Science at the Faculty of Electrical engineering of CTU in Prague As part of this thesis, implementations of Kriegspiel and Darkchess domains were added to the library.

A game within the library is modeled as a graph whose nodes represent the true state of the game. The edges or transitions from state to state happen only when the actual game state has changed. Each edge has an outcome assigned to it: the new state, vector of private observations, public observation, and rewards for each player. Extensive form game in as a proposed Factored-Observation Game (FOG) [18] is built on top of this graph.

Apart from implementing both domains, we also modified the implementation of CFR-D to work with multiple neural networks at different depths of the game tree.

Chapter 5

Experiments

In this chapter, we present the results of our experiments. First, we explore the usability of value functions for offline depth-limited solving of imperfect information chess. Next, instead of using FNNs that were previously used, we experiment with using a convolutional neural network as they proved effective in board game solving. Lastly, we take a step towards online play with the stacking of neural networks to allow solving large games.

For every experiment, the maximum depth of a game tree is set to 6, meaning that the game ends after each player makes 3 moves. For Kriegspiel, illegal moves are not counted.

5.1 Depth-limited solving

In the first part, the aim is to test the usability of our game representation, design, and training of counterfactual value networks for Kriegspiel and Darkchess. The public information available in both games is very sparse, sometimes players receive empty public observation for several moves; for example, in Kriegspiel, it is more likely to make a move without capturing opponents piece or checking his king, the only two actions in public observations. In previous work [21, 19], the domains always produced a non-empty public observation and so it was easier to train networks.

The goal is to achieve strategy with low exploitability produced using depth-limited solving. We divide the game into a trunk and subgames; trunk depth is the depth of the game tree containing trunk leaves $h \quad Z^T$. The counterfactual values of a public state at the root of each subgame is approximated with a neural network.

For Kriegspiel, we test the Minimal4x3 board configuration with the trunk border at depth 2, so after each player made one move. This depth already provides big game: there are 96 information sets and 17 augmented information sets, with 3 public states at the trunk border.

For Darkchess, we used both Minimal4x3 and Mate2.

The networks were fully-connected feed-forward neural networks (FNNs) with 2 hidden layers for the Minimal4x3 board and 5 for Mate2. Inputs of the networks are floats representing the public observation made prior to reaching the public state and corresponding ranges of both players, and the

Domain	Trunk depth	Input		Output
		Public Features	Range	
Kriegspiel Minimal 4x3	2	26	226	226
Darkchess Minimal 4x3	2	24	12	12
Darkchess Minimal 4x3	4	48	97	97
Darkchess Mate2	2	50	27	27

Table 5.1: Networks' input and output sizes for different domains.

networks output counterfactual values. Sizes of input and output for each used game configuration is shown in Table 5.1.

5.1.1 Training data

We generate training data for the network by splitting the games into trunk (each player makes 2 or 4 moves, depending on the trunk depth) and bottom (successive moves). We assign a randomly created strategy to all information sets in the trunk and fix it, so it will not change while solving the game. We use 500 iterations of CFR+ to solve the subgames in the bottom. We store the ranges (fixed probabilities of private observation) of each player and their corresponding near-optimal values. This way, we get as many data samples as there are public states at the trunk depth. Repeating the process with different random trunk strategy produces new training data.

5.1.2 Training

For training for Minimal 4x3 chessboard, we used 1000 game situations, and for Mate2 15 thousand, 90% of the data for training and 10% for validation. We minimized a Huber loss with the Adam [17] gradient descent optimization procedure. Even though Huber loss preforms good for training counterfactual networks [19, 21], it is not perfect for such networks. As we use its output in CFR+ iteration, it is crucial that it has a correct sign (CFR+ ignores negative counterfactual values).

5.1.3 Evaluation

To evaluate the quality of the networks, we used an algorithm similar to CFR-D. When a state in the trunk border is reached (leaf of the trunk), we take the current ranges and public observation and approximate the counterfactual values of that state with trained neural net and substitute them. We run 500 iterations of this algorithm and then computed the exploitability of the trunk strategy.

For each game, we trained and evaluated 20 networks. Exploitability of the best performing networks is shown in Table 5.2. It proved to be challenging to predict the quality of a given network by its valuation losses, as some

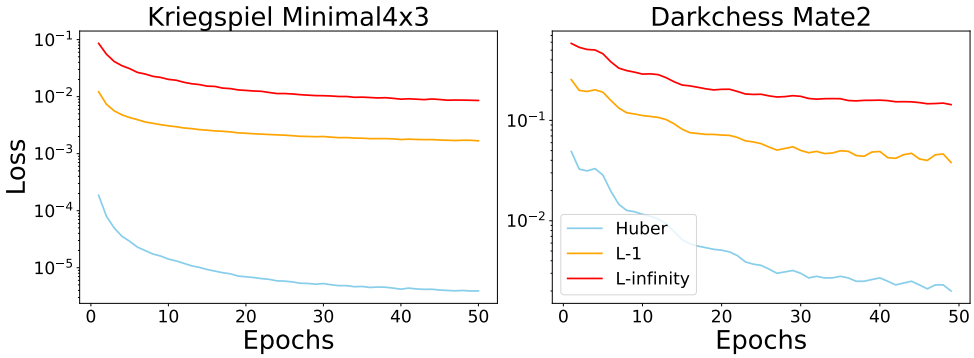

Figure 5.1: Validation losses on Kriegspiel Minimal4x3 and Darkchess Mate2.

network with almost identical losses as network producing strategy with $\epsilon \leq 0.1$, produced strategy with exploitability over 0.4.

Domain	Trunk Depth	Public States	Exploitability
Kriegspiel Minimal 4x3	2	3	0.0651
Darkchess Minimal 4x3	2	13	0.0009
Darkchess Minimal 4x3	4	407	0.0084
Darkchess Mate2	2	77	0.0960

Table 5.2: Best results for FNNs in different games.

This result shows that our game representation is suitable for use with value functions. Final exploitability indicates a close approximation of Nash equilibrium; with more iterations, it would be possible to be arbitrarily closer. It is important to note again that both Kriegspiel and Darkchess have significantly different properties than poker, for which this technique was developed. To mention few: information sets have arbitrary size, one player can play several moves in a row, and the game can terminate in different depths. Still, we showed that even for this complicated domain, value function could be successfully used.

5.1.4 Reach-weighted loss function

During training, we noticed that apart from networks with small training and validation loss and produced strategy with low exploitability. There were also networks with similarly small losses but which produced strategy with three to fourth times higher exploitability. One explanation for this behavior is that the network performs well in information sets that are rarely visited and poorly in highly visited information sets, leading to low loss and high exploitability.

We came up with an idea to use reach probability of information set to weight significance of errors during training, as the ones with low reach probability are not as interested for us as the ones with high probability. We

defined a *reach-weighted Huber loss* as:

$$L_{\delta}^{RW}(y, f(h)) = \eta^{\sigma^T}(h) \cdot L_{\delta}(y, f(h))$$

for $h \quad Z^T$. This loss is still differentiable, and thus we can use it to train a neural network.

In practice, reach-weighted Huber loss did not bring any notable improvement into the training. Results showed that it only linearly transformed the overall error closer to zero but did not improve the exploitability of the strategy obtained.

5.2 Convolution networks for Darkchess

As presented in [24], a convolutional network can be successfully used as a value function for perfect information game solving, when there is a game board with pieces. It allows for a better abstraction of the public features than domain-specific one-hot encoding. Darkchess public observation satisfies this property, so we decide to test, whether it will provide better results than FNNs.

Input to CNNs is a spacial plain of values, usually a matrix of pixels. We used a stack of plains, each representing one chessboard, which encoded pieces on their actual location. As we beside public observation use also ranges of both players, we had to split the input into two parts: the public observation and ranges.

The network architecture is two convolutional layers, each with 5 filters of kernel size 3x3 and stride 1. After each was ReLU activation, followed by one fully-connected layer as the output layer. The convolutional layers take only the public observations as an input, and to their output have concatenated the ranges, which is the input into the last fully connected layer.

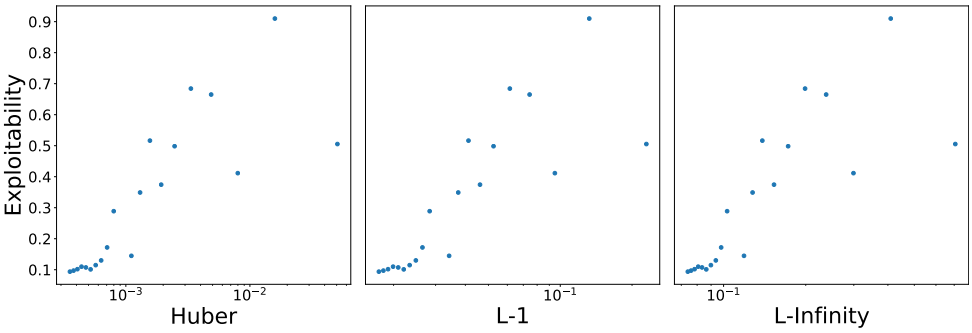

Figure 5.2: Exploitability versus validation error for convolutional network in Darkchess Mate2.

Even with this small network architecture, we were able to achieve a better result than with bigger FNNs. In Figure 5.2, we show the results of networks on Darkchess Mate2. We again minimized a Huber loss, but it is apparent we would achieve a similar result if we decided on L_1 or L_{inf} loss function.

This is a new contribution to solving imperfect information chess; previously, only FNNs were used.

5.3 Step towards online playing

Previous two experiments showed that it is possible to use a neural network to compute a low exploitable strategy with depth limited solving for small game or endgame. But to be able to play a whole game of Kriegspiel or Darkchess, which could be up to 50 moves, we need to provide a framework for online playing, as keeping strategy for the whole game is impractical (it would require over roughly 500TB of space).

One approach to online play is continual re-solving, as was used in [19]. The idea is to re-solve players' strategies during the game with limited depth. This requires to have multiple counterfactual value networks, that predicts values at different depths, for example, for poker, DeepStack used one network after the flop and one after the turn.

5.3.1 Darkchess

We examine this approach for Darkchess Minimal4x3. Two value functions are trained - first at the trunk depth 4 and second at the trunk depth 2.

For the network at depth 4, the training data are generated in the way described in section 5.1.1. For training, we used over 80 thousand random game situations as there are more public states at the trunk border, which corresponds to 200 random trunk strategies. Next, we evaluated the networks and picked one with the best exploitability ($\epsilon = 0.0469$). This network is used for generating training data for the second network.

We generated 1000 random game situations for the second network, at depth 2; the process is similar as before, but we used network at depth 4 to substitute counterfactual values, so the subgames to solve are smaller. We evaluated this network after 20 epochs of training, and the exploitability was similar to networks trained without stacking at depth 2, at $\epsilon = 0.0086$.

We briefly tested the effect of the bottom network's quality on the top network's performance. We used a network producing strategy with $\epsilon = 0.128048$, which is still a good result. Using it, we generated training data for the second network, trained it, and evaluated it. With this network at depth 2, the strategy performed poorly, $\epsilon = 0.24765$. This finding suggests that the quality of the bottom network affects the quality of training data for later networks and so even its performance. Even when the game has a terminal state in the trunk that the player wants to reach, if the value function is giving wrong enough approximation, the CFR algorithm can converge a highly exploitable strategy. This property needs to be more thoroughly explored in future work.

Chapter 6

Conclusion

This thesis examined the use of value functions for depth-limited search in imperfect information chess. These games have specific properties that make them difficult to reason about; most notable, the amount of public information given to the players is sparse and quickly loses value.

First, we reviewed related work for depth-limited solving in imperfect information games and different approaches to represent a value function used during the solving. We analyzed the properties of Kriegspiel and Darkchess and discussed their similarities and differences, and implemented both games within the GTLib2. Next, we have introduced a new representation of public states in both games alongside their rules that do not operate with common knowledge. We tested our representation in several game situations and showed its usability by training a neural network and using it as a heuristic for depth-limited CFR. Based on theoretical properties the value function should have, we designed a new loss function, Reach-weighted Huber loss. However, in an experimental test, it did not prove to be performing better.

We experimented with the use of convolutional neural networks as a value function for Darkchess. With fewer parameters than fully connected networks, convolutional network performance was similar to FNNs. The use of CNNs was a new contribution and could be a step for solving bigger games. Lastly, we showcased the use of network stacking, a crucial part of depth-limited solving on whole games.

In future work, the use of our representation for bigger games can be delivered. With the use of network stacking, an agent can be built for the entire game of tens of moves and be tested against human players. Also, the relation between network performance in stacking needs to be studied more.

In conclusion, this thesis provided a theoretical background and representation for depth-limited solving for imperfect information chess. We think of it as a proof-of-concept for building an agent for a whole game.

Bibliography

[1] Louis Victor Allis et al. *Searching for solutions in games and artificial intelligence*. Ponsen & Looijen Wageningen, 1994.

[2] Andrea Bolognesi and Paolo Ciancarini. Computer programming of kriegspiel endings: The case of kr versus k. In *Advances in Computer Games*, pages 325–341. Springer, 2004.

[3] Jim Boyce. A kriegspiel endgame. In *The Mathematical Gardner*, pages 28–36. Springer, 1981.

[4] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365:eaay2400, 07 2019.

[5] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[6] Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-limited solving for imperfect-information games. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7663–7674. Curran Associates, Inc., 2018.

[7] Noam Brown, Tuomas Sandholm, and Strategic Machine. Libratus: The superhuman ai for no-limit poker. In *IJCAI*, pages 5226–5228, 2017.

[8] Neil Burch, Michael Johanson, and Michael Bowling. Solving imperfect information games using decomposition. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[9] Murray Campbell, A.Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134(1):57 – 83, 2002.

[10] P Ciancarini, F DallaLibera, and F Maran. Decision making under uncertainty: a rational approach to kriegspiel. *Advances in Computer Chess*, 8:277–298, 1997.

[11] Paolo Ciancarini and Gian Piero Favini. Representing kriegspiel states with metapositions. In *IJCAI*, pages 2450–2455, 2007.

[12] Paolo Ciancarini and Andrea Gasparro. Priority level planning in kriegspiel. In *International Conference on Entertainment Computing*, pages 333–340. Springer, 2012.

[13] Gian-Piero Favini. *The dark side of the board: advances in chess Kriegspiel*. PhD thesis, alma, 2010.

[14] Sam Ganzfried and Tuomas Sandholm. Endgame solving in large imperfect-information games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[15] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[18] Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. Rethinking formal models of partially observable multiagent decision making, 2019.

[19] Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, Mar 2017.

[20] M Sakuta. Deterministic solving of problems with uncertainty. *Ph. D. Thesis, Shizuoka University*, 2001.

[21] Dominik Seitz, Vojtěch Kovařík, Viliam Lisý, Jan Rudolf, Shuo Sun, and Karel Ha. Value functions for depth-limited solving in imperfect-information games beyond poker, 2019.

[22] Jack Serrino, Max Kleiman-Weiner, David C Parkes, and Josh Tenenbaum. Finding friend and foe in multi-agent games. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 1251–1261. Curran Associates, Inc., 2019.

[23] Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[24] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, and et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[26] Oskari Tammelin. Solving large imperfect information games using cfr+, 2014.

[27] Oskari Tammelin, Neil Burch, Michael Johanson, and Michael Bowling. Solving heads-up limit texas hold'em. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[28] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.

Glossary

ϵ-Nash equilibrium An approximate solution to finding Nash equilibrium, where the expected value for each player i is within ϵ of the value of best response.

behavioral strategy A set of distributions over actions $A(I)$, one for each information set $I \quad I_i$ for player i.

best response A strategy that yields the maximum payoff against a particular set of strategies used by the opponents.

counterfactual regret The regret for *not* taking some action a at information set I and instead playing with some strategy $\sigma(I)$, weighted by the opponent's probability of reaching I, $\eta_{-i}(I)$.

exploitability A value $\epsilon_\sigma = \epsilon_1 + \epsilon_2$ assigned to a profile σ where σ_1 is exploitable by some amount ϵ_2 and σ_2 is exploitable by some amount ϵ_1.

flop First three public cards in poker.

history A particular sequence of actions taken by player; in a perfect recall game every history corresponds to a unique node in the game tree.

information set A set of histories $I \quad I_i$ that a player i cannot distinguish between due to information not known by that player.

mixed strategy A probability distribution over pure strategies.

Nash equilibrium A strategy profile such that all players' strategies are best responses to other players' strategies.

pure strategy A collection of tuples $\{(I, a) : I \quad I_i, a \quad A(I)\}$ for player i such that every I appears exactly once in an extensive-form game.

range Set of player i's reach probabilities of histories at public state S.

regret A quantification, based on difference in utility, of the amount a regret associated with decisions that were made during the course of a number of trials $1, 2, \ldots, T$ versus a set of other decisions that could have been made instead.

regret minimization An iterative process that leads to zero average external regret as the number of iterations T approaches infinity.

subgame Subset of a whole game, where the root is a set of states, and the subgame contains a set of states which is closed under both the "is a descendent of" and the "is in the same information set" relationships.

trunk All states that are not in a subgame.

turn Fourth public card in poker.

Appendix A

User guide

This is a brief user guide for GTLib2, main library used for this thesis. It can be cloned from https://gitlab.fel.cvut.cz/game-theory-aic/GTLib2; it is also attached to the thesis. Neural networks were developed with TensorFlow framework, the code is also attached to the thesis.

A.1 Requirements

GTLib2 uses C++17, python code is developed using python3.7. Other used libraries are listed in Table A.1.

Library	Version
TensorFlowCC	1.13.1
Keras	2.3.1
tensorflow	1.13.1
pandas	1.0.3
numpy	1.18.2
natsort	7.0.1

Table A.1: Libraries used in this thesis.

TensorFlowCC is available at https://github.com/FloopCZ/tensorflow_cc.

A.2 Evaluating networks

In the root folder of the attached code, there is a script for demonstration of the thesis work. It is necessary to run the script from the folder. The script performs several steps:

1. Builds the GTLib2 library.
2. Trains five FNNs for Darkchess Minimal4x3 with a trunk depth 4.
3. Transforms the network from python to C++ compatible format.
4. Evaluates each network by running 500 iterations of CFR-D with the network and then computing strategy's exploitability.