UNIVERSITY OF AMSTERDAM

MSc ARTIFICIAL INTELLIGENCE

MASTER'S THESIS

A Game-theoretic Approach to the Card Game Toepen

BY

DAAN LE
11329866

July 5, 2021

48 EC

NOV 2020 - JUL 2021

Supervisor:
prof. dr. Ulle Endriss

Assessor:
dr. Ronald de Haan

Institute for Logic, Language and Computation

Abstract

This thesis examines the card game Toepen from a game-theoretic perspective. Game theory is a mathematical framework to analyse strategic interactions between rational agents. It offers a strong framework in environments where there is hidden information for players of the game. Toepen is such a game which is also known as an imperfect-information game. Methods of the field of Game theory are first used to analyse the game from a theoretical point of view. This includes modeling Toepen as an extensive game, defining how an algorithm known as Monte Carlo Counterfactual Regret Minimization can be used to generate Nash equilibria in Toepen and how the game can be abstracted to make the size of the game more feasible for this algorithm.

After theoretically going over the game, the methods which are discussed are further investigated by testing them in practice. This is done by using Monte Carlo Counterfactual Regret Minimization to generate strategies that can play small versions of Toepen. Here we find that a larger deck does not necessarily mean that a strategy is harder to compute and that abstracting the hands of cards leads to worse performance than abstracting the history of the game.

Acknowledgements

First of all, I would like to thank my supervisor Ulle Endriss for helping me throughout all the stages of this long project. Every time I felt stuck or unmotivated, our meetings would give me a fresh perspective and help me move forward. You were my first college professor at the Bachelor's AI program, where you taught me a course on problem solving in Prolog using recursion. It is funny how I will now, five years later, end my studies by implementing a recursive algorithm.

Next I want to thank my fishing friends and fellow students, Mathieu, Paul, David and David for our great study sessions and recreational gatherings. I also want to thank my parents and brother for their support and love during my studies. Finally, I want to thank my partner Andrea for always being there and supporting me throughout this year.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The advancements in Artificial Intelligence (AI) are often measured in terms of the performance of artificial agents in popular games. Such advancements include beating top humans in chess [Campbell et al., 2002] and Go [Silver et al., 2016]. More recently, AI agents were able to beat top humans in heads-up no-limit Texas hold 'em Poker [Brown and Sandholm, 2018] and shortly thereafter multiplayer no-limit Texas hold 'em Poker [Brown and Sandholm, 2019]. Although these are all important milestones in the field, the breakthrough in the game of Poker is quite distinct from the previous findings due to the fact that Poker is an **imperfect-information** game. Imperfect-information games are games in which there is uncertainty about the current state of the game for one or more of the players. Whereas chess players both have exactly the same information about the state of the game, Poker players do not. Poker players each have a private hand which is only available to themselves. While the solutions to these games are interesting for recreational purposes, the techniques for solving such games also have broad applications outside of this domain. Many real-world interactions include some form of hidden information, for example in the context of negotiating, auctions or security.

The way in which perfect and imperfect-information games are solved through AI is different. For perfect-information games, the current state of the art techniques make use of Reinforcement learning combined with search algorithms. Reinforcement learning is an area of Machine learning which is concerned with how intelligent agents take actions in an environment to maximize a reward. By searching through a game tree, the valuations of the game states are updated iteratively. However, for imperfect-information games, the value of an action may depend on the probability that the action is chosen. This means that a state defined by a sequence of actions can have multiple valuations. Therefore, an AI system has to balance actions in such a way that it does not reveal too much private information about the game. Bluffing is an example of such an action for which the utility is dependent on the probability that one does that action. By bluffing too much, the bluff will get called out, which results in long term losses. Another difference is that in perfect-information games, parts of the game can be considered in isolation. In a chess endgame for example, all knowledge about chess openings is irrelevant. This means a chess end game can be considered by itself, without any regard for the previous moves. Imperfect-information games do not have this luxury as it may be the case that the optimal strategy of a subgame is dependent on the optimal strategy of a subgame that is not reached.

To deal with these difficulties, methods from the field of **Game theory** offer promising solutions. Game theory is a field developed mostly throughout the twentieth century, initially by mathematicians like John F. Nash and John von Neumann. It is the study of mathematical models to analyze strategic interactions between rational agents. This means that rather than just dealing with games, Game theory

is applicable in many real-world situations. During the twentieth century, for example, Game theory was used to strategize regarding the use of nuclear weapons in the cold war. Terms like 'mutually assured destruction' describe Game-theoretic problems that were investigated at the time. Since then, Game theory has been used in many fields like evolutionary biology, economics, computer science, philosophy and linguistics.

For heads-up no-limit Poker, the difficulties of imperfect-information games are solved using three main modules [Brown and Sandholm, 2018]. Since this game has 10^{160} unique decision points, the first module consists in creating an abstraction of the game which is much smaller and easier to solve. The module then creates game-theoretic strategies for this abstraction using a regret-minimizing algorithm. The resulting strategy is detailed for the early betting rounds but just an approximation of the later rounds. The second module constructs a fine-grained abstraction during play of the game for the later rounds and solves them in real-time. The third module enhances the initial strategy by filling in missing branches of the abstraction based on the actual moves of the opponent. While some of these techniques are specific to the domain of Poker, most of them can be applied more generally to other imperfect-information games.

In this project, a popular Dutch card game called **Toepen** will be investigated using these Game-theoretic methods. Toepen is an imperfect-information card game in which players play out their cards over a number of rounds. Throughout the rounds, players can bet to play for a higher number of points which get distributed after the last round. Winning rounds gives players an advantage in the next round. However, only the last round matters in terms of points. This means that a hand of Toepen requires players to think ahead and play in a way that makes them win the last round. A thorough explanation of the game will follow in Chapter 3.

The main focus of this research is to find methods that can be used to create agents that can play a game like Toepen well. This is done by investigating two techniques that are commonly used to find solutions to large imperfect-information games. The first technique is a solving algorithm called **Monte Carlo Counterfactual Regret Minimization** (MCCFR) which can be used to find optimal strategies in extensive-form games [Lanctot et al., 2009]. This is the algorithm which is used in the first module of the paper by Brown and Sandholm [2018] to find the optimal strategy for the abstraction of the game. This algorithm has theoretical guarantees to converge to an optimal strategy in an unlimited number of iterations.

The second technique is abstraction, which is used to simplify the game in order to make MCCFR a feasible technique in a game of this size. Abstraction is usually done in three stages. First, the game is abstracted to a smaller game, meaning decision points are grouped together. The solving algorithm then generates a strategy for this abstracted game. The strategy is then mapped back to the original game.

The thesis will be outlined in the following manner. In Chapter 2, the relevant background information from the existing literature is highlighted. In Chapter 3, two-player Toepen is modeled in a Game-theoretic framework as an extensive-form game. We then explain how both MCCFR and abstraction can be used to solve Toepen from a game-theoretic perspective. In Chapter 4, we do two experiments to investigate how the methods, explored in Chapter 3, perform in practice. A repository containing the code for the implementation of the methods and the experiments can be found at github.com/DaanLe/Toepen.

Chapter 2

Background

This chapter outlines the notation and terminology used throughout this thesis and gives the necessary background regarding the methods used.

2.1 Extensive-form Games

Extensive-form games model multiagent interactions in the form of a tree which captures the sequential nature of the interaction and are often used for their game-theoretical properties. Each node in the tree represents a decision point for a player or a chance event. Extensive-form games are useful models for representing imperfect-information games. These are games in which there is information which is not available to all players, such as a closed hand in a card game. Formally, an extensive-form game with imperfect information can be described as a tuple $\langle N, H, Z, \underline{i}, A, \underline{A}, s, \mathcal{I}, u \rangle$ in the following manner [Osborne and Rubinstein, 1994]:

- Let N be a finite set of **players**.
- Let H be a finite set of **nodes** or **histories** of actions which contains all the information of the situation up until that point. This set is constructed such that the empty sequence is in H and every prefix of a sequence in H is also in H. Define $h \sqsubseteq h'$ to mean that h is a prefix of h'. Let $Z \subseteq H$ be the set of terminal histories which are not a prefix of any other sequence. These are the points at which the game ends.
- Let $\underline{i} : H \backslash Z \to N \cup \{c\}$ be a **player function** which assigns each non-terminal node to a player in N or chance which is denoted as c.
- Let A be a finite set of the possible actions in the game and let $\underline{A} : H \to 2^A$ be an **action function** which maps a non-terminal node h to a set of the available actions in that node.
- Let $s : H \times A \to H$ be a **successor function** which returns the next node h', given the previous node h and an action a.
- Let $\mathcal{I}_i$ be a partition of $\{h \in H \mid \underline{i}(h) = i\}$ where any two nodes $h, h' \in I$ with $I \in \mathcal{I}_i$, are indistinguishable for player i. We require that $\underline{A}(h) = \underline{A}(h')$ which can also be expressed as $\underline{A}(I)$ and $\underline{i}(h) = \underline{i}(h')$ which can be expressed as $\underline{i}(I)$. The set Z_I is a the set of all terminal nodes with a prefix in I and for $z \in Z_I$, let $z[I]$ be that prefix. $\mathcal{I}_i$ is called the **information partition** of player i and $I \in \mathcal{I}_i$ is called an **information set** of player i.
- Let $u_i : Z \to \mathbb{R}$ be a **payoff function** for each player $i \in N$ such that each terminal node is mapped to a reward. Next define $\Delta_i = \max_{h \in Z} u_i(h) - \min_{h \in Z} u_i(h)$

as the range of utilities to player $i \in N$ and $\Delta = \max_{i \in N} \Delta_i$ as the range of utilities for the game.

Here we will only discuss **two-player zero-sum extensive-form games** which means $N = \{1, 2\}$ and $u_1 = -u_2$. Furthermore, we will assume **perfect recall** which means that players will always be able to access their previous actions and corresponding information sets.

2.2 Strategy

A **strategy** $\sigma_i(I)$ is a stochastic vector over $\underline{A}(I)$ for a given information set $I \in \mathcal{I}_i$. The probability of a particular action a being chosen will be denoted as $\sigma_i(I, a)$. In the literature, this type of strategy is known as a behavioral strategy since the probabilities are given locally for each information set as opposed to a mixed strategy where the probability vector is over all actions in A_i. For finite imperfect-information games with perfect recall, any behavioral strategy is outcome-equivalent to some behavioral strategy and vice versa [Kuhn, 1953]. Next we define σ_i to be a **strategy for player** i in each information set $I \in \mathcal{I}_i$ and Σ_i as the set of all possible strategies for player i. A **strategy profile** $\boldsymbol{\sigma}$, will be a tuple containing a strategy for each player. With $\boldsymbol{\sigma}_{-i}$ we will denote all the strategies in $\boldsymbol{\sigma}$ except for σ_i.

Let $\pi^\sigma(h)$ be the probability of node h occurring if all players play according to $\boldsymbol{\sigma}$. This is formally given as $\pi^\sigma(h) = \prod_{h' \text{ s.t. } h' \sqsubseteq h} \sigma_{\underline{i}(h')}(h', a)$. This simply means multiplying all the probabilities in the strategy which led to the node. This value can be decomposed as $\pi^\sigma(h) = \prod_{i \in N \cup \{c\}} \pi_i^\sigma(h)$ where $\pi_i^\sigma(h)$ is a player's contribution to the total probability. This contribution can be seen as the chance of getting to the node, given that all other players play to get to the node. $\pi_{-i}^\sigma(h)$ is the contribution of all players except for player i as well as chance. Let $\pi^\sigma(h, z) = \pi^\sigma(h) / \pi^\sigma(z)$ if $h \sqsubseteq z$, and zero otherwise. This can be seen as the probability of reaching a terminal node from a non-terminal node, given the strategy profile. For $I \subseteq H$, define $\pi^\sigma(I) = \sum_{h \in I} \pi^\sigma(h)$ which is the probability of reaching I given $\boldsymbol{\sigma}$ with $\pi_i^\sigma(I)$ and $\pi_{-i}^\sigma(I)$ defined in the same way. The expected payoff of a strategy profile $u_i(\boldsymbol{\sigma}) = \sum_{h \in Z} u_i(h) \pi^\sigma(h)$ is the result of summing up the utilities of the terminal nodes scaled according to the probabilities of reaching those nodes.

A strategy σ_i^* is a **best response** for player i to the partial strategy profile $\boldsymbol{\sigma}_{-i}$ if $u_i(\sigma_i^*, \boldsymbol{\sigma}_{-i}) \geq u_i(\sigma_i', \boldsymbol{\sigma}_{-i})$ for all $\sigma_i' \in \Sigma_i$. The **best-response value** is then the expected value of that strategy for that player $b_i(\boldsymbol{\sigma}_{-i}) = \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \boldsymbol{\sigma}_{-i})$

A **Nash equilibrium** is a strategy profile in which each σ_i is a best response to $\boldsymbol{\sigma}_{-i}$ for every player $i \in N$. In a two-player zero-sum game, a Nash equilibrium has the property that any player who plays according to a Nash equilibrium strategy is guaranteed to not lose in expectation regardless of the strategy of the opponent. This result is due to the fact that each player plays a best response which means that no player can unilaterally change its strategy to gain a higher expected utility. In a two-player game, this then means that if one player plays according to the Nash equilibrium, the other player has the best response of also playing according to the Nash equilibrium and all other strategies will cause her to lose in expectation. For unbalanced games where the game gives an inherent advantage to one of the players, the property of not losing in expectation only holds if the players are equally

likely to be this player. An ϵ-**Nash equilibrium** is an approximation of a Nash equilibrium where the following holds:

$$\forall i \in N \quad u_i(\sigma) + \epsilon \geq \max_{\sigma_i' \in \Sigma_i} u_i(\sigma_i', \sigma_{-i}) \tag{2.1}$$

If $\epsilon = 0$ then σ is a true Nash equilibrium. For two-player zero-sum games, we can use the metric of **exploitability** of a strategy $\epsilon_\sigma = b_1(\sigma_2) + b_2(\sigma_1)$ to determine how close a strategy profile is to a Nash equilibrium.

2.3 Counterfactual Regret Minimization

Counterfactual Regret Minimization (CFR) is a method to find Nash equilibria in extensive-form games in an iterative way originally proposed by Zinkevich et al. [2008]. It is important to note that during CFR, the algorithm is not actually playing the game but rather just adjusting the strategies for both players. Therefore the algorithm has perfect information, meaning it has access to the entire strategy profile σ which makes calculating values like $\pi_{-i}^{\sigma}(I)$ possible. To use CFR, we first need a notion of regret. Regret can be intuitively understood as a measure of how much a player would gain if it had used a different strategy. As this regret is defined over the span of multiple iterations of the game, we use σ_i^t to denote the strategy profile on iteration t. The **counterfactual value** $v^{\sigma}(I)$ is the expected payoff of player $i = \underline{i}(I)$ when reaching I, weighted by the probability of reaching I if i played to reach I. Intuitively this can be understood as a "counterfactual" as it is the expected payoff of player i reaching I if the player had tried to do so, keeping the strategy profile equal for everything else. This is formally given by:

$$v^{\sigma}(I) = \sum_{z \in Z_I} \pi_{-i}^{\sigma}(z[I]) \pi^{\sigma}(z[I], z) u_i(z) \tag{2.2}$$

Next we define $v^{\sigma}(I, a)$ in the same manner except there is now a 100% probability of playing action a at I. Formally:

$$v^{\sigma}(I, a) = \sum_{z \in Z_I} \pi_{-i}^{\sigma}(z[I]) \pi^{\sigma}(s(z[I], a), z) u_i(z) \tag{2.3}$$

The **instantaneous regret** is then defined as:

$$r^t(I, a) = v^{\sigma^t}(I, a) - v^{\sigma^t}(I) \tag{2.4}$$

This is called the instantaneous regret because it is a measure of how much a player regrets, not always playing action a, rather than playing the current strategy at this information set. The **counterfactual regret** $R^T(I, a)$ is the sum of the instantaneous regret $r^t(I, a)$ over T time steps. Next define the **positive counterfactual regret** $R_+^T(I, a) = \max\{R^T(I, a), 0\}$ and the **maximal positive counterfactual regret** $R^T(I) = \max_a R_+^T(I, a)$. The **total regret** of player i at iteration T is computed as follows:

$$R_i^T = \max_{\sigma_i^* \in \Sigma_i} \sum_{t=1}^{T} (u_i(\sigma_i^*, \sigma_{-i}^t) - u_i(\sigma^t)) \tag{2.5}$$

The **average strategy** σ_i^{-T} of player i from time 1 to T is defined as a distribution over all actions $a \in \underline{A}(I)$ for each information set $I \in \mathcal{I}_i$, formally defined as:

$$\sigma_i^{-T}(I) = \frac{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)}{\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)} \tag{2.6}$$

In two-player zero-sum games, if both player's average total regret $\frac{R_i^T}{T}$ is less than ϵ at time step T, then $\langle \sigma_1^{-T}, \sigma_2^{-T} \rangle$ is a 2ϵ-Nash equilibrium [Waugh, 2009]. This means that if we can bound the average total regret, we can approach an ϵ-Nash equilibrium.

The goal of CFR is to minimize regret on each information set which can be done using a number of different algorithms. An algorithm is regret-minimizing if the average total regret of all players goes to zero as t goes to infinity. A common algorithm for this purpose is **regret matching** which is developed by Hart and Mas-Colell [2000]. Here the strategy of a player is a distribution over the actions in an information set which is proportional to the positive regret of each action. This is formally given as:

$$\sigma^{t+1}(I, a) = \frac{R_+^t(I, a)}{\sum_{a' \in \underline{A}(I)} R_+^t(I, a')} \tag{2.7}$$

If $\sum_{a' \in \underline{A}(I)} R_+^t(I, a') = 0$ the strategy will be a uniform distribution over all possible actions. Cesa-Bianchi and Lugosi [2006] have shown that if a player selects actions according to regret matching, then on iteration T, $R^T(I) \leq \Delta\sqrt{|\underline{A}(I)|}\sqrt{T}$. One of the key results in the original CFR paper by Zinkevich et al. [2008] is that $R_i^T \leq \sum_{I \in \mathcal{I}_i} R^T(I)$, which means that CFR can be used in self-play to compute a ϵ-Nash equilibrium in two-player zero-sum games. Moreover, the bound on the average total regret is linear in the number of information sets and since only the values at each information set have to be stored, the space complexity is $\mathcal{O}(|\mathcal{I}|)$. However, each iteration still requires an entire traversal of the game tree.

2.4 Monte Carlo Counterfactual Regret Minimization

The obstacle of having to traverse the entire game tree on each iteration can be circumvented using Monte Carlo Counterfactual Regret Minimization (MCCFR) [Lanctot et al., 2009]. MCCFR restricts the number of terminal histories which are considered at each iteration. In practice, this means that we only update the strategy for a part of the tree which leads to a subset of terminal nodes, rather than traversing the entire tree.

To do this, let $\mathcal{Q}$ be a partition on Z for which we call a subset $Q \in \mathcal{Q}$ a **block**. An iteration will consist of sampling one block and only considering the nodes which lead to the terminal nodes in that block. The way in which these blocks are divided and the probability with which a block is chosen define the MCCFR algorithm. **External-sampling MCCFR** is a commonly used option for its simplicity and strong performance. This version samples choices external to a given player, meaning the actions of the opponent and chance. This means we have a block for every pure strategy of the opponent and chance. The probability of sampling a block is equal to the probability of this block occurring based on σ_{-i} and chance. Rather than tracking the instantaneous regret $r^t(I, a)$, we now track the instantaneous sampled regret $\tilde{r}^t(I, a)$. For information sets sampled at iteration t, $\tilde{r}^t(I, a)$ is equal to $r^t(I, a)$ divided by the probability of sampling I, while for unsampled information sets $\tilde{r}^t(I, a) = 0$.

The instantaneous sampled regret will match the instantaneous regret in expectation. The player's strategy is updated using regret matching, after which the process is repeated for the other player to finish an iteration. The player for which the strategy is updated is called the **update player**. From now on, when we use MCCFR in this research, we refer to external-sampling MCCFR unless specified otherwise.

When using MCCFR, for any $\rho \in (0, 1]$ the total regret is bounded by:

$$R_i^T \leq (1 + \frac{\sqrt{2}}{\sqrt{\rho}})|\mathcal{I}_i|\Delta\sqrt{|A_i|}\sqrt{T} \tag{2.8}$$

with probability $1 - \rho$ where $|A_i| = \max_{h \text{ s.t. } \underline{i}(h)=i} |\underline{A}(h)|$. This means that although MCCFR requires the same number of iterations for a worse lower bound than vanilla CFR, each iteration only traverses a fraction of the tree. For balanced games where both players make an equal number of decisions, the iteration cost of MCCFR is $\mathcal{O}(\sqrt{H})$ while vanilla CFR has an iteration cost of $\mathcal{O}(H)$.

2.5 Abstraction

Over the last decade, **abstraction** has proved to be an important part of solving large imperfect-information games [Sandholm, 2015a]. At its core, most abstraction techniques consist of three modules. First, the game is abstracted to a smaller game which is strategically similar to the original game. Then an ϵ-Nash equilibrium strategy is computed for this smaller game, after which this strategy is mapped back to the original game.

There are two overarching types of abstraction in imperfect-information games, namely: information abstraction and action abstraction. Information abstraction aims to reduce the number of information sets in the game by mapping the nodes of the game to a smaller number of information sets. Grouping together nodes that were previously distinguishable can be seen as forgetting some of the information which was used to keep them apart. Good information abstraction only groups together similar nodes. Action abstraction aims to restrict the number of actions available to a player to decrease the branching factor of the game tree, which then reduces the number of information sets that way.

2.5.1 Information Abstraction

There are two different types of information abstraction. The first type is lossless abstraction. As the name implies, this is a type of abstraction for which the abstract game still contains all the relevant information of the original game. An example of lossless abstraction in card games is card isomorphism, where there are certain hands of cards that are strategically identical, even though they are actually different. In a game of Texas Hold'em Poker, the hands '2 of hearts'-'3 of spades' and '2 of clubs'-'3 of diamonds' are strategically identical and are often just referred to as '2-3 offsuit'. In card games containing four suits where different suits do not have an inherent role in the mechanics of the game, card isomorphism can reduce the size of the game by at most a factor of $4! = 24$.

In a lossless abstraction, the solution maps back to an equal ϵ-Nash equilibrium solution in the original game. A more advanced lossless abstraction algorithm called GameShrink was able to solve Rhode Island Hold'em, which is a version of Poker with 3.1 billion nodes in the game tree [Gilpin and Sandholm, 2007b]. While this is a general algorithm applicable to games other than Poker, it does require that

the available actions are independent from the private information which is only available to one of the players. Toepen unfortunately does not have this property, as the playable cards depend on the hand of the player, which is private information.

The other type of information abstraction is lossy abstraction. In a lossy abstraction, information is intentionally ignored or forgotten in order to map more possible histories to a smaller number of information sets. This comes at the cost of having a solution which is not equal to the solution in the original game. Many games are still too large after lossless abstraction, which is why lossy abstraction is often needed. The main ways of doing lossy information abstraction in practice contain some form of bucketing and imperfect recall.

Bucketing is the act of grouping together distinct information sets. Information sets which are grouped together should be strategically similar in order to have a solution which is still viable in the original game. This can be done based on metrics such as 'strength' which can be derived from the probability to win, lose or draw from an information set. Information sets with similar scores on this metric can then be grouped together using a clustering algorithm like K-means [MacQueen, 1967]. While early Poker AI systems used this metric [Gilpin and Sandholm, 2007a], newer Poker AI systems use more sophisticated methods such as potential-aware abstractions [Ganzfried and Sandholm, 2014]. These are abstractions where information sets are bucketed based on the probability distribution over the strength in all future rounds of the game. The distance between these distributions is computed using the Earth Mover's Distance (EMD). Informally, if the distributions are seen as piles of earth, EMD can be seen as the lowest cost of turning one pile into the other; where cost is defined as the amount of earth moved, multiplied by the distance by which it is moved.

An Imperfect-recall abstraction is an abstraction where players forget some of the information about the history of the game. There are several problems with this type of abstraction. One problem is that this method removes the theoretical guarantees of both CFR and MCCFR. Another, bigger problem is that Nash equilibria are not even guaranteed to exist in behavioral strategies in games with imperfect recall [Wichardt, 2008]. However, Waugh et al. [2009b] have shown that Poker agents which use this type of abstractions are consistently able to outperform Poker agents which use perfect recall abstractions. A possible explanation for this is that there is the presence of a trade-off between the different sorts of information on which a decision is based. By forgetting some information about the past, the more relevant information about the current state of the game can be more refined which allows for better choices in practice.

One of the big challenges of lossy abstractions in games, as opposed to single-agent settings, is that they can be nonmonotonic [Waugh et al., 2009a]. This means that a strategy which is computed on a fine-grained abstraction, meaning an abstraction closer to the original game, can be worse in the original game than a coarser abstraction. The reason for this is that the opponent can play actions outside of the abstraction, whereas in a single-agent setting all actions fall in the abstraction which does ensure monotonicity. This makes the theoretical grounding of lossy abstractions weak in general. Despite this, lossy information abstraction seems to work well in practice and has been a staple in the recent artificial Poker agents [Brown and Sandholm, 2018, 2019].

2.5.2 Action Abstraction

Action abstraction is a type of abstraction which reduces the number of actions which can be taken from an information set. This is especially needed in games where there exist information sets which have a continuous or large set of actions. In no-limit Poker, this is the case as bets can range from the big blind all the way up to the entire stack which can often be as much as a hundred big blinds. With this type of branching the game tree quickly becomes intractable. Luckily, many of these actions are very similar and can be grouped together to reduce this problem. This can be done manually by using a heuristic, but in the last decade there have also been developments in automated action abstraction techniques [Brown and Sandholm, 2014, Sandholm, 2015b,a]. Since Toepen does not have continuous actions, this type of abstraction will not be used in this research.

Chapter 3

Strategy Design

This chapter contains techniques which can be used to generate strategies for a game like Toepen. First we will explain Toepen in detail, after which we will go over MCCFR and abstraction in the context of Toepen.

3.1 Toepen

This section consists of an examination of the game of Toepen. First, the rules of the game are outlined, both formally and informally. After that, the size complexity of the game is investigated and some simpler versions of the game will be introduced. Lastly, some theoretical properties of Toepen are explored.

3.1.1 Basic Toepen

Toepen is a Dutch card game which is played with a subset of a normal deck of cards, containing only cards higher or equal to 7. The goal of the game is to stay below a certain bound, usually 15 points, where players gain penalty points if they lose a **hand**. The game can be played with 2 to 8 players. The ranking of the cards is as follows, from high to low: 10-9-8-7-'Ace'-'King'-'Queen'-'Jack'. A hand plays out as follows. Each player gets dealt four cards which all get played out over the course of four rounds. The first player to act in a hand is called the **starting player**. The first player to act in a round is called the **active player** as she has the initiative for this round. This means that the card she plays decides the suit for that round. All players have to play a card of this suit if they have it. If they have multiple cards of that suit, they can choose which of those cards they play. If a player does not have the suit, she can play any card but is unable to win the round. These players are called the **reactive players** as they have to react to the card of the active player. The player who plays the highest card of the suit determined by the active player is the winner of the round and becomes the active player for the next round. This means this player gets to start the next round and decide the suit for that round. However, only the winner of the last round is the winner of the hand, which means she does not get any penalty points. The number of penalty points the other players get is dependent on the number of bets within that round. The round starts with an ante of 1 point. All players can bet at any point in the game to increase the ante with 1 point, except if they were the last player to bet or if their bet causes the sum of their penalty points and ante to become higher than the bound. After a bet each player gets the choice to fold, adding the old ante to their point total, or to play on for the new ante. After a hand is finished, the starting player is rotated. Players who reach the bound leave the game until a single winner remains.

For this research, we will only consider the two-player version of Toepen and we will just consider one hand at a time. While the strategy of a player within a

hand can somewhat change based on the penalty-point tally, the strategy in a hand should be independent of this tally if the players are not within reach of the bound during the hand. Furthermore, we only allow for betting by the players before they play a card. This reduces the size of the game and is also expected behavior since waiting with betting until a player has to play her card reveals the most information to that player while hiding the information of that card from the other player. To avoid confusion, we will also invert the penalty-point system which means that the winner of a hand will get points, rather than the loser.

More formally, a two-player hand of Toepen can be defined in the framework of extensive-form games in the following manner:
Toepen is a tuple $\langle N, H, Z, \underline{i}, A, \underline{A}, s, \mathcal{I}, u \rangle$ where:

- $N = \{1, 2\}$
- $H = \{h_1, ..., h_t\}$, where t is the total number of nodes.
- $Z \subseteq H$ are the terminal nodes which are not a prefix of any other sequence.
- $\underline{i} : H \backslash Z \rightarrow N \cup \{c\}$ is a function which maps each non-terminal history to a player in N or chance. In Toepen, only the first action is taken by chance, since chance only determines the cards which are dealt at the beginning of a hand. After the chance action, player 1 is the first player to act.
- A = {'Bet', 'Check', 'Call', 'Fold'} $\cup$ (Rank $\times$ Suit) $\cup$ { $\langle x, y \rangle$ | $x, y \in \mathcal{P}$(Rank $\times$ Suit) and $|x| = |y| = 4$ and $x \cap y = \emptyset$}, where Rank = {10, 9, 8, 7, 'Ace', 'King', 'Queen', 'Jack'}, Suit = {'Clubs', 'Diamonds', 'Hearts', 'Spades'}, $\mathcal{P}$(Rank $\times$ Suit) indicates the power set of all possible card combinations and $\langle x, y \rangle$ indicates a tuple containing the two starting hands, x for player 1 and y for player 2, as sets of size 4 containing cards as elements.
- $\underline{A} : H \rightarrow 2^A$ is an action function which returns the available actions given a node. One can see that A consists of the union of three sets, namely: the betting actions, the card playing actions and the card dealing actions. The card dealing actions happen only at the start of the hand, while the betting and card playing actions alternate throughout the rest of the hand. Betting actions can be further subdivided into active betting which is the set {'Bet', 'Check'} and reactive betting which is the set {'Call', 'Fold'}
- $s : H \times A \rightarrow H$ is a successor function which maps a node and action to a new node. This function encodes the rules of the game.
- $\mathcal{I}_i$ is a partition of $\{h \in H \mid \underline{i}(h) = i\}$, where any two nodes are indistinguishable for player i. For Toepen, this is the case for all $\{h \in H \mid \underline{i}(h) = i\}$ for which the histories $h' \sqsubseteq h$ are reached with the same sequence of actions, except for the card dealing action $\langle x, y \rangle \in \underline{A}(h_1)$ which can differ in the set corresponding to the hand of the opponent.
- $u_i : Z \rightarrow \mathbb{R}$ is a payoff function where $u_1 = -u_2$ as this is a zero-sum game. Further $u_i(z) = \pm|\{h \sqsubseteq z \mid s(h', a) = h \text{ and } a = \text{'Call'}\}|$ which means the payoff or ante of the hand is the number of calls in the history of the hand.

To illustrate how a hand of Toepen can play out, consider the following example:

Example 3.1.1. Two players, Alice and Beth, are playing a hand of Toepen against each other. Alice is the starting player. Alice has a hand containing a king of hearts,

a jack of spades and a 8 and 10 of diamonds. Beth has a hand containing an ace of hearts, a king and 9 of clubs and a 7 of diamonds.

Alice starts off by playing her jack of spades. Since Beth does not have a card with the spades suit, she can play any of her cards but is unable to win the round. Beth plays the king of clubs, after which the round ends with Alice as the winner. Alice starts the next round by playing the king of hearts. Beth has to play the ace of hearts since it is her only card of the same suit. However, before she plays her card, she decides to bet an extra point. Alice knows she will probably lose the initiative this round, but since the rest of her hand is strong, she decides to call the bet. The ante of the game is now 2 points. Beth plays her ace of hearts and wins the round. Beth starts the next round by playing her 9 of clubs. Alice does not have a card of the clubs suit, and since all the cards she has left are of the diamond suit, she decides to play the lowest rank which is the 8 of diamonds. Beth wins the round, which means she starts off the last round. This is the only round that matters in terms of winning or losing the hand. Beth plays out her remaining card which is the 7 of diamonds. Alice has the 10 of diamonds remaining in her hand, so she knows she will win the game. Therefore, she decides to bet another point. Beth now has to decide whether Alice's bet is a bluff or a value bet. Beth knows that Alice's remaining card can not be of the clubs suit since she would have had to play it in round 3. She also knows that it is probably a high card since Alice called her bet in round 2, even though Alice knew she would probably lose the initiative. After some thinking, Beth decides to call. Alice plays her 10 of diamonds which means she wins the game and gets 3 points.

3.1.2 Size Complexity

The size complexity of Toepen is outlined here. First of all, there are many decision points which means the extensive-form game representation contains many information sets for which a strategy has to be computed. For a two-player game using the basic rules, there are a total of 32 different cards from which 4 cards are drawn for each hand. This means that there are already $\binom{32}{4} \cdot \binom{28}{4} \approx 7.4 \cdot 10^8$ distinct hand configurations for two-players. To get the number of possible histories for a round of Toepen, this number must be multiplied by by the number of possible actions for the players. This leads to $\binom{32}{4} \cdot \binom{28}{4} \cdot 4^2 \cdot 3^2 \cdot 2^2 \approx 4.2 \cdot 10^{11}$ possible ways of playing the hand without betting. This number is higher than the actual possible histories in a game since the possible actions of the second player of the round are confined by the suit of the card which the first player played. Therefore this will be an upper bound, rather than the actual number of possible histories. Next, betting has to be taken into account. As discussed earlier, betting directly after a previous bet does not make much sense in practice since you could also wait until your opponent played her next card before betting to have more information. Since we expect players to behave in this manner, the increase of possible histories would be by a factor of 2^8, as each player could bet before playing each of their four cards, and each bet has a branching factor of two. This leads to a total of $1.1 \cdot 10^{14}$ possible histories per hand.

For a player who has to create a strategy, storing the entire game tree in memory is not necessary. Players only have access to the information set which a node belongs to, meaning that only the size of the information partition is important for this algorithm. For each player this can be computed by taking the number of possible hands for that player and then for each round, multiplying by the number of possible cards of the opponent and the playable cards. This would result in $\binom{32}{4} \cdot 28 \cdot 4 \cdot 27 \cdot 3 \cdot 26 \cdot 2 \approx 1.7 \cdot 10^{10}$ possible decision points. This is again higher than

Suits	Ranks	Hand size	Betting	Hand configs	Upper bound	Info sets
2	2	2	False	6	48	10
2	2	2	True	6	1,536	496
2	3	2	False	15	240	39
2	3	2	True	15	7,680	3,552
3	2	2	False	15	240	51
3	2	2	True	15	7,680	3,576
3	3	2	False	36	1,008	180
3	3	2	True	36	32,256	22,338
3	3	3	False	84	30,240	4,584
3	3	3	True	84	3,870,720	1,366,338

TABLE 3.1: The actual number of information sets in some smaller versions of the game.

the actual number of information sets due to restrictions in the possible actions. Betting adds the same complexity as before, namely a factor of 2^8 leading to $4.3 \cdot 10^{12}$ information sets per player per hand. Since the second player has to act after the first player's first action, the betting complexity adds a factor of 2^9 for this player. For two-players this leads to $1.7 \cdot 10^{13}$ information sets per hand. The practical number of information sets for some small versions of the game is shown in Table 3.1. As we can see, the upper bound is quite liberal.

While this is a large number of information sets, there have been games of similar size which are solved using a version of CFR called CFR+ [Bowling et al., 2015]. However, this implementation required more than 900 core-years of computation and 10.9 terabytes of disk space at the time. This is an immense amount of resources that will not be used for this research. Instead, MCCFR will be tested on smaller versions of the game. This allows for more extensive research on which features of the game cause the most difficulty for the algorithm. These smaller versions of the game will be explained next.

3.1.3 Smaller Versions

To reduce the size of the game, the game is reduced to smaller versions. First of all, only a heads-up version of the game is examined, meaning a game with only 2 players. Next, one hand is considered at a time. As mentioned earlier, the strategy in a hand should be independent of this tally if the players are not within reach of the bound during the hand. Next, a variable deck and number of rounds will be used. This means the game can be played with any number of cards and suits and for any number of rounds. Therefore, the complexity can be reduced or increased based on the intermediate findings of the research. The betting option will also be variable, which allows for a version of the game with no betting. In this version of the game, players cannot bet to play for a higher ante. These smaller versions of the game are not the same as abstractions but rather new original games which preserve the structure of the game but reduce the size. This is similar to Kuhn Poker [Kuhn, 1950] and Leduc hold'em Poker [Southey et al., 2005]. These are small versions of Texas hold'em poker containing only three and six cards, respectively, and two betting rounds. These versions of Poker are often used in research as a proof of concept for newly developed techniques.

3.1.4 Theoretical Properties

Toepen has some interesting properties which set it apart from other card games. The game contains four different types of actions, namely: active betting, reactive betting, active card playing and reactive card playing.

Betting actions are similar to the actions in Poker. This makes Toepen somewhat similar to Poker, although there are still many differences. Poker has chance events throughout the game which are unknown to any of the players, while Toepen only has chance events at the beginning of a hand. Once the hands are dealt in Toepen, the strength of the hand is entirely dependent on the hand of the other player the strategy profile. In Poker, if both the strategies and the hands of the players are known, there is still uncertainty about the outcome of a game. This makes evaluation of a hand different for each game, since concepts like potential-aware evaluation cannot be computed in the same way in Toepen, as in Poker. Evaluating a hand in Toepen is difficult since the quality of the cards is hard to determine based on the individual cards themselves. This is the case, because different combinations of suits and ranks have different strengths and weaknesses, which are also based on which player is the active player. Having many cards of the same suit is often good if you are the active player, but bad if the opponent is the active player. Consider the following simple example:

Example 3.1.2. Let us assume a player has a hand containing the 7, 8, 9 and 10 of hearts. This would be considered a perfect hand if this player is the active player. In this case, she is guaranteed to win the hand. However, if the opponent is the active player the hand suddenly becomes quite bad. The hand contains half of all the heart cards in the game, so the chance of the opponent having one is small. If the opponent does not have a heart, the hand always results in a loss.

If the game did not contain betting, it would be more straightforward to play. Many of the actions would become strictly dominated in this setting, meaning there exists no situation where this action leads to a higher utility than some other action. However, the game does contain betting which makes it so that these strictly dominated actions can now be used to bluff or mislead the opponent. Consider for example the following situation:

Example 3.1.3. Player 1 has a hand containing the 10 of clubs and the 8 of clubs. Player 2 won the previous round and starts this round off with a 8 of hearts. In a game without betting, playing the 8 of clubs would always be better as the 10 of clubs would be a better card in the last round, and the current round cannot be won. This means that playing the 10 of clubs is a strictly dominated action, and there is no possible scenario where playing the 10 of clubs is better than playing the 8 of clubs. If we introduce betting, this is no longer the case. By playing the 10 of clubs in the current round, player 1 reveals that she could have a really strong hand, containing for example one of the other 10's in the deck. If player 2 would now play a 9 of diamonds in the last round, player 1 can bluff and bet. Playing the 10 of clubs made the bluff more believable in the eyes of player 2, who is now more likely to fold.

While it is not clear that this would actually be a better action, it is no longer the case from a game-theoretic point of view that this is a strictly dominated action.

3.2 MCCFR

MCCFR is the algorithm which can solve the game once the game is properly defined. In this section we will go over the specifics of this algorithm and the implementation in practice.

3.2.1 Algorithm

To use MCCFR, we first need both the action function and the successor function of Toepen. To use these functions, we introduce the concept of a **game state**. A game state is similar to a node, but it contains all the data of the game explicitly, rather than implicitly, as is the case with nodes. This means a game state contains the player who has to act, the hands of both players, the history containing all played actions, whether the state is a terminal state and the ante of the hand.

Using this information, we can define the action function, which returns the possible actions at a given game state. The function distinguishes between 4 types of actions, namely: active betting, reactive betting, active card playing and reactive card playing. Active betting happens at the beginning of the game and after a card has been played. However, a player cannot bet if she was the last player to bet. This returns the actions {'Bet', 'Check'} to the player. Reactive betting only happens when the previous action was a 'Bet' by the other player. In this case, the returned actions are {'Call', 'Fold'}. Active card playing happens at the start of each round by the player who won the previous round, after this player's betting action. If this is the case, the player can play any of the cards in her hand. Reactive card playing is a bit more complicated. Reactive card playing is the last action to happen in the round, after there is one card in play and all betting for the round is finished. The available actions are dependent on the suit of the card which was played by the active player and the hand of the player. If the player has any cards of this suit in her hand, the player has to play one of them. If the player does not have any card of this suit in her hand, she can play any of the cards in her hand.

The successor function takes a game state and action and returns the next game state. The first thing to determine for the next game state is whether it is a terminal state or not. The next game state is terminal if either the end of the last round is reached, meaning all cards have been played, or when the action is 'Fold'. In all other cases the next game state is a non-terminal game state. Next, the ante of the next state can be determined by taking the ante of the current game state and adding one to it of the action is 'Call'. The player which has to act in the next game state is only the same player if the action is 'Check' or if the player won the previous round as a reacting player. In all other cases, the player who has to act is switched to the opponent. Lastly, the action is added to the history of actions which have been played, and if the action was a card, it is removed from the hand of the player.

Now that the action and successor functions are both defined, they can be used in an MCCFR implementation to generate a ϵ-Nash equilibrium for the game. The MCCFR function is a recursive function which takes in a game state, the probability of reaching that game state π^{σ^t} and the update player i for which the strategy is currently being updated. The base case for this recursion is a terminal game state which returns the ante for the hand at the moment the hand ended. The initial input is a game state where the cards are sampled uniformly from the deck and no actions have been played. For the update player, each of the possible actions is generated using the action function, after which the successor function is used to find the new game state, which is then used for the next recursive step. For the other

player, a random action is sampled according to the average strategy of that player at the current iteration $\pi_{-i}^{\sigma^t}$. After the base case is reached, the instantaneous sampled regret $r^{\tilde{t}}(I,a)$ for each reached information set for the update player is computed by using equation 2.4 and dividing by the probability of that information set occurring, given the strategy of the opponent. This value is then added to the counterfactual regret $R^T(I,a)$ which is just the sum of all instantaneous sampled regrets for a given information set and action. The strategy for the next iteration is computed using equation 2.7. The counterfactual regrets are stored in a dictionary where the keys are the information sets. Alongside these values, the sum of weighted strategies $\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)$ is also stored. This is the sum over the strategies multiplied by the probability of the update player reaching the information set using the her strategy. The algorithm then changes the update player and starts again for a given number of iterations.

After the algorithm is done, one has a dictionary containing the sum of weighted strategies for all information sets which were reached during training. These values can then be normalized using equation 2.6 to compute the average strategy. This average strategy is theoretically guaranteed to converge to an ϵ-Nash equilibrium over time.

3.2.2 Example

While the MCCFR algorithm can seem complex, it is quite easy to understand in practice. Consider the following practical example which is an altered version of an example by Johanson [2016] for Toepen of two iteration of MCCFR on one of the information sets:

Example 3.2.1. Let us consider the actual first iteration which means the strategy profile σ^1 contains only uniform probabilities over all choices in the game. A visual representation of this example is given in Figure 3.1. Consider the information set where a player has three possible actions of playing a hand containing a 10 of clubs, 7 of clubs and 8 of hearts. We will refer to these actions as action 1, action 2 and action 3, respectively. Since it is the first iteration, the player's strategy is $\frac{1}{3}$ probability for each action. Let us assume the probability of reaching this information set is $\pi_{-i}^{\sigma^t}(I) = 0.2$ if the current player was actively playing to reach this information set. Now we assume that the expected value of the information sets which are reached by actions 1, 2 and 3 are -3, 6 and 3, respectively. This makes the counterfactual value of the current information set equal to $0.2 \cdot (-3 \cdot \frac{1}{3} + 6 \cdot \frac{1}{3} + 3 \cdot \frac{1}{3}) = 0.4$. We can now calculate the instantaneous regret of each action by subtracting the counterfactual value from the value of always taking that action. This would result in a regret of $0.2 \cdot -3 - 0.4 = -1$ for action 1, $0.2 \cdot 6 - 0.4 = 0.8$ for action 2 and $0.2 \cdot 3 - 0.4 = 0.2$ for action 3. This can intuitively be understood as the player regretting to not play actions 2 and 3 more often while also regretting she plays action 1 as often as she does with the current strategy. These regret values are stored and accumulate as the counterfactual regret whenever the information set is reached again. The next step consists of calculating the strategy $\sigma^{t+1}(I,a)$ of the next iteration using equation 2.7. This is done in proportion to the positive counterfactual regret. This means the probability of choosing action 1 becomes 0 as it has negative counterfactual regret, the probability of choosing action 2 becomes $0.8/1 = 0.8$ and the probability of choosing action 3 becomes $0.2/1 = 0.2$. Finally, we also store the the sum of weighted strategies $\sum_{t=1}^{T} \pi_i^{\sigma^t}(I)\sigma^t(I)$. Let us assume $\pi_i^{\sigma^t}(I) = 0.4$ which would result in 0, $0.4 \cdot 0.8 = 0.32$ and $0.4 \cdot 0.2 = 0.08$ for actions 1, 2 and 3, respectively.

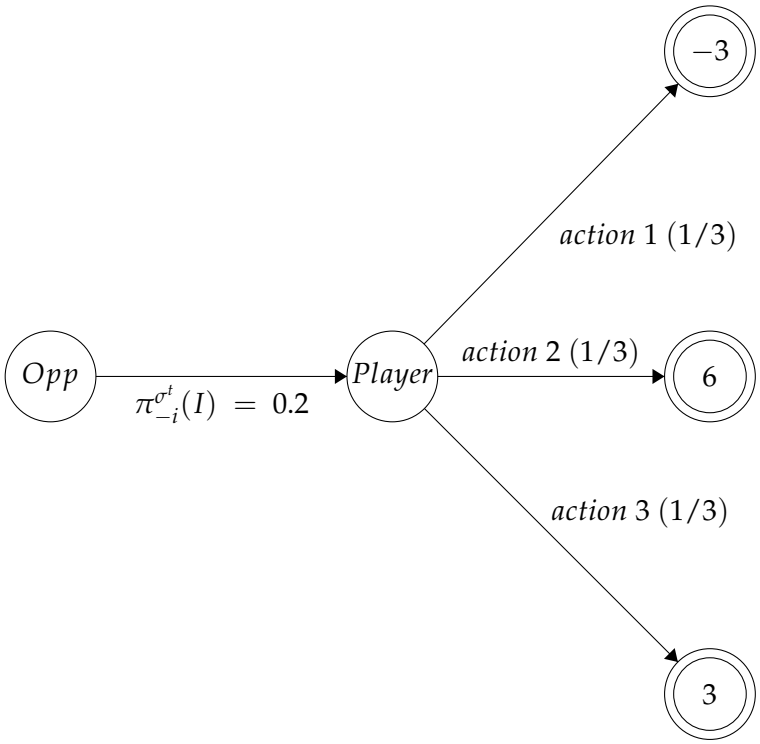

FIGURE 3.1: An example of the first iteration of MCCFR on one information set.

To further showcase how the MCCFR algorithm works, another example is given. For the next iteration where this information set is reached, the counterfactual regret of the previous iteration is used as follows:

Example 3.2.2. The next iteration where this information set is reached is visually represented in Figure 3.2. The information set and thus the possible actions are all the same as the previous example. However, the strategies of the players and the hand of the opponent are now different, resulting in new reach probabilities $\pi^{\sigma^t}(I)$ and expected values. Let us assume that now, the probability of reaching this information set is $\pi^{\sigma^t}_{-i}(I) = 0.5$ if the current player was actively playing to reach this information set. The expected value of the information sets which are reached by action 1, 2 and 3 are -1, 0 and 6, respectively. This makes the counterfactual value of the current information set equal to $0.5 \cdot (-1 \cdot 0 + 0 \cdot \frac{4}{5} + 6 \cdot \frac{1}{5}) = 0.6$. We can now calculate the instantaneous regret which becomes $0.5 \cdot -1 - 0.6 = -1.1$ for action 1, $0.5 \cdot 0 - 0.6 = -0.6$ for action 2 and $0.5 \cdot 6 - 0.6 = 2.4$ for action 3. These regret values accumulate to the counterfactual regret resulting in $-1 - 1.1 = -2.1$ for action 1, $0.8 - 0.6 = 0.2$ for action 2 and $0.2 + 2.4 = 2.6$ for action 3. Using equation 2.7, $\boldsymbol{\sigma}^{t+1}(I, a)$ is computed. For the next iteration, the probability of choosing action 1 is still 0 as it has negative counterfactual regret, the probability of choosing action 2 becomes $0.2/2.8 \approx 0.071$ and the probability of choosing action 3 becomes $2.6/2.8 \approx 0.93$. If we assume $\pi^{\sigma^t}_i(I) = 0.3$ this time, $\sum_{t=1}^{T} \pi^{\sigma^t}_i(I)\sigma^t(I)$ would now be $0 + 0 = 0$, $0.32 + 0.3 \cdot 0.071 \approx 0.34$ and $0.08 + 0.3 \cdot 0.93 \approx 0.36$ for actions 1, 2 and 3, respectively. If we now want to compute the average strategy, we can normalize the sum of weighted strategies using equation 2.6. This would result in a probability of 0, $\frac{0.34}{0.3+0.4} \approx 0.49$ and $\frac{0.36}{0.3+0.4} \approx 0.51$ for actions 1, 2 and 3, respectively.

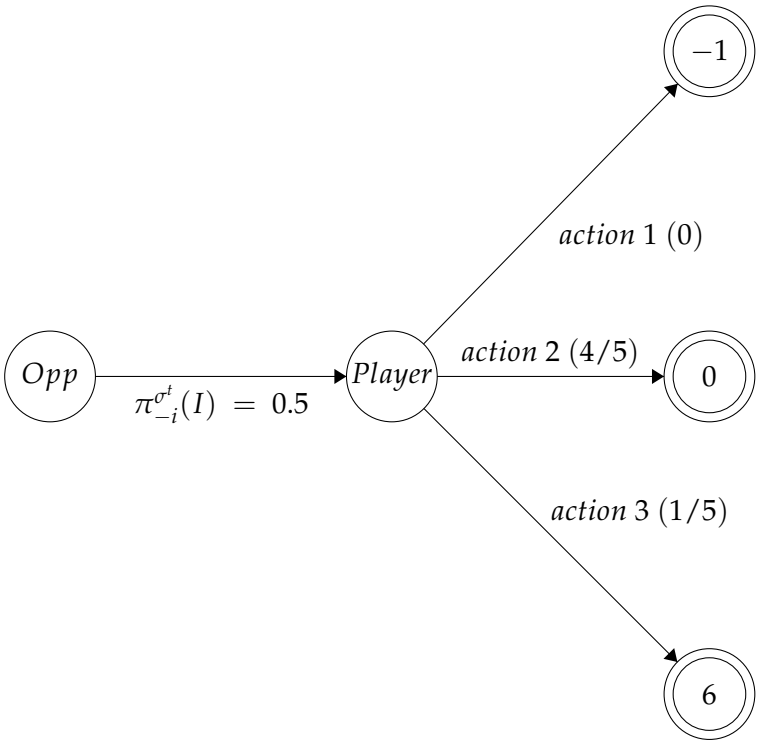

FIGURE 3.2: An example of a later iteration of MCCFR on one information set.

As we can see in these examples, the calculations are pretty straightforward. Each time the information set is reached, the computed strategies can fluctuate, based on the current strategy. However, over time, the average strategy will stabilize and approach a Nash equilibrium.

3.3 Abstraction

Abstraction is used to make Toepen tractable as it is too large in its original form.

3.3.1 Card Isomorphism

The first and simplest abstraction which will be used is called card isomorphism. This is a type of abstraction which bundles together game states which are only different in the suits of the cards. For example, a hand containing the 10 of hearts, 8 of hearts and 7 of clubs can be bundled with a hand containing 10 of spades, 8 of spades and 7 of diamonds. It is important to use a specific translation of the suits, which is then also used for the cards of the opponent. Otherwise, the suits would no longer match between a player and her opponent. This is a lossless abstraction as it does not lose any information from a strategic point of view. To avoid confusion, the suits are mapped to stings other than the actual suit names, namely: 'first', 'second', 'third' and 'fourth'.

To do this abstraction, each player needs a dictionary to translate the suits. This dictionary has to be dependent on the ranks of the cards in order to actually abstract the hands. Otherwise the suits would just rotate and there would be no reduction

Suits	Ranks	Handsize	Betting	Info sets
2	2	2	False	6
2	2	2	True	330
2	3	2	False	21
2	3	2	True	2,130
3	2	2	False	14
3	2	2	True	954
3	3	2	False	48
3	3	2	True	5,586
3	3	3	False	1,173
3	3	3	True	376,958

TABLE 3.2: The practical number of information sets after card isomorphism.

in the size of the game. The dictionary is made dependent on the ranks by sorting the hand from high to low, based on the ranks. The suit of the lowest card gets translated to 'first', the suit next lowest card that was not the first gets translated to 'second' and this is repeated until all cards in the hand have been translated. If there are suits which were not in the hand, they can be translated arbitrarily as long as the method is consistent. In this implementation the remaining suits are translated using an alphabetical order which is 'clubs'(first)-'diamonds'-'hearts'-'spades'(last). This ordering is also used in case of ties in the rankings. Two examples of a dictionary based on a hand are given bellow:

Example 3.3.1. hand: 8 of clubs, 9 of clubs, 10 of hearts, 10 of diamonds
dictionary: 'clubs': 'first', 'diamonds': 'second', 'hearts': 'third', 'spades': 'fourth'

Example 3.3.2. hand: ace of hearts, 10 of hearts, ace of diamonds, 8 of diamonds
dictionary: 'diamonds': 'first', 'hearts': 'second', 'clubs': 'third', 'spades': 'fourth'

Once the dictionary is made for a hand, it can be used to translate all game states which also contain cards of the opponent. Each player has her own dictionary as the translation has to be constructed at the beginning of the game when the only information available to each player is their own private hand. This reduces the number of information sets by the number of suits factorial at most. The practical number of information sets for some small versions of the game, after this abstraction is shown in Table 3.2.

3.3.2 Lossy Abstraction

Lossless abstraction is not enough to make the game of Toepen tractable for an algorithm like MCCFR. Therefore lossy information abstraction is needed. In the non-abstracted version of MCCFR, each game state is mapped to a key for the dictionary which contains the information sets. This key contains all the information available to the player which is the hand of the player and the history of the bets and the cards played. To use lossy information abstraction, a function is used which maps different hands and histories to the same key.

In Poker, abstraction is often done using potential-aware methods in combination with simple clustering algorithms. In potential-aware abstractions, information sets are bucketed based on the probability distribution over the strength in all future rounds of the game. Potential-aware abstraction makes sense in Poker, since

the strength of a hand in future rounds is determined by simple chance events with known probabilities. In Toepen, the strength of a hand in future rounds is determined by both chance events (the hand of the opponent) and the strategy of the opponent. This makes potential-aware abstraction a less suitable option for a game like Toepen. Therefore, we will rely on heuristic abstraction methods in this research. This means we will group information sets together based on abstraction rules which are predetermined.

A thing to note here is that MCCFR only requires to know the number of possible actions from a given game state. This means one could map all hands and histories with the same number of actions to the same key, resulting in just a few information sets. This would obviously result in a strategy that makes no sense and tries to balance the same probability distribution over groups of entirely different possible actions. However, it does mean that we can choose the abstraction rules freely, as long as the abstraction does not group information sets together with a different number of possible actions.

Abstraction rules can be applied to two different aspects of the game, namely: the hands and the history. Abstracting hands in Toepen can be done by grouping them based on some similarity metric. A straightforward metric would be the average rank of the hand. Since the rank of the cards determines the winner of the game, grouping the hands in this way would make sense. However, this metric misses many of the nuances of Toepen. Two hands with the same average rank can be very different in Toepen, based on the spread of the ranks of the cards. Having four mediocre cards is often worse than having two high and two low cards. This is due to the fact that a player can throw the away her low cards in the early rounds and use her high cards to win the later, more crucial rounds. A better metric is therefore the number of high cards in the hand, meaning the number of cards which is higher than the average rank of the deck. This metric is better at capturing the importance of the rank in a hand. Another important feature of a hand is the number of different suits in the hand. For the active player, having few different suits is often good, as that means there is less chance of losing the initiative to the opponent. For the reacting player, having many suits can be better as that increases the chance of gaining the initiative back at a later round.

An important note on the hand abstractions is that they abstract away information about the possible actions. The cards in a hand make up the actions from which a player has to choose. To the algorithm, the only inherent information about the actions are their indexes. Therefore, hand abstraction could cause different actions to be mapped to the same index. If those action are very different in outcome, it would be bad for the convergence of the algorithm. To combat this, the cards are ordered before they are given these indexes. This ordering is done by first grouping the suits in the hand together and then ordering those groups, based on the the rank of their lowest ranking cards. This should cause the same sort of actions to map to the same index. However, if the hand abstraction is too coarse, it will still map different sorts of actions to the same index. Consider the following example:

Example 3.3.3. Let us assume we use a hand abstraction which groups hands, based on their average rank. Now assume we have to play as a reacting player in the second round in a hand of Toepen. The active player played the 8 of clubs. We know how many actions are available, which we can assume is three. As the reacting player, we also have to play a card of the clubs suit if we have it. However, we have no information on the suits in our hand. If all three cards in our hand are cards of the clubs suit, we would want to play one of our high index cards as that means we

could win the round and gain the initiative. However, if our hand contains no cards of the clubs suit, throwing away the low index card is probably better, as that would preserve our higher ranking cards for the later rounds.

Since the MCCFR algorithm can not distinguish between these states, it will have to balance its strategy according to the probabilities of these hands occurring. This means that the strategy at this decision point has to be optimized for the player's own hand, rather than the hand or strategy of the opponent, which makes it less refined.

Abstracting the history is less intuitive than abstracting the hands. The history contains much more implicit information which is hard to abstract. In the non-abstracted game, the history is an ordered tuple, containing all the actions up until that point. This information can be used to infer which player is the active player, what the ante of the game is, which suits the opponent does not have, which specific cards are not in the opponents hand and at what points the opponent decided to bet. Information about the cards in the opponent's hand seems crucial to make the right decision at any point in the game. Therefore, abstracting the card playing actions could be too costly in terms of the information that is lost. However, a possibility here could be to group together histories with the same actions but different orders. This would still allow players to infer which specific cards are not in the opponents hand, but it would lose information on the suits which the opponent does not have as that can only be inferred based on the order of the actions.

Abstracting the betting actions in the history can be easier as the exact betting order has less influence on the way a game plays out. The betting order can be used to get a read on the strength of the opponent's hand, but if a player has a well balanced betting strategy, this should be hard to do. Whereas the order of the card playing actions gives first-order knowledge about the cards in the hand of the opponent, the betting order gives second-order knowledge through the strategy of the opponent. An abstraction rule for the history of the betting actions could therefore be to remove all the betting actions and replace them with the value of the ante. This would still keep the most crucial betting information, while leaving out the exact order of the betting.

To further explore the right abstraction rules for a game like Toepen in practice, a number of experiments will be conducted. These experiments will be outlined in Chapter 4.

Chapter 4

Experiments

This chapter contains a number of experiments which are used to explore the game of Toepen and ways to find optimal strategies for Toepen. The first section describes experiments which can determine the exploitability of a strategy, generated by MCCFR. This will be used on smaller versions of Toepen, to inspect how MCCFR handles certain aspects of the game. The second section describes experiments on the abstraction methods which are used to reduce the size of the game. Here we will look at the trade-off between certain fine-grained and coarser abstractions to determine which abstractions are right for a game like Toepen.

4.1 MCCFR

The first experiments are done to determine the strengths and weaknesses of MCCFR on Toepen in practice. The motivation for these experiments is to highlight the interaction between certain aspects of the game and the MCCFR algorithm. By looking at the exploitability curves across different game settings, we can see how the convergence differs for each of these settings. This can be used to infer how MCCFR would converge on the full game. To do this, we use MCCFR on the simpler versions of Toepen and track the exploitability over a number of iterations.

4.1.1 Methodology

For these experiments, we want to track the exploitability over a number of iterations. The exploitability of a strategy profile $\epsilon_\sigma = b_1(\sigma_2) + b_2(\sigma_1)$ is defined as the sum of the best response of player one to the strategy of player two and the best response of player two to the strategy of player one. If a strategy profile is a Nash equilibrium, the exploitability would be zero. Exploitability can therefore be seen as a metric of how close a strategy profile is to a Nash equilibrium. However, a problem with this metric is that it can be hard to determine the best response to a given strategy. In this research, we will approximate the best response by using the MCCFR algorithm exclusively for one player while keeping the strategy of the opponent unchanged. This means that the strategy for this player will become better and approach the best response to the strategy of the opponent over time. After the best response is computed, the expected payoff of this new strategy profile can be computed by summing over all terminal histories multiplied by the probability of them occurring, given this new strategy profile. We do this for both players and take the difference in expected payoff as an approximation for the exploitability. We call this procedure the evaluation algorithm.

Suits	Ranks	Hand size	Betting
2	3	2	False
2	3	2	True
3	2	2	False
3	2	2	True
3	3	2	False
3	3	2	True

TABLE 4.1: The game parameters for the MCCFR experiment

4.1.2 Parameter Setup

There are two different types of parameters for this experiment. The first parameter type is the game parameters. These are the parameters which determine the features of the game. The game parameters are the following:

- Suits, which is the number of different suits in the deck.
- Ranks, which is the number of different ranks in the deck.
- Hand size, which is the number of hands in each players hands. This also determines the number of rounds in a hand.
- Betting, which is a boolean and indicates whether the game contains betting or not.

The combinations of these parameters which are tested are given in Table 4.1.

The other type of parameters are the experiment parameters. These are the parameters which determine the scope of the experiment. These parameters are kept equal in order to capture the difference between the game parameter settings, which is what we are interested in. The experiment parameters are the following:

- Train iterations, which is the total number of iterations for which the MCCFR algorithm is run. For this experiment, this parameter is kept at 50,000.
- Evaluation intervals, which is the number of intervals at which an evaluation step is performed. For this experiment, this parameter is kept at 500.
- Evaluation iterations, which is the number of iterations for which the evaluation algorithm is run. For this experiment, this parameter is kept at 10,000.

Since the algorithm has inherent randomness, each experiment is repeated 15 times after which the average and standard deviation are taken.

4.1.3 Results

Figure 4.1 shows how the MCCFR algorithm converges for games without betting, a hand size of 2 and differing deck sizes. The x-axis shows the number of training iterations, while the y-axis shows the exploitability. A lower exploitability is better, meaning the strategy profile is closer to a Nash equilibrium. Since this game does not have betting, the ante of the game is always 1. This means the range of utilities is $\Delta_i = \max_{h \in Z} u_i(h) - \min_{h \in Z} u_i(h) = 1 + 1 = 2$ which also means the upper bound on the exploitability is 2.

Figure 4.2 shows how the MCCFR algorithm converges for games with betting, a hand size of 2 and differing deck sizes. The layout is similar to Figure 4.1. Since this is a game with hand size 2 and players can only bet before they play a card, the highest possible ante is 4. This means the range of utilities and the upper bound on the exploitability is $4 + 4 = 8$. The algorithm starts with random strategies meaning a uniform probability over each decision point in the game. We can see that this strategy leads to an exploitability of approximately 2 for all deck sizes in this setting.

FIGURE 4.1: The exploitability of strategy profiles for hand size 2 without betting. Lower is better.

4.1.4 Discussion

When looking at Figure 4.1, we can see that each deck size seems to converge to a certain exploitability and not improve that much from there. The order of the deck sizes in terms of exploitability from high to low is 2 ranks and 3 suits first, 3 ranks and 3 suits second and 3 ranks and 2 suits third.

It is important to consider that this is a setting with a hand size of 2. This means that there is not that much choice in terms of card playing actions. Both players can only make choices in the first round of play since the last round is predetermined by the remaining card in each of their hands. The rules of Toepen further restrict these actions, since a reacting player has to play the same suit if she has it.

The deck with 3 ranks and 2 suits converges the fastest and also reaches the lowest exploitability. A reason for this could be that the choices in this setting are more straightforward to compute and less dependent on the strategy of the opponent. Since there is only two different suits, many of the hands are predetermined to win or lose, regardless of the strategy. This is the case for any hand containing two connected cards in terms of rank of the same suit and a hand with the two highest or two lowest cards of different suits, which is already 12 of the 21 information sets.

FIGURE 4.2: The exploitability of strategy profiles for hand size 2 with betting. Lower is better.

In the other situations, the optimal decisions are either fully determined or of equal probability. For example, the reacting player should always play her lowest card if she cannot win the round, and her highest card if she can. The algorithm seems to pick up on this quickly. This means the Nash equilibrium does not have to be balanced based on the strategy of the opponent.

For the deck with 2 ranks and 3 suits, the strategies have to be more balanced with relation to the strategy of the opponent. This could due to the fact that, when the reacting player has two high ranking cards of different suits and the starting player plays a card of the third suit, the reactive player has to choose which one to throw away. If the strategy for this choice is not a uniform distribution, it means that the starting player has to balance how she plays a hand where she has a hand containing a a low and high card from two different suits. The less balanced these strategies are, the more they become exploitable. Balancing these strategies seems to be harder for the algorithm than approaching a Nash equilibrium in a game in which it does not have to do this.

An unexpected finding is that the curve of the exploitability of a deck with 3 ranks and 3 suits is lower than the curve of a deck with 2 ranks and 3 suits. This is not what one would expect, given that a larger deck has more information sets for which a strategy has to be computed. The result seems to suggest that this is not necessarily the case. A reason for this could be that, while the deck is larger, many of the new information sets are more straightforward than in the smaller deck. For example, most of the new information sets consist of hands where one card is ranked higher than the other. It could be that in these situations, throwing away the low card becomes a less balanced option, because the probability of that card still winning in the last round is lower than in a smaller deck. This would cause the initial convergence of the larger deck to be quicker.

When looking at Figure 4.2, we can see that the curves are in a different order than in Figure 4.2. A deck with 3 ranks and 2 suits is now the most exploitable, while the other two settings are both less exploitable and very similar to each other. This result highlights the difference which betting makes to the MCCFR algorithm. A deck with 3 ranks and 2 suits converged the fastest and was the least exploitable in the setting with no betting, but converged slower and was more exploitable in the setting with betting. One of the reasons for this is that betting causes strictly dominated actions to become viable again, as was shown in Example 3.1.3. Strictly dominated actions are very straightforward for MCCFR as these actions would get no regret, meaning the chance of exploring those branches diminishes quickly. Now that these are no longer strictly dominated actions in the game with betting, the algorithm has balance its resources on all branches of the tree, which means it will converge slower.

The other notable result of this experiment is that a deck with 2 ranks and 3 suits and a deck with 3 ranks and 3 suits are very similar in their convergence. The reason for this could be the same as for the setting with no betting, which was that, while there are more information sets, the information sets are more straightforward. This could also be true when betting is available since a good betting strategy involves estimating your hand strength in relation to the strategy of the opponent. In a small deck, the betting strategy has to be more refined as there are fewer possible hands for the opponent to consider. In a larger deck, the strength of a hand can be easier to determine since the probability of specific hands for the opponent is lower. It is also harder for the opponent to determine your specific hand. This means that the betting actions in a larger deck can become more straightforward and less dependent on the strategy of the opponent, which could cause the algorithm to converge faster.

The two main takeaways of these results are that betting can cause relatively simple games to become much more complicated in unpredictable manners and that a larger deck, does not necessarily mean that a strategy is harder to compute. The first takeaway is due to the fact that the deck with 3 ranks and 2 suits went from being the easiest the fastest to converge in the setting with no betting, to being the slowest in the setting with betting. This means that using a version of Toepen with no betting can be misleading when you want to learn a viable strategy to the full game of Toepen. It also means that using abstraction which abstracts away from betting could be unreliable. The second takeaway is due to the fact that the convergence in the larger deck setting was faster than the smaller deck settings. This is a promising result, as it means that learning the full game with a deck of 32 cards could be much easier than the size complexity, in terms of information sets, would suggest.

A shortcoming of these results is that they are tied strongly to the number of evaluation iterations. This parameter obviously has an impact on the results as more evaluation iterations results in more exploitability. However, it is also a parameter which has a big impact on computational resources. A reason why this is not a big problem is that we focus more on the convergence than the actual values of the exploitability. While the exploitability could be higher if we used a higher number of evaluation iterations, it would be higher for the entire curve, which means the structure of the curve would probably look similar.

4.2 Abstraction

The experiments in this section are done to determine which abstraction rules are most suitable for a game like Toepen. This is done by generating multiple strategy profiles using different rules and tracking their performance against a non-abstracted strategy profile. However, rather than just finding the optimal abstraction rules for Toepen, we will go over what aspects of the abstractions cause the results.

4.2.1 Methodology

For this experiment, we track the performance of strategy profiles, generated using abstraction rules, against strategy profiles, generated without any lossy abstraction, over a given number of training iterations. It is important to note that a strategy profile contains a strategy for both players in the game, meaning both the starting player and the reactive player. However, since we want to track the performance between different strategy profiles, it can be confusing to think of the strategy profile as containing a strategy for two different players. It can be more intuitive to view a strategy profile as a strategy of a single player for both starting positions in the game. The performance of a strategy profile is computed by simulating a number of games between the different strategy profiles. To simulate games, we take a strategy for each starting position from two different strategy profiles and use those strategies to sample an actual instance of a game of Toepen. Since there is an asymmetry between the starting positions in Toepen, each strategy profile is used an equal number of times per starting position. After this number of games is sampled, we take the average number of points to determine which strategy profile is better.

Since lossy abstraction is a trade-off between the rate of convergence and the quality of the solution, it is important to investigate the abstraction rules by tracking how they interact in combination with MCCFR. If we were to run the MCCFR algorithm for an infinite number of iterations, we would always expect a non-abstracted strategy profile to perform better than an abstracted strategy profile, as it has more information to act on and has theoretical guarantees of convergence. However, if we only ran the MCCFR algorithm for a very small number of iterations, we would expect the abstracted strategy profile to perform better, as it can reach more unique information sets per iteration. It is therefore less interesting to look at the performance of an abstracted strategy profile after a set number of iterations, and more interesting to look at the convergence of the performance as the number of iterations increases. This is done by training both the non-abstracted and abstracted strategy profiles from scratch and measuring their performance against each other at given intervals.

The following abstraction rules are tested:

- Naive abstraction, which abstracts away all information available to the player except for the number of possible actions at each information set. This means that for each player, all information sets with the same number of possible actions are grouped together.
- Simple history abstraction, which abstracts away all information about the history of the game. This means that all information sets where a player has the same hand and possible actions are grouped together.
- Simple hand abstraction, which abstracts away all information about the hands. This means that all information sets with the same history and possible actions are grouped together.

- Advanced hand abstraction, which abstracts hands down to a tuple containing the average rank of the hand, the number of cards which is higher than the average rank of the deck and the number of different suits in the hand.
- Advanced history abstraction, which just abstracts the betting history down to the the ante. This means that all histories with the same order of card playing actions and antes are grouped together.
- Advanced combined abstraction, which combines advanced hand and advanced history abstraction.

The simple and naive abstraction rules are chosen because their convergence can be used to see if our intuitions about abstraction are correct. They will most likely not be the best types of abstraction in terms of their score. The advanced abstraction rules are based on the observations made in subsection 3.3.2. They will more likely lead to actual abstractions which can be used in the full game of Toepen.

4.2.2 Parameter Setup

These experiments have the same parameters as the MCCFR experiments which can be found in subsection 4.1.2.

For this experiment, we will use the following game parameter settings:

- Suits: 3
- Ranks: 3
- Hand size: 3
- Betting: True

These game parameter settings are chosen because they allow for intricate dynamics in the game. Having a hand size of 3 makes the game considerably larger than a game with a hand size of 2, but this is still feasible due to the fact that the evaluation function is less costly than in the MCCFR experiments.

The experiment parameters are the following:

- Train iterations: 100,000
- Evaluation intervals: 1,000
- Evaluation iterations: 20,000

We will also compute the exploitability of the final non-abstracted strategy profile to see whether how far the algorithm has converged. This will only be done after all training iterations are done using 100,000 evaluation iterations.

As we mentioned previous experiment, both the algorithm and the evaluation method have inherent randomness. Therefore, each experiment is repeated 15 times, after which the average and standard deviation are taken.

4.2.3 Results

Figures 4.3 and 4.4 show how the abstracted strategy profiles perform against non-abstracted strategy profiles, trained for the same number of iterations. Figure 4.3 can be seen as a baseline, which compares the advanced abstraction method to the

simple abstraction rules which are very coarse and abstract away a lot of information. Figure 4.4 shows how the advanced abstraction methods perform separately, as well as combined. Figure 4.5 shows the number of information sets which are reached after a given number of training iterations. The size of the naive and simple history abstraction are 4 and 166 information sets, respectively. They are left out of the plot for visibility. The top line represents the total number of information sets in the non-abstracted game.

The exploitability of the final non-abstracted strategy profile after 100,000 evaluation iterations was approximately 0.914.

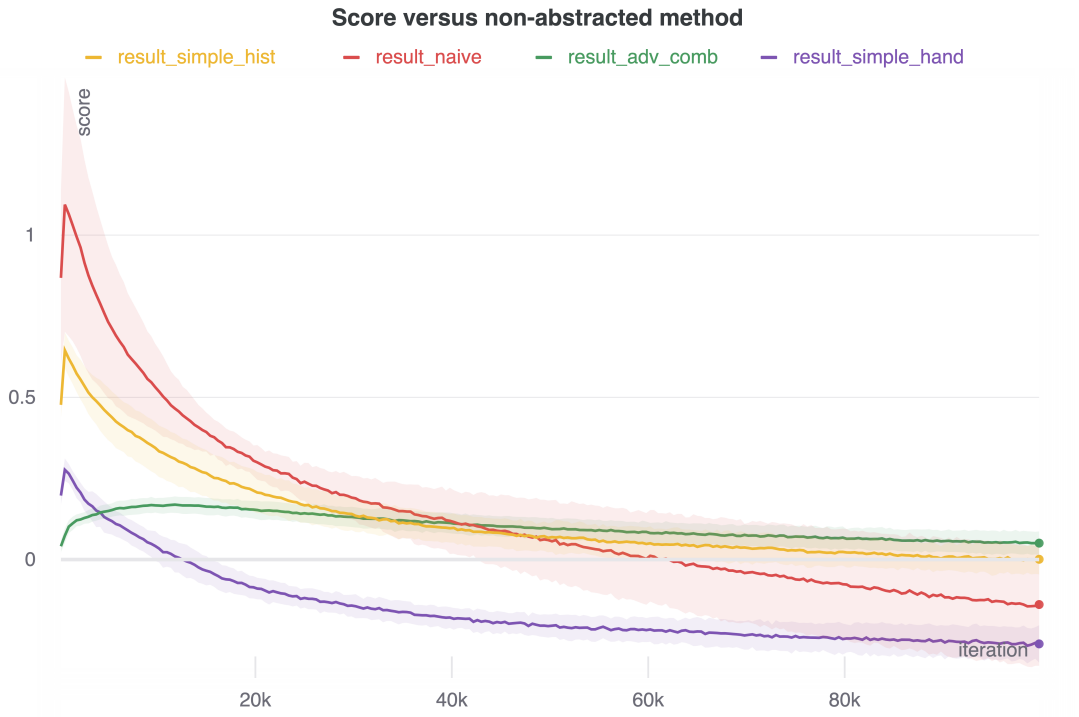

FIGURE 4.3: The score of strategy profiles generated using naive abstractions versus non-abstracted strategy profiles. Higher is better.

4.2.4 Discussion

If we look at Figure 4.3, we can see that the coarse abstractions actually perform very good in the beginning of the training process but quickly fall off. This expected, as the lower number of information sets means that the MCCFR algorithm can learn quicker. The simple hand abstraction falls off quickly, which also makes sense. Having no information about your own hand is obviously a big handicap which causes the strategy to fall off quickly. However, it is interesting that both the naive and the simple history abstraction perform better for a longer time. This is a clear example of the nonmonotonicity of abstractions in multiplayer games. The naive and simple history abstractions have 4 and 166 information sets, respectively. This means they are vastly coarser abstractions than the simple hand abstraction, but are still able to outperform it. In the case of the simple history abstraction, this is expected because the cards in the hand contain more crucial information than the history in isolation. However, for the naive abstraction this result is stranger. The naive abstraction should be a really bad abstraction. This abstraction has to balance the same

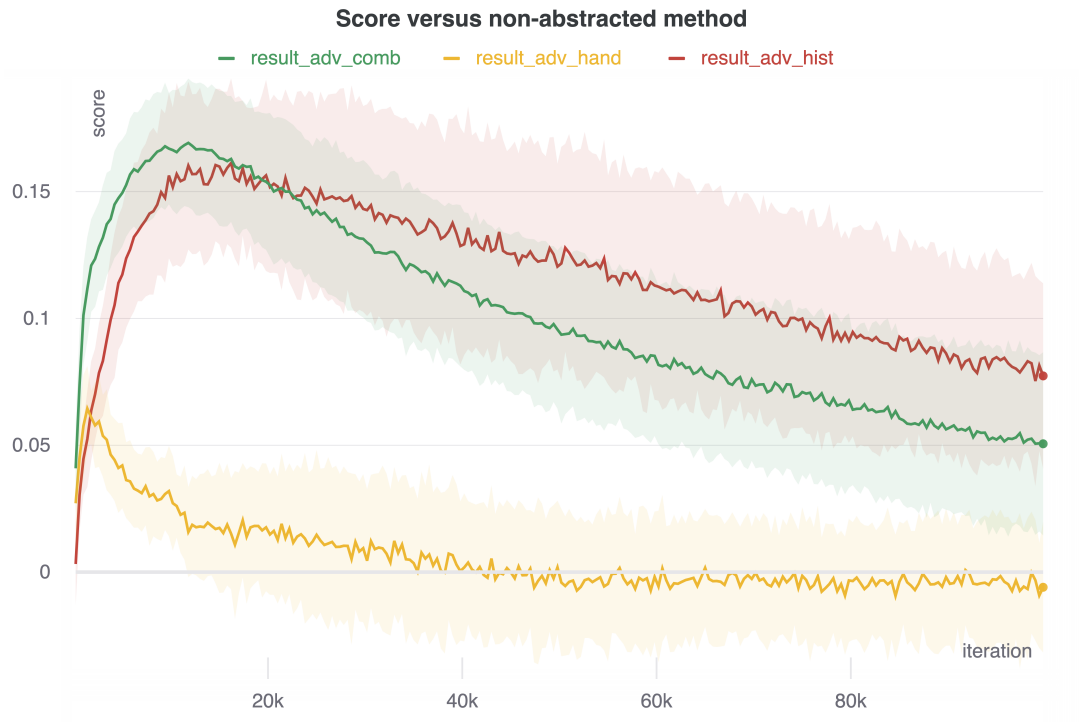

FIGURE 4.4: The score of strategy profiles generated using advanced abstractions versus non-abstracted strategy profiles. Higher is better.

probability distribution over card playing actions and betting actions, which obviously does not make sense. For the simple hand abstraction, this is not the case as it can use the history to infer what type of action it has to make. However, when we look at Figure 4.5, we can see that the simple hand abstraction converges to about 200,000 information sets. If these all contained high quality information, it might have been good to have this many information sets. In this case, it seems that this specific abstraction has the worst of both worlds. The history information is not enough to make well informed decisions, while the number of information sets is to large for MCCFR to converge properly in the given time.

In Figure 4.4 we can see the convergence of the advanced abstraction methods. The first thing we notice is that the advanced hand abstraction performs worse than the other methods. The fact that it performs worse than the combined advanced abstractions is unexpected. These results seem to suggest that abstracting away the betting order, somehow makes the hand abstraction perform better. A reason for this could be that advanced hand abstraction has about twice as many information sets. This means that the advanced hand abstraction method has to learn a strategy for many more information sets which could cause a slower convergence in the algorithm. Further, we can see that the advanced history abstraction performs better than the combined advanced abstractions after a small number of initial training iterations. This is what we would expect, given that the advanced history abstraction is only slightly bigger in terms of information sets, but contains far more fine-grained information about the hand.

From these results, we can infer that the advanced history abstraction, which groups together information sets with different betting orders, is a good abstraction to make. The number of information sets in this abstraction is reduced by half in

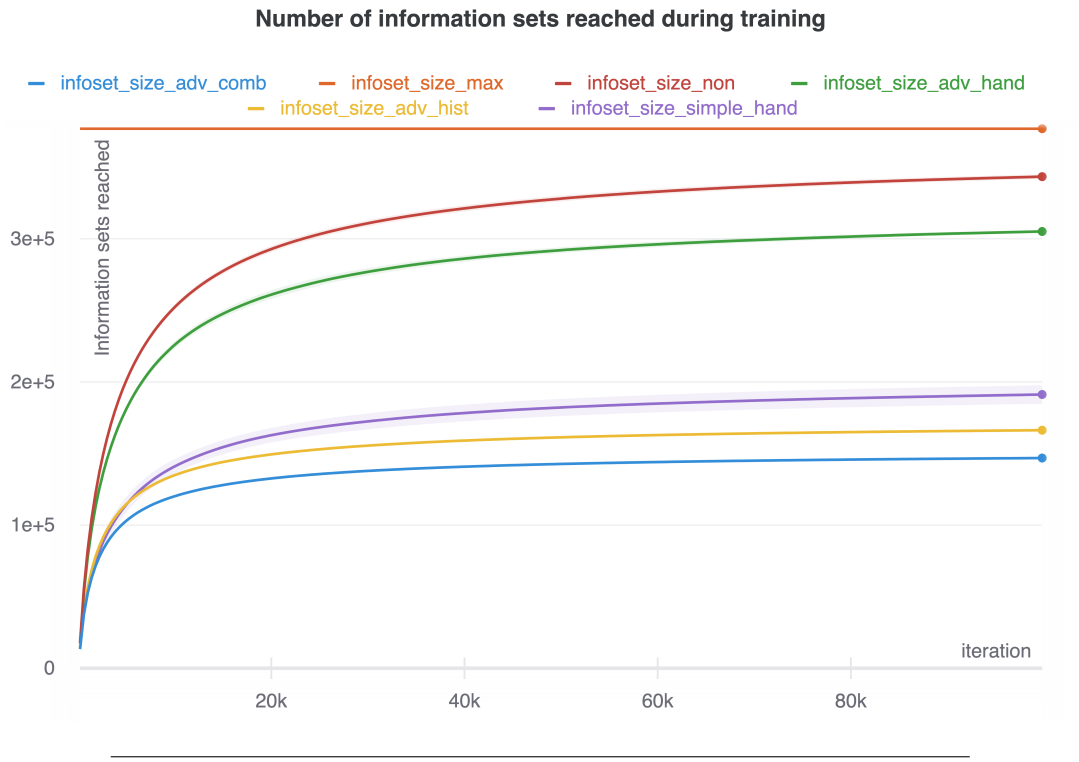

FIGURE 4.5: The number of information sets reached during training for different abstraction methods.

comparison to the non-abstracted version, while the performance over 100,000 iterations is the highest of all abstractions which were tested.

The general takeaways from these experiments are that history abstraction seems promising and hand abstraction seems bad terms of performance. Both in the simple and advanced settings, we can see that the history abstractions reduce the number of information sets the most, while still converging better than the hand abstractions. Hand abstraction also performs worse in isolation than the combination of hand an history abstraction.

In these experiments we could not approximate the exploitability in the same way as in the MCCFR experiments. This is the case because the abstraction rules are used within the MCCFR algorithm, which means the best response would be a best response for the abstracted game, rather than the original game. This could be circumvented by translating the abstraction back to the original game and then using the method described in the MCCFR experiments, but this was not done in this research.

A shortcoming of these experiments is that the number of training iterations is too low to make any theoretical claims on the convergence of the non-abstracted method. The total number of information sets for these game parameter settings is 376,958 where the starting player has 131,763 information sets and the second player has 245,195 information sets from which to act. If we use Equation 2.8, we can see that the total regret for the second player, using the non abstracted method, for any $\rho \in (0, 1]$ would be $R_i^T \leq (1 + \frac{\sqrt{2}}{\sqrt{\rho}}) * 245195 * 8 * \sqrt{4} * \sqrt{100000}$ with probability $1 - \rho$. Even if we chose a ρ approaching 1, this would still result in $R_i^T \leq 3.0 * 10^9$ which means the exploitability of this strategy profile is at least $2\frac{R_i^T}{T} \approx 2 * \frac{3.0*10^9}{100000} \approx 60000$. This means that from a theoretical point of view, we know the strategy profile is at

least a ϵ-Nash equilibrium with $\epsilon = 60000$. This value is obviously way to large to be useful. The range on the rewards $\Delta = 8$, meaning any value above 8 does not actually bound the exploitability. However, this theoretical bound is much worse than the actual result, which was an exploitability of approximately 0.914. This is still a big exploitability which means the non-abstracted strategy method was far from converged. If we look at Figure 4.5, we can also see that over the total number of training iterations, the non-abstracted method did not reach the total number of information sets.

Chapter 5

Conclusion

This chapter contains a summary of the findings of this thesis and some directions for future research in this area.

5.1 Thesis Summary

This thesis examined the card game Toepen from a game-theoretic perspective. This was done by using theoretical methods which were then tested in practice. First, we modelled Toepen as an extensive-form game. We then analysed the game in terms of complexity and explained how the game can be made smaller while still capturing the essence of the game. Next we explained what is needed in order to use the MCCFR algorithm in Toepen. Lastly, we went over how we can use both lossless and lossy abstraction in Toepen. Lossless abstraction can be done through card isomorphism, while lossy abstraction can be done through abstraction rules which need to be specific to Toepen.

Two experiments were done to investigate how these methods perform in practice. The first experiment was an experiment on the convergence of MCCFR in smaller versions of the game. This was done to investigate how specific aspects of Toepen influence the converge of MCCFR. The two main takeaways of this experiment were that betting can cause relatively simple games to become much more complicated in unpredictable manners and that a larger deck, does not necessarily mean that a strategy is harder to compute. The second takeaway is a promising result, as it means that learning the full game of Toepen with a deck of 32 cards could be much easier than the size complexity, in terms of information sets, would suggest. In the second experiment, we looked at the convergence of MCCFR, using different abstraction methods. This was done to investigate how abstraction works and what types of abstraction would be best for Toepen. The general takeaways from this experiment were that history abstraction seems promising and hand abstraction seems bad terms of performance.

The focus of this research was to find methods that can be used to create agents that can play a game like Toepen well. While we did not find a solution to do this for the full game of Toepen, we did put forth some findings which can be used as a foundation to investigate Toepen in the future.

5.2 Future Work

This research highlighted how Toepen can be understood through game theory and how to apply that understanding to generate strategies for the game. However, the full game of Toepen cannot yet be solved using just these methods. We will go over some of the directions for future research into Toepen.

One of the things we touched upon in Chapter 3 is how abstraction is done in Poker. This often includes clustering hands together based on their potential strength in future rounds using a clustering algorithm. This exact method is not as suitable for Toepen, since the strength of a hand is dependent on the strategy of the opponent. However, this general framework of using a metric and grouping using a simple clustering algorithm could still be strong in a game like Toepen. For example, one could model the potential strength in future rounds of the game, using a strategy generated by MCCFR.

Another clear direction is the use of real-time search. In this research, all computation is done beforehand to generate a strategy profile for the game. Many AI systems for both perfect- and imperfect-information games use some sort of real-time search to update a strategy during play. This reduces the computational cost dramatically.

Lastly, it would be interesting to look at other solution concepts than a Nash equilibrium. The strength of a Nash equilibrium is that it will never lose in expectation. However, this is very different from playing to win as often as possible. Playing to win requires the player to learn the weaknesses of the opponent during play, in order to exploit them. While this obviously leaves the player open to be exploited as well, it is interesting to explore how the agent can interact with the opponent.

Bibliography

Michael H. Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em Poker is solved. *Science*, 347(6218):145–149, 2015.

Noam Brown and Tuomas Sandholm. Regret transfer and parameter optimization. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit Poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer Poker. *Science*, 365(6456):885–890, 2019.

Murray Campbell, A. Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

Nicolo Cesa-Bianchi and Gábor Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

Sam Ganzfried and Tuomas Sandholm. Potential-aware imperfect-recall abstraction with Earth Mover's Distance in imperfect-information games. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

Andrew Gilpin and Tuomas Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em Poker. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007a.

Andrew Gilpin and Tuomas Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM (JACM)*, 54(5):25–es, 2007b.

Sergiu Hart and Andreu Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

Michael Bradley Johanson. *Robust strategies and counter-strategies: from superhuman to optimal play*. PhD thesis, University of Alberta, 2016.

Harold William Kuhn. A simplified two-person Poker. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 97–103. Princeton University Press, 1950.

Harold William Kuhn. Extensive Games and the Problem of Information. In Harold William Kuhn and Albert William Tucker, editors, *Contributions to the Theory of Games*, volume 2, pages 193–217. Princeton University Press, 1953.

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael H. Bowling. Monte carlo sampling for regret minimization in extensive games. In *Proceedings of the 22th conference on Advances in Neural Information Processing Systems*, pages 1078–1086, 2009.

James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*, page 200. MIT Press, Cambridge, Massachusetts, 1994.

Tuomas Sandholm. Abstraction for solving large incomplete-information games. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015a.

Tuomas Sandholm. Solving imperfect-information games. *Science*, 347(6218):122–123, 2015b.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Finnegan Southey, Michael H. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' Bluff: Opponent Modelling in Poker. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.

Kevin Waugh. Abstraction in large extensive games. Master's thesis, University of Alberta, 2009.

Kevin Waugh, David Schnizlein, Michael H. Bowling, and Duane Szafron. Abstraction pathologies in extensive games. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 781–788, 2009a.

Kevin Waugh, Martin Zinkevich, Michael Johanson, Morgan Kan, David Schnizlein, and Michael H. Bowling. A practical use of imperfect recall. In *Proceedings of the 8th Symposium on Abstraction, Reformulation and Approximation*, 2009b.

Philipp C Wichardt. Existence of Nash equilibria in finite extensive form games with imperfect recall: A counterexample. *Games and Economic Behavior*, 63(1):366–369, 2008.

Martin Zinkevich, Michael Johanson, Michael H. Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Proceedings of the 20th Conference on Advances in Neural Information Processing Systems*, pages 1729–1736. Curran Associates, Inc., 2008.