Czech Technical University in Prague

Faculty of Electrical Engineeing

Using Sequence-Form Double-Oracle Algorithm for Simplified Poker

Bachelor thesis

Martin Münch

Supervisor: Mgr. Branislav Bošansky, Ph.D.

May, 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Martin M ü n c h

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Použití iterativního double-oracle algoritmu pro řešení zjednodušené varianty pokru

Pokyny pro vypracování:

Extenzivní hry s nulovým součtem lze řešit pomocí iterativního algoritmu využívajícího kompaktní representaci strategií pomocí sekvenční formy, tzv. sequence-form double-oracle algoritmus. Hlavní myšlenkou je vytvoření omezené hry, kde hráči mohou volit pouze z některých akcí, vyřešení takto zjednodušené hry a následně její rozšíření o nové akce, které odpovídají nejlepší odpovědi na aktuální strategii. Tento algoritmus lze vnímat jako obecný postup a doménově nezávislé metody lze pro konkrétní hru nahradit specifičtějšími a tudíž rychlejšími algoritmy. Jednou z domén často používaných pro evaluaci algoritmů jsou i zjednodušené varianty hry Poker, ve kterých není rychlost doménově nezávislého double-oracle algoritmu ideální.
Cílem studenta je proto:
(1) přezkoumání možností využití kompaktnější reprezentace hry Pokru pro hledání nejlepších odpovědí hráčů,
(2) přezkoumání různých možností pro výběr akcí, které budou přidány do zjednodušené hry,
(3) experimentální porovnání navržených metod vůči doménově-nezávislému double-oracle algoritmu.

Seznam odborné literatury:
[1] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek: An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. Journal of Artificial Intelligence Research (JAIR) (to appear). 2014.
[2] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich: "Accelerating best response calculation in large extensive games." In IJCAI, vol. 11, pp. 258265. 2011.
[3] Yoav Shoham, and Kevin LeytonBrown: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2009.

Vedoucí bakalářské práce: Mgr. Branislav Bošanský, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Martin M ü n c h

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Using Sequence-Form Double-Oracle Algorithm for Simplified Poker

Guidelines:

Sequence-form double-oracle algorithm is an iterative algorithm for finding an exact Nash equilibrium in zero-sum extensive-form games. The main idea is to create a restricted game by limiting possible sequences of actions the players can play, solve this restricted game, and expand the restricted game by allowing new actions to be played. This generic framework can be tailored for specific games by replacing the domain-independent methods with algorithms that exploit domain knowledge. Poker is one of the established domains for comparing the performance of the algorithms for solving extensive-form games, however, the relative performance of sequence-form double-oracle algorithm is somewhat weak.

The goal of the student is therefore:
(1) explore possibilities for using more compact representation of a Poker game in order to speedup the algorithm for finding new actions to add,
(2) analyze different methods for selecting new actions to be added to the restricted game, and
(3) experimentally evaluate the impact of proposed methods.

Bibliography/Sources:

[1] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek: An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information. Journal of Artificial Intelligence Research (JAIR) (to appear). 2014.
[2] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich: "Accelerating best response calculation in large extensive games." In IJCAI, vol. 11, pp. 258265. 2011.
[3] Yoav Shoham, and Kevin LeytonBrown: Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2009.

Bachelor Project Supervisor: Mgr. Branislav Bošanský, Ph.D.

Valid until: the end of the summer semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 14, 2015

Abstract

Poker is one of the popular domains of game theory and it is used (in a simplified form) as a benchmark domain for comparing the algorithms for solving finite sequential games. In this thesis, we use sequence-form double-oracle algorithm, iterative approach for finding an exact Nash equilibirum for extensive-form zero-sum games. Intiution behind this agorithm is the following: firstly, it creates a game with restricted possible sequences of actions for players. Secondly, it solves this restricted game, and finally, it finds the best response against the solution of restricted game and expands the restricted game by the best-response sequences. In poker performance of domain-independent double oracle is not good. But the methods of the general algorithm can be replaced by domain-specific methods. Therefore, the goal of this thesis is (1) explore possible compact representation of Poker, (2) explore different methods for selecting actions to expand the restricted game, (3) experimentally compare proposed methods with general double oracle.

Abstrakt

Poker je jednou z populárních domén teorie her, která se často ve zjednodušené formě využívá pro porovnávání a evaluaci algoritmů. V této práci používáme sequence-form double-oracle algoritmus, což je obecný iterativní algoritmus hledající řešení pro extenzivní hry s nulovým součtem. Intuice algoritmu je následující: nejprve se vytvoří omezená hra, kde hráči mají omezené akce, které mohou zahrát. Najde se řešení této omezené hry a následně se rozšíří o nové akce odpovídající nejlepší možné strategii. Rychlost obecného double oraclu není pro Poker ideální. Metody obecného algoritmu jdou nahradit metodami využívající vlastnosti dané domény. Proto je cílem této práce (1) prozkoumat možnosti kompaktnější reprezentace hry Pokru, (2) prozkoumat různé možnosti pro výběr akcí, které rozšíří omezenou hru a (3) experimentálně porovnat navržené metody vzhledem k obecnému double oracle.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Acknowledgements

Firstly, I would like to thank my supervisor Mgr. Branislav Bošansky, Ph.D. for his encouragement, patience and guidance through the subject of this thesis.

Secondly, I warmly thank to my family and friends for their endless support.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Game theory is a part of applied mathematics, which aims to mathematically describe competetive scenarios and find solution for them. Solution is a strategy or behavior for player based on rational analysis of the scenario. These scenarios can be situations from the real world, including scheduling inspections of gas stations, checking passenger's tickets in subway, also some cooperative usage in multiagent conflicts as self-driving cars, multiple airplanes passing through same spot or drones patrolling area. Most of algorithms and methods for game theory come from testing, inspecting and experimenting of games like prisoner's dilema, tic-tac-toe, chess, go or poker.

In every competetive scenario or game, there are competitors and set of rules they must follow. Competitors are usually called players or agents and they are the one, who takes actions to progress the game and get payoff. Payoff is an expression of game outcome and player's interests. When we examine a game using game theory and its tools, we are trying to find optimal solution for an agent. Optimal solution or optimal strategy, is the solution maximizing payoff of the player. Nash equilibrium is the mostly used concept of optimal solution. Optimal solution according to Nash equilibrium for all players is a stable one, meaning no player wants to change his strategy, because it would lead to worse payoff. Nash equilibrium expects all players to act rationaly and if all players do, it guarantees to find optimal strategy for every player.

Games differ in many properties. One of them is, if they are one shot games or turn based, where players alternate between their actions. Poker is a sequential game,

where game alternate between card dealing or card revealing and betting round, where players take turns of betting. This property of a game leads to description using tree graph (called game tree), where action, as bet or dealing card, is an edge and state of game is a node. Some games have many possible actions in each node or many turns and that make their game tree very large.

Poker game usually starts with more players, for example 8. And as the game proceeds, players are eliminated, when they run out of chips. This eventually leads to two player poker game called Heads up which is the main interest of this thesis. We will describe, analyse and try to find optimal solution for Heads up poker. Therefore we will restrict our definitions for 2 player games.

Poker interests game theory almost since its founding (see for example [5], [7] or [10]), different aspects of poker were analyzed, at the early begining only by hand. Since then, modern hardware is involved with great computational power [3]. Now there are typically 2 options for evaluation of an extensive game strategy. First option is to gather strategies and make a tournament between them, for example Annual Computer Poker Competetion. The outcome of tournament between strategies is one ultimate winner, on the other hand, interpretation of the resuts is not that clear. Tournaments are the more popular choise between researchers and new aproximative algorithm were invented due to the tournaments. Second option is to challenge strategy against worst-case, where this choise suggests, strategy is robust to the choices of the other player. [4] In this thesis we will use double-oracle algorithm for finding Nash equilibrium for game of poker, which finds exact Nash exquilibrium, but for poker has not reached its potential.

Computing of Nash Equilibrium on game tree of texas hold'em, which has approximately $9.17 * 10^{17}$ game states, would require 10 years, if we could process 3 billion states per second [4]. This is the reason, why we need to explore more domain specific features of large games and exploit its characteristics to find Nash equilibrium on large games. In this paper we will explore Leduc Hold'em poker, which is very small, approximately 1900 game states.

Double-oracle algorithm is designed for two-player zero-sum extensive-form games with imperfect information, where poker belongs. The main idea of this algorithm is to firstly solve the restricted game, where players have only limited available sequences of actions. Then finding the best response against solution of the restricted game and adding this new sequences to the current solution [1].

1.1 Thesis Outline

Chapter 2 presents used terms of the game theory. Double-oracle algorithm with description of its main parts is described in chapter 3. Chapter 4 povides poker-specific best response. We presents our experiments and heuristics for the accelerated best response in chapter 5. Lastly, chapter 6 is summary of the thesis and discussion about possible extensions.

Chapter 2

Game Theory

Games are various situations, conflicts, diplomatic negotiations, etc. They vary in many properties, games can be turn-base or simultaneous, can end after one decision or take many turns to end, and so on. Each game has its own properties, which makes it unique, but all games has some properties in common.

Every game has player or players, whom are objects making decision based on some rule or pattern, players try to maximise their payoff. Decisions in games are called actions. Action can be move with chesspiece, betting in poker, placing your sign in tic-tac-toe. Payoff or gain for player from the game is a function, decribing preferences, interests, wins and losts. Payoff depends on played actions by the player and his opponents.

The poker is a zero-sum game with imperfect information. Zero-sum games are those, where gain for one player directly implies loss of the same value for the other player. Imperfect information means, player does not know all information about current state of the game. In our poker case, player does not know cards of the opponents.

In this chapter we provide a formal definition of these essential game theoretic concepts, that are necessary for following description of solution concepts and methods for finding these solution, as *player* or *action*, game representation as *normal-form* and *extensive-form* and finally with concepts as *nash equilibrium*, *best response*. All essential definitions and concepts are based on Multiagent Systems written by Shoham, Y. and Leyton-Brown, K. [9] and thesis of B. Bošanský [2]

2.1 Games

Mostly used form of representing game in publications about game theory is the normal form. Main reason for this popularity and usage is its straightforward notation and that all finite games can always be represented in the normal form. Normal form is a very general and does not use any structure of a game, not even alternation of players on the move.

2.1.1 Games in Normal Form

Normal form is usually used for one shot games, where players decides simultaneously, what action to play and the game terminates immediately after the selected actions are executed. Normal form uses matricies for representation of payoff function, where for each possible action of player is assigned his utility.

Definition 1. *Normal-form game for 2 players is a tuple (N, A, u), where:*

- *N is a set of two players, we use i to denote a player from N and $-i$ to denote his opponent*
- *$A = A_1 \times A_2$, where A_i is a finite set of actions available to player i*
- *$u = (u_1, u_2)$, where $u_i : A \to \mathbb{R}$ is a payoff function for player i*

This definition states, that normal-form game is a triplet, containing set of players, set of all possible game states and functions defining payoff (sometimes called profit or utility) for each player for each state.

Using payoff functions is the dominant approach in game theory to model preferences of players. There are no general requirements on payoff functions. But there is a large set of interesting games called zero-sum games, which are defined as [9]:

Definition 2. *A two-player normal-form game (N, A, u) is zero-sum if for each $a \in A$ applies $u_1(a) + u_2(a) = 0$.*

Example 1. *We have rock-paper-scissors game with two players, where each player can play rock, paper or scissors. Outcome depends on signs chosen by players. Rock beats scissors, scissors beats paper and paper beats rock.*

Let us call our two players Player 1 *and* Player 2*, and actions will be denoted R as rock, P as paper and S as scissors. Now we can describe rock-paper-scissors as a normal-form game:*

- $N = \{$ *Player 1, Player 2* $\}$
- $A = A_1 \times A_2 = \{R, P, S\} \times \{R, P, S\}$

		Player 2		
	(u_1, u_2)	R	P	S
Player 1	R	0, 0	-1, 1	1, -1
	P	1, -1	0, 0	-1, 1
	S	-1, 1	1, -1	0, 0

Table 2.1: Payoff for players in rock-paper-scissors game

Table with payoffs for players represents function $u = (u_1, u_2)$ from the definition of normal-form game. u_1 is gain for the first player and u_2 for the second player. We use payoff function with possible numbers 1, -1, 0 for denoting if player wins (1), loses (-1) or draws (0).

The normal-form representation is often not suitable for the describing sequential games. We should keep in mind, poker is a turn based game, which would be very difficult to describe with matrix or matricies. It can be done, but it would consume exponential amount of memory and time, than using extensive-form games. Therefore we use the extensive form, which is more suitable representation using tree structures to describe game.

2.1.2 Games in Extensive Form

As we stated before, extensive form is more suitable for poker and other sequential games, it uses tree graph to represent the game. Game tree represents actions as edges, game states as nodes and outcomes of the game as leaves.

Definition 3. *Extensive-form game for 2 player is a tuple (N, H, Z, A, χ, p, u, ρ, I), where [9]:*

- *N is a set of two players*
- *H denotes a finite set of nonterminal choice nodes*
- *Z is a finite set of terminal nodes, disjoint from H*
- *A is a set of actions*
- $\chi : H \to 2^A$ *is the action function, which assigns each choice node a set of possible actions*
- $p : H \to N \cup \{c\}$ *is the player function, which assigns to each nonterminal node a player or Nature, who choses an action at that node, c denotes a Nature player*
- $u = (u_1, u_2)$*, where* $u_i : Z \to \mathbb{R}$ *is a payoff function for player i for each terminal node*
- $\rho : H \times A \to H \cup Z$ *is the successor function, which maps a choice node and action to a new choice node or terminal node such that for all* $h_1, h_2 \in H$ *and* $a_1, a_2 \in A$*, if* $\rho(h_1, a_1) = \rho(h_2, a_2)$ *then* $h_1 = h_2$ *and* $a_1 = a_2$
- *I are information sets, which we will define and describe next*

Solution for the extensive form could be computed by converting the game into the normal form. The normal form is exponentialy larger representation of the extensive-form games, which would be ineffective. For solution of normal-form imperfect-information games is very often used compact representation called sequence form. Before definition of sequence form we need to define some more terms as imperfect information or strategy.

2.1.3 Perfect and Imperfect Information Games

Perfect-information games are those, where all players has every information about the game. For example in chess, both players see the whole board and know where every piece is and what actions the opponent plays. Poker, on the other hand,

has some information public and some hidden. Cards on the table and chips of the players is public information observable to every player. Cards in the player's hand are hidden, it is the private information.

Imperfect information causes uncertainity for a player, in which state he is in. In our case, game state in poker depends on the cards of the other player. One player only knows the probability for each card being in the opponent's hand. Player can be in one of the number of states as number of cards opponent may have.

The issue of imperfect information is addressed by information sets. Information set contains all possible nodes, player can be in. Intuition behind informaion sets is, that player knows which set he is in, but is not able to distinguish between nodes of this information set. He also knows, what actions can he play. [8]

I from Definition 3 is defined as [2]:

Definition 4. *$I = (I_1, ..., I_n)$, where $I_i = (I_{i,1}, ...I_{i,k_i})$ is a set of equivalence classes over nodes assigned to player i $\{h \in H : p(h) = i\}$. The nodes are indistinguishable to the player i and each node has the same set of possible actions $\forall h_1, h_2 \in I_{i,k_i} : \chi(h_1) = \chi(h_2)$.*

Example 2. *Figure 2.1 is an example of a game tree with the information sets for a nameless game, where players do not know which action opponent plays. There are 4 information sets, one **I2** for box player and two **I1**, **I3**, **I4** for circle player. Box player cannot distinguish between opponent's action **X**, **Y**. Circle player remembers if he played **X**, **Y** (he knows he is in **I3** or **I4**), but does not know whether opponent played **A** or **B**.*

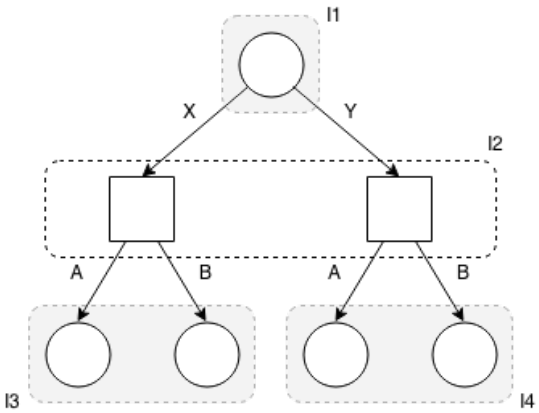

Figure 2.1: Information set example

2.2 Strategy

Strategy for the player is a map of an actions for a player to play in any given information set. This kind of strategy is called pure strategy [9].

Definition 5. *Let $G = (N, H, Z, A, \chi, p, u, \rho, I)$ be an imperfect-information extensive-form game. Then the pure strategy for player i consist of Cartesian product $\prod_{I_{i,j} \in I_i} \chi(I_{i,j})$*

Pure strategies for all players in the game are called pure strategy profile.

Since Nash equilibrium does not always exist in pure strategies for imperfect-information games, we need to define mixed strategies that does not assign single action for each game state, but assign set of actions with probabilities to each information set:

Definition 6. *Mixed strategy for extensive-form game [9]*

Let $G = (N, H, Z, A, \chi, p, u, \rho, I)$ be an imperfect-information extensive-form game and let Δ be set of all pure strategies. Then for any set X let $\mathcal{P}(X)$ be the set of all probability distributions over X. Then the set of mixed strategies for player i is $S_i = \mathcal{P}(\Delta)$

2.2.1 Best Response

Assuming we know strategy of our opponent, then we can find optimal strategy against his strategy. This optimal strategy is called best response.

Definition 7. *Let S_i be set of mixed strategies for player i, then the best response for player i, denoted as b_i, against opponent strategy σ_{-i} is [4]:*

$$b_i(\sigma_{-i}) = \arg\max_{\sigma'_i \in S_i} u_i(\sigma_{-i}, \sigma'_i) \tag{2.1}$$

2.3 Nash Equilibrium

For finding solutions to Poker game, we will use Nash equilibria. Optimal strategy profile according to Nash equilibrium is profile, where no player could get higher payoff by choosing different strategy. We will restrict our definition only for 2 player games:

Definition 8. *Nash equilibrium for 2 players [9]*

A strategy profile $\sigma = (s_1, s_2)$ is a Nash equilibrium if, for all players $i = \{1, 2\}$, s_i is a best response to s_{-i}

Example 3. *Example of Nash equilibria. Let G = (N, A, u) be normal-form game, where $N = \{P_1, P_2\}$, $A = \{X, Y\} \times \{X, Y\}$ and function $u = (u_1, u_2)$ is defined in the following matrix:*

(u_1, u_2)	X	Y
X	0, 0	-2, 2
Y	4, -4	0, 0

Nash equilibrium is strategy profile (Y, Y), both players plays Y. If player 1 deviate from the optimal strategy and playes X his payoff is -2. Similarly, player 2 would get -4 after playing X.

2.3.1 Nash Equilibrium in Extensive-Form Games

Using the mixed strategies for the extensive-form games, Nash equilibria is defined for the extensive form same way as for the normal form. Strategy from Nash equilibrium is sometimes perceived as not rational behavior, because it cannot exploit mistakes of opponent player. This is caused by assumption of Nash equilibrium, that both players are rational, therefore we should not be able to reach part of the game tree, where one player made mistake. [2]

When we consider only zero-sum games, if player i follows Nash equilibrium, he cannot achieve lower utility than value of the game $\mathcal{V}$, therefore player i gets at least game value, even if the opponent $-i$ is irrational.

2.4 Games in Sequence Form

As mentioned before, sequence form is useful mostly for representing extensive-form imperfect-information games in order to compute Nash equilibrium. Defined as follows [9]:

Definition 9. *Let G be an imperfect-information game, the sequence-form representation of G is a tuple (N, Σ, g, C), where:*

- *N is a set of two players*
- *$\Sigma = \Sigma_1 \times \Sigma_2$, where Σ_i is the set of possible sequences for player i*
- *$g = (g_1, g_2)$, where $g_i : \Sigma \to \mathbb{R}$ is the payoff function for player i*
- *$C = (C_1, C_2)$, where C_i is a set of linear constraints on the realization probabilities of player i*

2.4.1 Sequences

A sequence is an ordered list of actions, that player i has to take from the root to the given node h. Sequences are defined as : [9]

Definition 10. *A sequence of player i, denoted as σ_i, defined by a node $h \in H \cup Z$ fo the game tree, is the ordered set of player i's actions that lie on the path from the root to h. Let $\emptyset$ denote the sequence corresponding to the root. The set of all sequences for player i is denoted as Σ_i.*

2.4.2 Payoff Function

Payoff function extends the utility function to all nodes of the game tree. The payoff function for the sequence form is defined in [1] as follows:

Definition 11. *The payoff function g_i represents utility value of all nodes reachable by the pair of sequences σ.*

$$g_i(\sigma_i, \sigma_{-i}) = \sum_{h \in Z : \forall N, \sigma_j = seq_j(h)} u_i(h) \cdot \mathcal{C}(h)$$

2.4.3 Realization Plan

Realization plan is a probability for a given sequence seq_i of player i, that player will play this sequence, when opponent plays actions that reach information sets, where actions of the seq_i are defined.

2.4.4 Finding Solution for Sequence-Form Games

Finding solution for sequence form means finding a strategy profile, that satisfies conditions of given solution concept. We are using Nash equilibria as solution concept, where every player plays the best response against strategies of the opponent. We can compute Nash equilibrium for sequence form, and thus for extensive form, using a following linear program [9] of polynomial size in the size of the game tree:

Definition 12. *Linear program for computing Nash equilibrium of two-player zero-sum sequence-form game*

$$\begin{aligned} & \max_{r,v} v_{inf_{-i}(\emptyset)} \\ v_{inf_{-i}(\sigma_{-i})} - \sum_{I'_{-i} \in \mathcal{I}_{-i} : seq_{-i}(I'_{-i}) = \sigma_{-i})} & v_{I'_{-i}} \leq \sum_{\sigma_i \in \sum_i} g_i(\sigma_{-i}, \sigma_i) \cdot r_i(\sigma_i); \forall \sigma_{-i} \in \sum_{-i} \\ & r_i(\emptyset) = 1 \\ \sum_{\forall a \in A(I_i)} & r_i(\sigma_i a) = r_i(\sigma_i); \forall I_i \in \mathcal{I}_i, \sigma_i = seq_i(I_i) \\ & r_i(\sigma_i) \geq 0 \end{aligned} \tag{2.2}$$

Solution of this linear program is a realization plan for player i and expected value v for each information set of the opponent player $-i$. Realization plan is constrained by equation on the last three lines. Firstly, probability of playing empty sequence $\emptyset$ has to be 1. Secondly, probability of playing a sequence σ_i is the sum of sequences extended by one action. Thirdly, probability of playing a sequence is non-negative number.

Found realization plan is constrained by the best response of the opponent $-i$. This is ensured by second equation, where the opponent plays action that minimizes the expected utility $v_{I_{-i}}$ in each information set I_{-i}.

Chapter 3

Double Oracle for Computing Nash Equilibria

Main algorithm used for computing Nash equilibria in this thesis is the double-oracle algorithm. The algorithm is composed of two parts: the best response and the restricted tree. Finding the best response is faster than finding Nash equilibrium, and we usually do not have to compute Nash equilibrium for the complete game tree. Therefore double oracle firstly computes the best response for realization plan of the opponent, and then adds best-response sequences to the restricted game. The restricted game is smaller than complete game tree and it is easier to compute Nash equilibirum. Against Nash equilibrium realization plans we compute the best response and iterate again.

Firstly, we describe double oracle for the normal-form games and then for the extensive-form games. Although we use double oracle only for the extensive-form games, explanation of core mechanics of double oracle is easier on the normal form. Secondly, we describe the concept of the restricted game in double oracle for the extensive-form games and algorithm for computing the best response in the extensive-form games.

3.1 Double Oracle

Double oracle is an iterative algorithm [6] that computes an exact Nash equilibrium in the extensive-form zero-sum games with imperfect information. Double oracle

is our choice for two reasons. Firstly, double oracle has not performed well on poker, so there is space for improvement. Secondly, it uses best response, which is great for exploiting domain-specific knowledge and heuristics.

Double-oracle algorithm is based on idea of constraint/column generation [1], used for large scaled problems. Oracle algorithms exploit structure of the games in two ways. Firstly, it is usually not necessary to search through all possible strategies to find Nash equilibrium, but only small fraction of them. And secondly, it is computationaly easier to compute best response than to compute Nash equilibrium.

3.1.1 Double Oracle for Normal-Form Games

As stated before, normal-form games use matricies, where rows are pure strategies for one player and columns for the other one. Main loop of the double-oracle algorithm iterates through these steps until convergence (ilustrated in Figure 3.1) [1]

1. Restricting the game by restricting the set of pure strategies for each player
2. Solving the restricted game for both players
3. Compute a pure best response for each player against the strategy of the opponent from the previous step; best response strategy is not limited by the restricted game. This best response strategy is then added in the restricted game

Since the algorithm adds strategies for both players it is called double-oracle algorithm. It is called single-oracle algorithm if it adds strategy for only one player.

The algorithm ends, when neither player can improve his payoff from the game by adding a new sequences into the restricted game. This happens when both players play best responses against the strategies of the opponent.

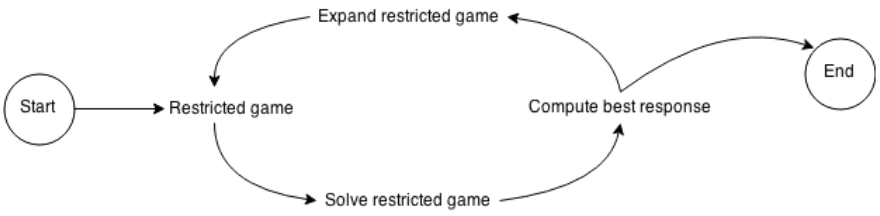

Figure 3.1: Double-oracle algorithm schema

3.1.2 Double Oracle for Extensive-Form Games

We use variant of double-oracle algorithm specifically for the extensive form. The most distinctive difference from double oracle for the normal form is in the approach for defining the restricted game. The restricted game for the extensive form is a subset of the complete game, defined by sequences. These sequences define actions available for players and reachable nodes and informations sets for each player.

3.2 Restricted Game

Restricted game for a normal-form game is a selection of rows and columns from matrix representing the whole game.

Restricted game for extensive-form games is defined by the set of available sequences for players, as we stated earlier [1]. This definition has two issues to resolve, both coming from the possibility of not having actions or strategy defined for every information set. Firstly, a strategy computed in the restricted game is not necessarily full strategy in the complete game, because it may not prescribe actions for information sets, that are not in the restricted game. Secondly, it may be impossible to play action from sequence allowed in the restricted game, because in the restricted game must be defined also compatible sequence of opponent, in order to reach a given state. Second issue is targeted by creating temporary leaves in restricted game from inner nodes of the complete game.

Restricted game is formally a subset of the original unrestricted game specified by the set of possible sequences. We use definition from [1]:

Definition 13. *Let* $G = (N, H, Z, A, p, u, \mathcal{C}, \mathcal{I})$ *be an unrestricted extensive-form game, then we define restricted game as* $G' = (N, H', Z', A', p, u', \mathcal{C}, \mathcal{I}')$*, where:*

- N*,* p *and* $\mathcal{C}$ *remain the same*
- $H' = \{h \in H : \forall i \in N, seq_i(h) \in \sum'\}$
- $A'(h) = \{a \in A(h) : ha \in H'\}, \forall h \in H'$
- $Z' = (Z \cap H') \cup \{h \in H' \setminus Z : A'(h) = \emptyset\}$
- $\mathcal{I}'_i = \{I_i \in \mathcal{I}_i : \exists h \in I_i, h \in H' \setminus Z'\}$

and utility function u':

$$u'(h) = \begin{cases} u(h), & h \in Z' \cap Z \\ u^*(h), & h \in Z' \backslash Z \end{cases}$$

where $u^*(h)$ is the outcome in the original game if the player i plays default strategy π_i^{DEF} and opponent plays best response to this default strategy.

Strategy for the restricted game is defined as [1]:

Definition 14. *Let r'_i be a mixed strategy represented as a realization plan of player i in the restricted game, then we define extended strategy $\bar{r}'_i$ as r'_i in nodes of restricted game and as default strategy π_i^{DEF} in other nodes:*

$$\bar{r}'_i(\sigma_i) = \begin{cases} r'_i(\sigma_i), & \sigma_i \in \sum'_i \\ r'_i(\sigma'_i) \cdot \pi_i^{DEF}(\sigma_i \backslash \sigma'_i), & \sigma_i \in \sum'_i; \sigma'_i = \arg\max_{\sigma''_i \in \sum'_i; \sigma''_i \sqsubseteq \sigma_i} |\sigma''_i| \end{cases}$$

If the sequence is defined in the realization plan of player i, then it is also defined in the extended strategy. If the sequence is not defined, then the extended strategy is composed of the longest defined strategy from realization plan and the default strategy.

Double-oracle algorithm each iteration solves sequence-form linear program for every player in order to compute a pair of strategies in the restricted game.

Example 4. *Figure 3.2 shows example of the restricted game, defined by sequences $\{YW, YZ\}$ for the first player and $\{A, B\}$ for the second player.*

3.2.1 Default Strategy

To resolve the first issue of restricted games (i.e. missing strategies for the complete game) the concept of default strategy is used every time the algorithm gets in the information set, without defined strategy. A default strategy can be predefined set of actions for every information set, but that would consume unnecessary amount of memory. It is more effective to use a rule-based default strategy instead. For example, the first action from possible set of actions for given information set.

Definition 15. *Default strategy for player i, denoted as π_i^{DEF}, is first action of deterministic method for generating ordered set of actions. [1]*

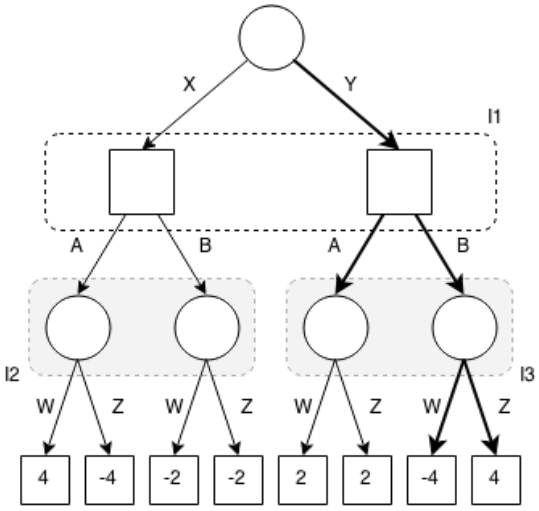

Figure 3.2: Restricted game example

3.2.2 Temporary Leaves

When there is no possible sequence for a given player from an information set in the restricted game, this set will become temporary leaf. For this leaf we define temporary utility value, which is needed for using this node as a leaf for linear program. The algorithm computes temporary utility value from expanding this node further, by playing default strategy for searching player and best response for the opponent. [1]

3.2.3 Solving Restricted Game

Restricted game is actually valid extensive-form game, therefore it can by solved by sequence-form linear program defined in Definition 12.

Double oracle computes a Nash equilibrium of the restricted game be solving a pair of linear programs. Optimal strategy for the restricted game can be translated as s strategy for the complete game, by using pure default strategy to complete restricted strategy, where restricted game is not defined. [1]

3.3 Best-Response Sequence Algorithm

In terms of game theory, best response is an optimal strategy for given player against known opponent's strategy. Best-response sequence algorithm is an algorithm

computing this optimal strategy. In the context of double oracle, the best-response sequence algorithm either generates new sequences for expanding restricted game, or proves there are no sequences to be added to the restricted game. [2]

Algorithm is a depth-first search on a complete game tree, where the strategy of the opponent is fixed in the realization plan $\bar{r}'_{-i}$. The algorithm differentiates between two types of game tree nodes: searching player node and opponent's or nature node. Where searching player is the player for whom the best response is computed and nature player is the environment. Pseudocodes for two node types are in Algorithm 1 and Algorithm 2 respectively. [1]

Other Player Nodes

Other player nodes are either opponent's (player $-i$) or Nature (chance nodes). In these nodes, the algorithm calculates the expected utility for a node according to the strategy of the player. When the player is the opponent, we use his fixed strategy given by extended realization plan $\bar{r}'_{-i}$ on the beginning of the best-response algorithm. When the player is Nature, we use stochastic environment ($\mathcal{C}$).

Throughout the algorithm, w denotes variable of probability from realization plan of the opponent and environment. Lastly, v_h denotes expected utility for the node.

When the algorithm is in the leaf, it returns probability of opponent reaching that leaf and stochastic environment probability. Otherwise, the algorithm performs depth-first search and returns sum of the results.

Algorithm 1 BRS in the nodes of the other players

Input:
i - searching player; h - current node; I_i^k - current information set;
$\bar{r}'_{-i}$ - opponent's strategy;

$w \leftarrow \bar{r}_{-i}(seq_{-i}(h)) \cdot C(h)$
if $h \in Z$ **then**
    return $u_i(h) \cdot w$
end if
$v^h \leftarrow 0$
for $a \in A(h)$ **do**
    $v^h \leftarrow v^h + BRS_i(ha)$
end for
return v^h

Searching Player Nodes

Different situation is in the nodes of the searching player. If the node is a leaf, it returns the utility multiplied by the probability of reaching this node. Then algorithm iterates over all actions for every game state in current information set, according to the reaching probability in descending order. Reaching probability is given by opponent's realization plan and Nature. The algorithm recursivelly calls BRS on $h'a$. Then, algorithm selects action with the highest possible utility, stores it and returns payoff for playing this action.

Algorithm 2 BRS in the nodes of the searching player

Input: Same as in Algorithm 1
if $h \in Z$ **then**
  return $u_i(h) \cdot \bar{r}_{-i}(seq_{-i}(h)) \cdot C(h)$
end if
if v^h is already calculated **then**
  return v^h
end if
$H' \leftarrow \{h'; h' \in I_i\}$
$v_a \leftarrow 0, \forall a \in A(h); maxAction \leftarrow \emptyset$
for $h' \in H'$ **do**
  for $a \in A(h')$ **do**
    $v_a^{h'} \leftarrow BRS_i(h'a)$
    $v_a \leftarrow v_a + v_a^{h'}$
  end for
  $maxAction \leftarrow \arg\max_{a \in A(h')} v_a$
end for
store $v_{maxAction}^{h'}$ as $v^{h'} \forall h' \in H'$
return $v_{maxAction}^{h}$

Chapter 4

Double-Oracle Algorithm for Poker

Poker is a sequential game, where the actions of players are observable, but the cards of players are not and they are dealt randomly. Cards in player's hands are the only imperfect information in the game of poker. We can exploit this special structure of poker game and modify double oracle to work more efficiently.

In this chapter we will describe improvements for the best response in the poker games.

4.1 Accelerated Best Response

We can use the specific structure of poker games to improve the best-response algorithm. To do that, we use a more compact representation termed public tree and accelerated best response as described in [4]. We firstly formally define the public tree and then describe the best-response algorithm that exploits this representation.

4.1.1 Public Tree

Public tree is a view of the game from the observer point of view. Nodes (called public states) are defined by all information both players have: by the table cards and played actions.

Definition 16. *We call a partition of the game states, $\mathcal{P}$, a public partition and $P \in \mathcal{P}$ a public state if*

- *no two game states in the same information set are in different public states (i.e., if information is public, all players know it)*
- *two game states in different public states have no descendants in the same public state (i.e., it forms a tree)*
- *no public state contains both terminal and non-terminal game states*

Public tree is a game tree with public states as nodes, instead of game states. Possible actions from public state are the same, because we basically traverse tree of information sets. In the original game tree, player's cards are resolved at the beginning of the game, where it causes more branches. In the public tree, we traverse a smaller tree, because we resolve outcomes for all possible cards in the leaves. Instead of traversing branches for all possible card combinations, public tree traverses only one branch.

Example 5. *Example of public tree.*

In Figure 4.1 (where all box nodes continues with 2 actions of first player: check, bet; as indicated at the middle box node) and Figure 4.2 we show the game tree and the public tree for first 3 actions of Leduc Hold'em poker. The deck of cards in Leduc Hold'em consists 3 different values, 2 cards for each value. First action is dealing a card to the first player, second action is dealing a card to the second player. Then the betting round occurs (box state). There are 9 game states in 3 different information sets in the game tree, which all continues the same until the end of game. But there is only single public state in the public tree. The information set ***I1*** *is represented by public state* ***A1*** *and sets* ***I2***, ***I3***, ***I4*** *are represented as* ***A2****. By this representation we do not have to compute 9 branches, that are the same, we just compute one branch and resolve player's card in the leaves.*

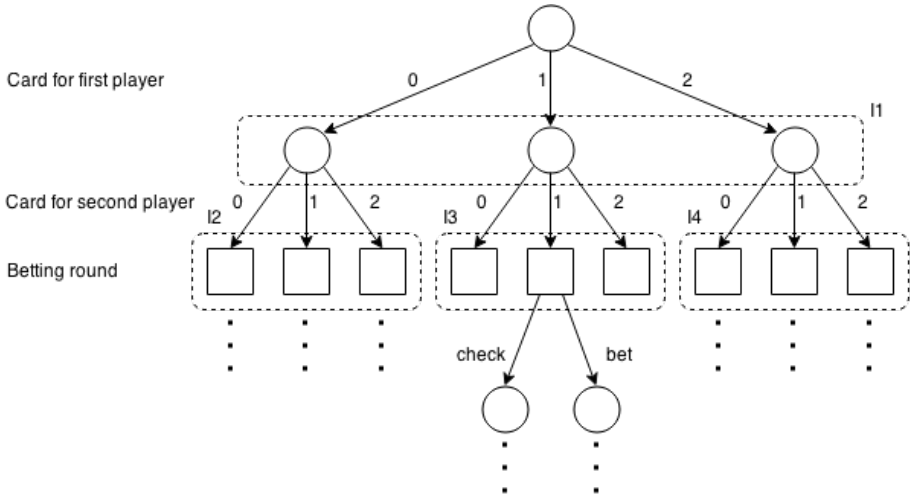

Figure 4.1: Leduc Hold'em game tree

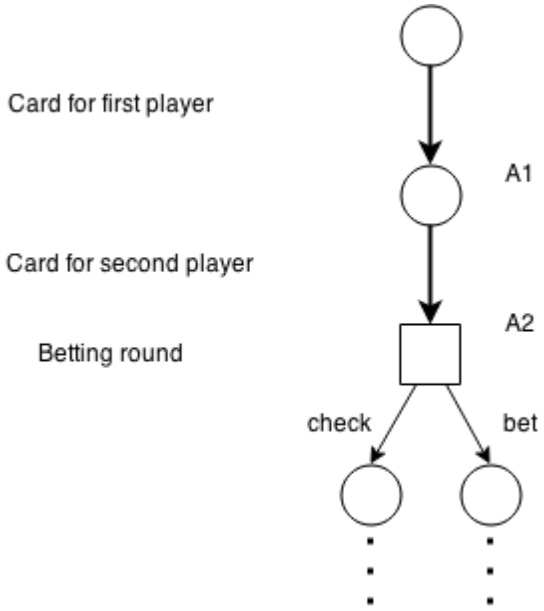

Figure 4.2: Leduc Hold'em public tree

4.1.2 Accelerated Best Response

Accelerated best response is a best response on a public tree. We divide the description of the accelerated best response into three parts: states of opponent and

nature player, states of searching player and leaves. Pseudocodes for these three parts are in Algorithm 3, Algorithm 4 and Algorithm 5 respectively.

Other Players Public States

In these states, accelerated best response returns the sum of the recursive calls results (lines 5-8). If the current player is the opponent, it also updates *probs*, when the probabilities for given seqeuences exists in realization plan of the opponent (line 2-3), otherwise they remains the same.

Algorithm 3 ABR in the states of other players

Input:
s - current public state
probs - vector of probabilities for opponent of reaching this s
RealizationPlan - the realization plan of the opponent
Output:
$\boldsymbol{v}$ - vector of outcomes for every combinations of cards of searching player

1: $\boldsymbol{v} \leftarrow \mathbf{0}$
2: **if** $\exists s \in RealizationPlan$ **then**
3: update *probs*
4: **end if**
5: **for** $a \in A(s)$ **do**
6: $\boldsymbol{v} \leftarrow \boldsymbol{v} + ABR(sa, probs)$
7: **end for**
8: **return** $\boldsymbol{v}$

Searching Player States

In the states of searching player, we are searching for the best action for each possible card of the player. Firstly, we recursively go through all possible actions for the current public state s (lines 3-5). Secondly, we go through all returned vectors and for every possible card of the player, we select the highest values (lines 6-8). If we order all returned vectors in a matrix, where vectors are columns, we return a vector, where values are highest numbers for each row separately. Returned value is a vector, where values represent best outcomes for all possible actions for all possible cards of the searching player.

Algorithm 4 ABR in the states of searching player

Input:
s - current public state
$probs$ - vector of probabilities for opponent of reaching this s
Output:
$\boldsymbol{v}$ - vector of the best outcomes for every combinations of cards of searching player

1: $\forall a \in A(ps) : \boldsymbol{v_a} \leftarrow \mathbf{0}$
2: $maxAction \leftarrow \emptyset$
3: **for** $a \in A(s)$ **do**
4: $\quad \boldsymbol{v_a} \leftarrow ABR(sa, probs)$
5: **end for**
6: **for** c *in* All possible combinations of cards **do**
7: $\quad \boldsymbol{v}_c \leftarrow \max \arg_c v_{a,c}$
8: **end for**
9: **return** $\boldsymbol{v}$

Efficient Terminal Node Evaluation

Next acceleration of calculation is by exploiting knowledge of poker in the leaves. Naive approach is to compute outcome for all possible combinations of cards in the terminal node. More efficient is to sort combinations of cards by rank and then we divide this sorted list into three parts: where a given hand of searching player is better, equal or worse [4].

In a terminal node, we know the public state and the vector of probabilities for every possible card combination, that opponent reaches this public state. We sort combinations of cards by rank for each player (lines 1-2), starting with the weakest and ending with the strongest combination. We keep two indicies for this sorted vector: first ($sameRank$) at the combination, that has equal rank as combination of searching player, second ($higherRank$) at the combination, that has higher rank than combination of searching player.

For a given combination of searching player, we compute sum of probability of the weaker combinations of the opponent and sum of probability of the stronger combinations of the opponent. Outcome for this combination of searching player is difference between those two sums, multiplied by half of the pot (lines 6-7). Then we update the indicies up by one and continues with better combination of searching player.

Algorithm 5 ABR in the leaves

Input:
s - current public state
probs - vector of probabilities for opponent of reaching this s
pot - sum of bets during the game
Output:
$\boldsymbol{v}$ - vector of outcomes for every combinations of cards of searching player

$searchingHandsSet \leftarrow$ sorted list of combination of cards of searching player
$oppHandsSet \leftarrow$ sorted list of combination of cards of opponent
$sameRank \leftarrow$ start of $oppHandsSet$
$higherRank \leftarrow$ first better combination of cards in $oppHandsSet$, than the weakest possible combination of cards
for $hand \in searchingHandsSet$ **do**
    $v_{hand} \leftarrow (\sum_{start}^{sameRank} oppHandsSet) - (\sum_{higherRank}^{end} oppHandsSet) oppHandSet$
    $v_{hand} \leftarrow v_{hand} \times (pot/2)$
    $sameRank \leftarrow sameRank + 1$
    $higherRank \leftarrow higherRank + 1$
end for
return $\boldsymbol{v}$

Chapter 5

Implementation

We have used GTLibrary (see Appendix A) with already implemented double-oracle algorithm as a baseline, where we implemented accelerated best response (described in previous chapter). Double oracle, as mentioned before, consists of three main components (best response, expanding restricted game, solving restricted game), which are implemented as three independent units. This separability allows us to replace original best-response algorithm with accelerated best response.

We have implemented public tree traversing and best response using Public tree. Subsequently we searched for patterns through best response outputs in Leduc Hold'em poker.

5.1 Implementation

Implementation of the public tree for Leduc Hold'em is straightforward. Actions of both players remains the same as in normal game tree. Main difference is in the dealing cards, where on the beginning we deal a default card to each player. And after first betting round we traverses through all possible table cards.

Leduc Hold'em poker usually has small number of different card values. Therefore we implemented two variants of terminal node evaluation, in order to compare their performance. First is efficient terminal node evaluation as described in previous chapter (see Algorithm 5). Second variant uses a property, that matrix of outcomes for every possible card combination is anti-symmetric. Matrix of outcomes is computed from point of view of the searching player, where the rows represents his cards and the

columns represents cards of opponents. For a given card combination, we can evaluate who wins. When we switch cards between the searching player and the opponent, the outcome is the same with the opposite sign. Note, that all experiments with heuristics were performed with the terminal node evaluation using matrix.

5.2 Experiments

Leduc Hold'em is a poker variant with a small game tree. Default deck consists of 6 card, 3 different values 2 cards each. Betting rounds remains the same as in Texas Hold'em. Firstly, both players are dealt one card, after first betting, table card is dealt. After second betting round comes end of the game with resolving winning player. There are only two possible combinations: High card or Pair. This poker is very easily scalable, to more possible card values and number of occurences of each value in deck, which we will use and explore more possible decks.

Important part of the poker game is the betting round, which occurs after dealing cards to the players or dealing cards on the table. Each player can play 5 different actions: check, bet, call, raise, fold. Check and bet can be played only if there was no bet in this round yet. Call, raise and fold are played only after bet or raise. Through played actions, players reveal their position and narrow possible cards they can have. But until the end of game, neither of the players can certainly know, the card of the opponent.

Firstly, we focus our experiments on comparison between original best response and accelerated best response. Secondly, we compare original best response with accelerated best response with heuristics. We are mainly comparing number of iterations, because as experiments in [1] shows, double oracle spends most of the time on solving linear programs during computation of solution for poker game. With less number of iterations, the algorithm will compute lower number of linear programs.

5.2.1 First Action Heuristics

Our hypothesis for Leduc Hold'em was, that player does not know, if his card is good or bad, until the table card is revealed (for example, card of the highest value is worse than pair of the lowest value cards). Therefore, it is better to be defensive in the first betting round and than be aggresive in the case that table card is in player's favor.

Players are not in the same position, first player takes action first and is in a worse position, because he must reveal his position and intentions before second player.

This heuristics is called in each searching player node in the accelerated best response. The method for the actual searching player adds the sequence to the set of sequences for the restricted game, if the conditions are met.

Heuristics for First Player

First player is playing check as his first action for every possible card he can have. In the second betting round, he bets if his card is same or above table card. We describe this more clearly in Algorithm 6.

Algorithm 6 First action heuristics for first player

```
Input: ps - current public state
1: if isPlayersFirstState(ps) then
2:     return sequence(ps) + "check"
3: else if isFirstStateOfSecondBettingRound(ps) then
4:     if playersCard ≥ tableCard then
5:         return sequence(ps) + "bet"
6:     end if
7: end if
```

Heuristics for Second Player

Second player's reaction on possible check of the first player is dependent on his card. He is defensive, checks, if value of his card is half or lower then maximal value, and aggressive, bets, in other cases. More clearly shown in Algorithm 7.

Algorithm 7 First action heuristics for second player

```
Input: ps - current public state
1: sequences ← ∅
2: if isPlayersFirstState(ps) AND lastAction = "check" then
3:     if HighestPossibleValue/2 < playersCard then
4:         sequences ← sequence(ps) + "bet"
5:     else if
6:         thensequences ← sequence(ps) + "check"
7:     end if
8: end if
9: return sequences
```

Experimental evaluation

Count of iterations computed through experiments on 9 different Leduc Hold'em deck settings for original best response, accelerated best response (abbreviated as ABR) without heuristics and ABR with heuristics are in the Table 5.1.

Deck setting		Number of iterations		
Number of card types	Cards of each type	Original best response	ABR without heuristics	ABR with heuristics
2	2	59	49	43
3	2	38	36	32
4	2	44	46	31
5	2	39	47	32
3	3	43	45	33
6	3	43	45	33
8	3	42	45	39
5	4	40	43	37
10	4	36	43	34

Table 5.1: Comparison of best responses with ABR with first round heuristics

Accelerated best response is traversing smaller tree than original best response, but that does not influence number of iterations needed for computing Nash equilibrium. Our assumption is, that we decrease number of iterations by adding more sequences to the restricted game. We compare our two variants of best response againt original best response (which will be our 100% baseline) in Table 5.2.

	ABR without heuristics	ABR with heuristics
Percentual difference	3.91%	18.23%
Standard deviation	3.64	3.98

Table 5.2: Percentual comparison of best response implementations

Our best response performs with heuristics about 18% better, than original. But it is suprising, that ABR without heuristics is similar, than original best response, because it typically adds more sequences to the restricted game.

5.2.2 Second Betting Round Heuristics

After implementing heuristics for first action of players, we focused on second betting round, the round, where the players know table card. Our hypothesis is, that best-response sequences from the second betting round are independent on seqeuences for first round, therefore we add into restricted game all possible sequences for first betting round concatenated with best-response sequences for second betting round. This approach is ilustrated in Algorithm 8. This method is called after selecting best actions in the node of searching node.

Algorithm 8 Second betting round heuristics

Input: ps - current public state
$firstBettingRoundSequences$
1: $sequences \leftarrow \emptyset$
2: **if** $isTerminalState(ps)$ AND $tableCard \neq NULL$ **then**
3: **for** $\forall seq \in firstBettingRoundSequences$ **do**
4: $sequences \leftarrow seq + sequenceSecondBettingRound(ps)'$
5: **end for**
6: **end if**
7: **return** $sequences$

Experimental evaluation

We have run experiments either with First round heuristics or with both heuristics presented in this section. Comparison of number of iteration is in following table Table 5.3.

Deck setting		Number of iterations	
Number of card types	Cards of each type	ABR with first round heuristics	ABR with both heuristics
2	2	43	37
3	2	32	28
4	2	31	34
5	2	32	28
3	3	33	36
6	3	33	34
8	3	39	33
5	4	37	30
10	4	34	32

Table 5.3: Comparison of effect of first round heuristics and both heuristics

Then we again compared ABR with original best response, but now running ABR with both heuristics. It performed about 25% better, than original best response (see Table 5.4).

	ABR with both heuristics
Percentual difference	25.52%
Standard deviation	3.23

Table 5.4: Percentual comparison of first round heuristics and both heuristics

Subsequently, we have compare time performance of the original best response with the accelerated best response, with following results (see Table 5.5):

Deck setting		Runtime [milliseconds]	
Number of card types	Cards of each type	Original best response	ABR with both heuristics
2	2	2423	3289
3	2	5129	6947
4	2	13738	11769
5	2	19353	17344
3	3	6568	7827
6	3	37379	32196
8	3	77815	62640
5	4	22795	19741
10	4	106503	102475
12	4	269136	202510

Table 5.5: Time comparison of best responses

The accelerated best response performed 16.78% better than the original best response. Generally, it has better time results in the bigger deck settings, whereas the original best response performes better the smaller deck settings.

5.2.3 Terminal Node Evaluation

Lastly, we have compared the implementations of the terminal node evaluation. We have compared both the number of iterations and runtime of ABR with both heuristics with efficient terminal node evaluation and with evaluation in the matrix (see Table 5.6).

Deck setting		Efficient terminal node evaluation performance		Matrix terminal node evaluation performance	
Number of card types	Cards of each type	Number of iterations	Runtime	Number of iterations	Runtime
2	2	37	3323	37	3162
3	2	28	5985	28	5857
4	2	31	11305	29	10684
5	2	37	17052	37	14163
3	3	36	6346	34	6063
6	3	30	23355	35	26531
8	3	34	50950	33	54944
5	4	30	18509	30	17705
10	4	33	85642	32	87818

Table 5.6: Comparison of terminal node evaluation

These two possibilities for terminal node evaluation performed equally good. The number of iterations is very similar, difference is about 0.34%. The changes are probably cause by different precission, efficient terminal node evaluation passes through the accelerated best response vector, while matrix evaluation passes matrix. Runtime is also very similar, difference is about 2.00%. This similarity is caused by the small size and simple structure of Leduc Hold'em. In larger poker variants (such as Rhode Island Hold'em or Texas Hold'em), the efficient terminal node evaluation should perform better, than matrix evaluation.

Chapter 6

Conclusion

Poker is a popular domain for game theory, due to its specific structure and size. It is a imperfect-information game with chance involved. We used the double-oracle algortihm for finding Nash equilibria, which iteratively solves the restricted game, computes best response against this solution and expands the restricted game by the best response. We have exploited the characteristics of poker games in the implementation of the accelerated best response, which traverses compact tree structure. We replaced implementation of accelerated best response instead of original best response in the double-oracle algorithm for extensive-form games. Our accelerated best response performs with equal results in the approximately same amount of iterations. Improvement of the accelerated best response is, it traverses smaller tree than original best response, which saves amount of memory needed by the double oracle.

Subsequently, we have examined results of accelerated best response for Leduc Hold'em poker and searched for patterns. Several patterns were identified and we exploited them by creating a domain specific variants of accelerated best response. Two heuristics rules were incorporated: first round heuristics and second betting round heuristics. First round heuristics is based on idea, that poker player is playing defensively before the table card is shown. Second betting round heuristics is based on hypothesis, that the betting rounds of poker are independent. Therefore the best-response sequence for second betting round should be the best-response sequence for all possible first round sequences. The improvement in the number of iterations of both these heuristics is approximately 25% from the original best-response sequence algorithm.

Lastly, we have compared two possible terminal node evaluation, which performed equally well. This equality is caused by size of Leduc Hold'em, which is a very small poker variant. The efficient terminal node evaluation will be faster, than matrix terminal node evaluation, when used on larger games. (For Texas Hold'em poker it is approximately 7.7 times faster, as stated in [4])

This improvement can have significant impact on bigger games, such as no-limit poker, which usually requires hundreds of iterations, which we can save by using accelerated best response with heuristics.

6.1 Further Work

This thesis provides implementations of accelerated best response and two heuristics for simplified poker with two main directions for future work: further improvements of the accelerated best response and implementation of larger variants of poker.

Improving the accelerated best response with other exploits of poker structure as Isomorphisms or Parallel Computation as described in [4]. Isomorpishms make some of the possible combinations of cards equally strong. Parallel computation uses structure of public tree, where a level of the public tree is chosen and branches from this level are solved in parallel.

Secondly, implementation of larger variants of poker (for instance Rhode Island), with more specific heuristics and effective usage of memory. Usage of accelerated best response should help to compute games, that were too large for domain-independent double-oracle algorithm. These larger variant would also benefit from the efficient terminal node evaluation.

Appendix A

Content of CD

Content of CD attached to this thesis:

- `doc/thesis.pdf` Electronic version of this thesis
- `src/gtlibrary/` GTLibrary used as baseline for this thesis. Important classes:
  - `PublicInformationSet.java` contains implementation of public tree
  - `PublicTreeBR.java` contains implementation of accelerated best response on public tree
  - `WinLoseProbabilities.java` contains matrix terminal node evaluation
  - `TerminalPublicTreeBR.java` contains implementation of ABR using efficient terminal node evaluation
  - `TerminalWinLoseProbabilities.java` contains matrix terminal node evaluation
- `src/experiments.sh` bash script running the experiments with same deck settings on original best response and accelerated best response
- `src/analyse.sh` divide results of experiments.sh into different file for better readability

Appendix B

GTLibrary

GTLibrary is a baseline for our implementation and experiments. Important parts for this thesis:

- Double oracle implementation is in the package `algorithms.sequenceform.doubleoracle`
- Generic poker, implementation of Leduc Hold'em with customizable deck settings is in the package `domain.poker.generic`
- Accelerated best response is in the package `algorithms.sequenceform.doubleoracle.pokerdomained`

Bibliography

[1] Branislav Bošanský, Christopher Kiekintveld, Viliam Lisý, and Michal Pěchouček. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *J. Artif. Int. Res.*, 51(1):829–866, September 2014.

[2] Branislav Bošanský. *Iterative Algorithms for Solving Finite Sequential Zero-Sum Games*. PhD thesis, Czech Technical University in Prague, 2014.

[3] A. Gilpin and T. Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM, Vol. 54, No. 5, Article 25*, 2007.

[4] Michael Johanson, Kevin Waugh, Michael Bowling, and Martin Zinkevich. Accelerating best response calculation in large extensive games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 258–265. AAAI Press, 2011.

[5] H. W. Kuhn. A simplified two-person poker. *Annals of Mathematics Studies, 24, vol. 1*, 1950.

[6] H. B. McMahan, G. J. Gordon, and A. Blum. Planning in the presence of cost functions controlled by an adversary. *Proceedings of the International Conference on Machine Learning*, 2003.

[7] J. F. Nash and L. S. Shapley. A simple three-person poker game. *Annals of Mathematics Studies, 24, vol. 1*, 1950.

[8] Noam Nisan. *Algorithmic Game Theory*. Cambridge: Cambridge University Press, 2007.

[9] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.

[10] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.