

Bachelor Project



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Cybernetics**

No-regret Learning in Generalized Normal-Form Games with Sequential Strategies

Prokop Šilhavý

**Supervisor: Mgr. Branislav Božanský, Ph.D.
Field of study: Open Informatics
Subfield: Computer and Informatic Science
May 2017**

Acknowledgements

I would like to thank my supervisor, Mgr. Branislav Bošanský, Ph.D., for his patient guidance, helpful advises, and constructive critique.

Furthermore, I would like to thank my family for the love, support during the study, and for the permanently full fridge. And special thanks go to my sister Terezie for many typographic and linguistic suggestions.

Finally, my thanks belong also to VO MetaCentrum, which provides distributed computing infrastructure, and which enables us to run all the experiments.

Declaration

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

.....
signature

Prague, date May 25, 2017

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.

.....
podpis autora práce

V Praze dne 25. května 2017

Abstract

This work formalizes generalized normal-form games with sequential strategies and finds Nash equilibria in them. This game model is the same as normal-form game with sequential strategies (NFGSS); the only difference is that there is no restriction on the utility. First, we have evaluated the solution of NFGSS in generalized NFGSS. Then we have used the transformation of generalised NFGSS to normal form (NFG) and solving it by standard methods, and finally, we adapted Monte Carlo Counterfactual regret minimization (MCCFR) algorithm for generalized NFGSS. We have tested these methods in three game domains: Transit game (TG), Border protection game (BPG), and Ticket inspection game (TIG).

The MCCFR algorithm converges to Nash equilibrium in BPG and TIG, and for TG it gives as a better estimate as the approximation by NFGSS. The scalability of MCCFR is bad. However, it is faster as the standard methods for NFG. The results show that MCCFR algorithm gives the best result from these three approaches either as an exact solver in TIG and BPG or as a heuristic in TG.

Keywords: Game Theory, Normal form, Sequential strategies, NFGSS, generalized NFGSS, CFR, MCCFR

Supervisor: Mgr. Branislav Bošanský, Ph.D.

Abstrakt

Práce formalizuje zobecněné hry v normální formě se sekvenčními strategiemi a představuje koncepty pro hledání Nashova ekvilibría v těchto hrách. Tento herní model je identický s hrami v normální formě se sekvenčními strategiemi (NFGSS), ale nemá žádná omezení na výplatní funkci. Nejprve jsme použili Nashova ekvilibría z NFGSS jako odhad řešení zobecněného NFGSS. Dále jsme hledali řešení pomocí převedení NFGSS do normální formy a následně použili standardní metody. Jako poslední přístup jsme adaptovali Monte Carlo Counterfactual regret minimization (MCCFR) přímo na zobecněné NFGSS. Všechny tyto metody jsme testovali na třech doménách: Transit game (TG), Border protection game (BPG) a Ticket inspection game (TIG).

MCCFR algoritmus konverguje k Nashovu ekvilibriu v BPG a v TIG. Pro TG nám dává lepší odhad řešení než zjednodušení zobecněného NFGSS na standardní NFGSS. Škálovatelnost tohoto algoritmu není vysoká, nicméně dokážeme s jeho pomocí vyřešit větší hry než za využití běžných metod na řešení NFG. Výsledky ukazují, že MCCFR algoritmus má nejlepší výsledky ze všech tří zkoumaných přístupů, a to nejen jako algoritmus na přesné řešení NFGSS pro TIG a BPG, ale i jako heuristika pro TG.

Klíčová slova: Teorie her, normální forma, sekvenční strategie, NFGSS, zobecněné NFGSS, CFR, MCCFR

Překlad názvu: Učení v zobecněných normálních hrách se sekvenčními strategiemi

Contents

1 Introduction	1	5.2 MCCFR algorithm application for NFGSS	24
1.1 Overview	2	5.2.1 MCCFR Algorithm for Modified NFGSS	26
2 Background	3	6 Experimental Evaluation	27
2.1 Normal-Form Games	3	6.1 Used Software and Resources	27
2.1.1 Definition	3	6.2 Game Domains	28
2.1.2 Strategy	4	6.2.1 Transit Game	29
2.2 Extensive-Form Games	4	6.2.2 Border Protection Game	29
2.2.1 Perfect-Information Games	4	6.2.3 Ticket Inspection Game	30
2.2.2 Strategies in Extensive-Form Games	5	6.3 Nash Equilibrium in NFGSS Evaluated on Generalized NFGSS	31
2.2.3 Imperfect-Information Games	6	6.4 LP Solver	33
2.2.4 Alternative Definition	7	6.5 Analysis of MCCFR for Generalized NFGSS	34
2.2.5 Perfect Recall	8	6.5.1 Border Protection Game	35
3 Solving Games	9	6.5.2 Ticket inspection game	38
3.1 Solution Concepts	9	6.5.3 Transit game	39
3.1.1 Best Response	9	7 Conclusions	45
3.1.2 Nash Equilibrium	10	7.1 Future work	46
3.1.3 ϵ -Nash Equilibrium	10	A Bibliography	47
3.2 Finding Nash Equilibria in Normal-Form Games	11	B CD Content	49
3.3 Finding Nash Equilibria in Extensive-Form Games	12	Running the Implementations	49
3.3.1 Counterfactual Regret Minimization	12	C Project Specification	51
3.3.2 MCCFR	13		
4 NFGSS	15		
4.1 Definition	15		
4.2 Conversion of NFGSS to Normal-Form Game	16		
4.3 Conversion of NFGSS to Extensive-Form Game	17		
4.4 Generalized NFGSS	18		
4.4.1 Definition	18		
4.4.2 Conversion of Generalized NFGSS to the Normal Form	19		
4.4.3 Conversion of Generalized NFGSS to the Extensive Form	20		
4.5 Solving NFGSS	20		
4.6 Best Response in NFGSS	20		
4.6.1 LP Finding Nash Equilibria in NFGSS	21		
4.6.2 CFR Algorithm on NFGSS	21		
5 Solving Generalized NFGSS	23		
5.1 Best Response in Modified NFGSS	23		

Figures

<p>2.1 Payoff matrix of rock, paper, scissors game (<i>Figure 3.7 in [1]</i>) . . . 3</p> <p>2.2 The perfect information game tree of the Sharing game (<i>Figure 5.1 in [1]</i>) 5</p> <p>2.3 An imperfect information game tree. Dashed line connects nodes in the same information set. (<i>Figure 5.10 in [1]</i>) 6</p> <p>2.4 An imperfect recall, imperfect information game tree. (<i>Figure 5.12 in [1]</i>) 7</p> <p>3.1 Payoff matrix of Companies competition game 10</p> <p>4.1 Example of NFGSS. At the top there are MDPs of both players and predefined utility is below them. . . 16</p> <p>4.2 The imperfect-recall, imperfect-information, extensive-form representation of the game from Figure 4.1. 18</p> <p>6.1 Scheme of a graph of Transit game. 29</p> <p>6.2 Graphs of Border protection game with width 3. 30</p> <p>6.3 MDP of passenger in Ticket inspection game. 31</p> <p>6.4 Sizes of utility matrices and computation times for Transit game (a, b), Border protection game (c, d), and Ticket inspection game (e, f). In the time analysis chart, the labels mean: ‘strategies’ - the part of getting all strategies, ‘utility’ - the part of getting utility, and ‘LP’ - the part of solving LP. 34</p> <p>6.5 Convergence chart of the BPG with the weakly connected graph, width 3, and 4 steps. 35</p> <p>6.6 Convergence chart of the BPG with the almost fully connected graph, width 3, and 4 steps. 35</p>	<p>6.7 Convergence chart of the BPG with the fully connected graph, width 3, and 4 steps. 36</p> <p>6.8 Scalability of MCCFR in Border protection game with weakly connected graph 36</p> <p>6.9 Scalability of MCCFR in Border protection game with almost fully connected graph 37</p> <p>6.10 Scalability of MCCFR in Border protection game with fully connected graph 37</p> <p>6.11 Convergence chart of the TIG with two stations, two trains, and approximative 2000 passengers. . . . 38</p> <p>6.12 Scalability of MCCFR in Ticket inspection game 39</p> <p>6.13 Convergence chart of the TG 2 2 40</p> <p>6.14 Convergence chart of the TG 3 2 40</p> <p>6.15 Convergence chart of the TG 3 2 2 40</p> <p>6.16 Convergence chart of the TG 3 2 2 in the detail. 41</p> <p>6.17 Convergence chart of the TG 4 3 2 41</p> <p>6.18 Convergence chart of the TG 4 3 3 42</p> <p>6.19 An example of a situation in Transit game, where utility gained by action a depends on the previous history. 43</p> <p>6.20 Scalability of MCCFR in Transit game 43</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Tables

4.1 An example of the definition of nonlinear utility in the game described in Figure 4.1. The utility is defined for the whole sequences in MDPs.	19
6.1 Table of versions of the software used in the experiments.	28
6.2 Table of the clusters used in the experiments. ¹	28
6.3 Nash equilibria from ‘linear’ approximation evaluated on generalized Border protection game on the weakly connected graph with width 3.	32
6.4 Nash equilibria from ‘linear’ approximation evaluated on generalized Transit game with the width 2 and with the number of steps equal to the length + 1.	32
6.5 Nash equilibria from ‘linear’ approximation evaluated on generalized Ticket inspection game with 3-hour long shift, with the number of stations equal to the number of trains +1, and with the approximative number of passengers equal to the 1000 · the number of trains.	32



Chapter 1

Introduction

Many of us like various games and puzzles. Maybe it is the excitement of competition, the feeling of winning, or the solution of a riddle, which makes games popular. Also, many of real-life situations involve these attractive properties. It is no coincidence that many of the games are based on the real-life problems. Typical examples are strategic computer games from the Middle Ages or the board game Scotland Yard. The effort to solve the games goes hand in hand with the number of these problems. *Game theory* deals with this topic.

In game theory, there exist two basic game formalizations: *the normal and the extensive form*. The normal form describes games, where a player has only one decision as in the rock, paper, scissors game. The extensive form is more complex and describes games with sequences of decisions. An example of an extensive-form game is chess. In game theory, we evaluate the result of the game by a value called utility. For example, a player gets 1 for victory, -1 for losing and 0 for a tie.

The fundament of solving a game is to find a rational strategy, which has the maximum probability of winning or returns a maximum gain. Game theory gives us solution concepts, which tries to provide us with this strategy. Every game contains such strategies for each player that no-one can increase his gain by altering the strategy. These strategies are called *Nash equilibrium*, and they rank among these concepts.

Finding such strategies without computers is almost impossible. There exist standard algorithms for solving games in the normal form. Nevertheless, games in the normal form are exponentially large for games with sequences of moves. That is why, we need something, which is faster and usable for larger games. Thus, there exist also other formalizations of games as *Normal-form games with sequential strategies* (NFGSS). In NFGSS, there are two players, and both of them make sequences of decisions as in the extensive form, but they can not observe the impact of moves of the opponent immediately. The utility is computed as a sum of ‘partial utilities’ of all pairs of actions of both players. It gives us a possibility to calculate a gain of choosing a move. As an example of NFGSS can be a cop and robber game. There are a police officer and the robber in a town represented by a graph. The robber tries to escape, and the cop wants to catch him. When the robber leaves the town, wins, and

whenever the cop catches the robber, gains a payoff. The game is limited by a number of steps because we want to keep it finite.

These NFGSS games can be represented in the extensive form and the normal form, but there exist more efficient algorithms applied directly on NFGSS. An example is *Counterfactual regret minimization* algorithm (CFR), which iteratively finds the optimal strategy in self-play.

Many real-world scenarios, however, do not exactly correspond to NFGSS. For example, in the cop and robber game, mentioned above, the policeman can catch the robber multiple times, and the game does not end. For this scenario, it is more realistic to end after catching the robber. The robber is arrested, he receives the negative utility, and can not leave the town anymore. This problem needs another model. We call it *generalized NFGSS* or nonlinear NFGSS. There is no restriction on the computation of utility, but it can be computed only at the end of the game.

The goal of this work was to find Nash equilibria in generalized NFGSS. First, we evaluated the Nash equilibrium from NFGSS in the generalized NFGSS, and we tried to find out if this estimate is usable as a heuristic. Then we created the solver, which uses the transformation of NFGSS to the normal form and the standard algorithm for solving it. Finally, we adapted the *Monte Carlo extension of CFR algorithm* (MCCFR) for the NFGSS, and we utilize the property, that it do not need to compute utility before the end of the game. Therefore, we also used this algorithm for solving generalized NFGSS. However, the strategy found by MCCFR algorithm do not need to converge to the Nash equilibrium in generalized NFGSS, because the linearity of NFGSS utility is necessary for the proof of it. We have tested all these three approaches to find out which one gives reasonable results in a shorter time.

1.1 Overview

- In Chapter 2 are formalizations of the normal form and the extensive form
- Nash equilibrium can be found in the introduced game models by algorithms from Chapter 3
- NFGSS and solving methods for them are in Chapter 4
- In Chapter 5, we present the adaptation of MCCFR for NFGSS, and we introduce methods for solving generalized NFGSS
- Chapter 6 contains the experimental results of all presented methods for solving generalized NFGSS
- Finally, we conclude all algorithms and results in Chapter 7

Chapter 2

Background

In this chapter, we will introduce the concept of games and strategies in them. We will define games in normal form and show some examples of it. Then we will move to extensive-form games with perfect and with imperfect information. Finally, we will show a concept of perfect recall. All definitions are from [1].

2.1 Normal-Form Games

The first and the most basic game model is the normal form. It represents games, where all players make only one decision. Afterwards, each player receives a reward or a penalty based on the choices.

2.1.1 Definition

The game in normal form is defined as a tuple (N, A, u) , where N is a set of n players, also called agents, $A = A_1 \times \dots \times A_n$, where A_i is a set of possible actions of player i and $u = (u_1, \dots, u_n)$, where $u_i: A_i \mapsto \mathbb{R}$ is a utility function.

A classical example is the popular game rock, paper, scissors.

		Player 2		
		Rock	Paper	Scissors
Player 1	Rock	0, 0	-1, 1	1, -1
	Paper	1, -1	0, 0	-1, 1
	Scissors	-1, 1	1, -1	0, 0

Figure 2.1: Payoff matrix of rock, paper, scissors game
(Figure 3.7 in [1])

This game is in the normal form represented by a matrix of payoffs. Each player has three possible actions: rock, paper, and scissors. If the first player plays paper and the second plays scissors, they get payoff written in the second row and the third column, which means the first player got -1 and the second 1.

We focus on two-players zero-sum games, where there are only two active agents and utility for each combination of actions of players sums to 0, i.e.

$u_1 = -u_2$. Rock, paper, scissors game is a simple example of a two-players zero-sum game.

2.1.2 Strategy

A *pure strategy* is defined as that a player chooses his actions deterministically[1]. Players can also play with some randomization. For example, they throw a crown before choosing an action. It is called a *mixed strategy*, and it is defined as a probability distribution over all actions[1].

For example in the rock, paper, scissors game a player in a mixed strategy can choose paper with the probability of $\frac{1}{2}$ and rock with the probability of $\frac{1}{2}$.

A *strategy profile* is called a tuple of pure or mixed strategies for each player[1]. For a strategy profile, we can compute an expected output for the player i . It is a sum of the utilities of outcomes multiplied by the probability of reaching the corresponding outcome under the strategy profile.

Formally:

$$u_i(\sigma) = \sum_{a \in A} u_i(a) \prod_{j=1}^n \pi_j^\sigma(a_j)$$

Where $\sigma = (\sigma_1, \dots, \sigma_n)$ is a strategy profile $A = A_1 \times \dots \times A_n$ is a set of actions, $u_i(a)$ is a utility for the player i , when $a = (a_1, \dots, a_n)$ is played and $\pi_j^\sigma(a_j)$ is the probability, that the player j plays the action a_j . It should be noted, that we use u_i for utility and also for expected utility.

For example, we can compute the expected utility for the first player in the rock, paper, scissors game for a strategy profile $\sigma = (\sigma_1, \sigma_2)$, where $\sigma_1 = (0, \frac{1}{2}, \frac{1}{2})$ and $\sigma_2 = (\frac{1}{2}, \frac{1}{2}, 0)$. It means that the player one plays only paper and scissors with the same probability and the player two plays only rock and paper with the same probability. The expected utility for the player one will be:

$$u_1(\sigma) = 1 \cdot \frac{1}{4} + 0 \cdot \frac{1}{4} - 1 \cdot \frac{1}{4} + 1 \cdot \frac{1}{4} = \frac{1}{4}$$

Since the rock, paper, scissors game is two-player and zero-sum, $u_2(\sigma) = -u_1(\sigma) = -\frac{1}{4}$.

2.2 Extensive-Form Games

The normal form is the basic and straightforward form. But not always it is reasonable to represent a game by a matrix. Extensive form is introduced for games, where players do not need to make decisions simultaneously. The representation as a table is not suitable for these games because, in general, the size of it grows exponentially.

2.2.1 Perfect-Information Games

Perfect-information games (in the extensive form) are games, which can be represented by a tree. Each node in the tree is a state of the game where one

player makes a choice, each edge is a possible action, and in leaves, there are payoffs given by the game. This tree is also known as a game tree.

Formally, the perfect-information game is a tuple $(N, A, H, Z, \chi, \rho, \gamma, u)$, where N is a set of n players, A is a set of actions, H is a set of nonterminal choice nodes, Z is a set of terminal nodes. $\chi: H \mapsto 2^A$ is a function defining possible actions in each choice node. $\rho: H \mapsto N$ is a function defining which player is on the move in each choice state. $\gamma: H \times A \mapsto H \cup Z$ is a function defining successor nodes when an action is played in a choice node. And $u = (u_1, \dots, u_i, \dots, u_n)$, where $u_i: Z \mapsto \mathbb{R}$ is an utility function defined for terminal nodes[1].

As an example, we can get a sharing game [1]. Two siblings have to share two indistinguishable gifts. The brother has three choices. He can offer both gifts to the sister or keeps both or each of them can keep one. The sister can accept or reject this offer. If she rejects it, none of them gets gifts. This game is represented by the following game tree:

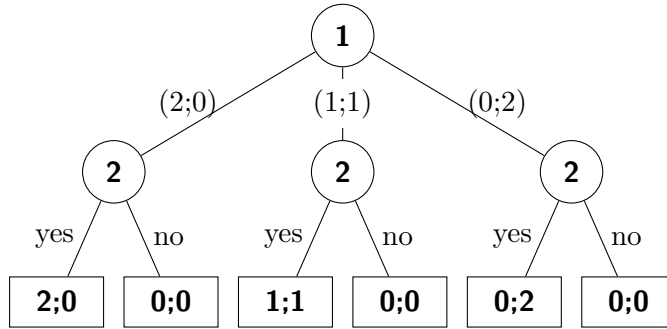


Figure 2.2: The perfect information game tree of the Sharing game (Figure 5.1 in [1])

2.2.2 Strategies in Extensive-Form Games

A pure strategy in extensive-form games is defined as the Cartesian product $\times_{s \in H, \rho(s)=i} \chi(s)$. It means that in each choice node one action is chosen, whether it is possible to reach that choice node or not. A mixed strategy is a probability distribution over pure strategies.

All possible pure strategies in the Sharing game (Figure 2.2) are:

$$S_1 = \{(2 - 0), (1 - 1), (0 - 2)\}$$

$$S_2 = \{(yes, yes, yes), (yes, yes, no), (yes, no, yes), (yes, no, no), (no, yes, yes), (no, yes, no), (no, no, yes), (no, no, no)\}$$

In extensive-form games, there is also defined a *behavioral strategy*. Mixed strategies are probability distributions over pure strategies, which is like a probability distribution over vectors of decisions. Unlike that, a behavioral strategy is a vector of probability distributions over possible choices[1].

Formally, let $\Delta(A)$ be a set of all probability distributions over A , the behavioral strategy of player i is a function $\sigma_i(s) \in \Delta(\chi(s)); \forall s \in H : \rho(s) = i$

2.2.3 Imperfect-Information Games

In perfect-information games, players make a decision in each choice node. It means that they exactly know where in the game tree they are. It implies that players remember the whole history of the game. It is a very strong assumption. In some situations, we want to model players with only a partial or no knowledge. Thus, are defined imperfect-information games (in the extensive form).

Imperfect-information game is a tuple $(N, A, H, Z, \chi, \rho, \gamma, u, I)$, where $(N, A, H, Z, \chi, \rho, \gamma, u)$ is a perfect-information game, and $I = (I_1, \dots, I_n)$, where I_i is a set of equivalence classes $I_{i,k}$ on states of the player i , where for each $s, s' \in I_{i,k}$ holds $\chi(s) = \chi(s')$, and $\rho(s) = \rho(s')$.

Imperfect-information game is the same as a perfect information game with the difference that players choice nodes are grouped into *information sets*. A player cannot distinguish between nodes in one information set [1]. Since nodes in one information set are truly indistinguishable, they must have the same possible actions.

We will show it on an example [1]:

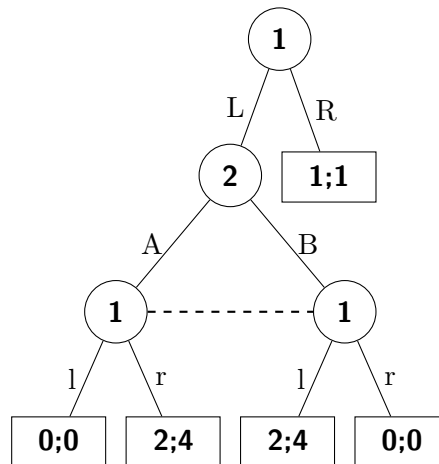


Figure 2.3: An imperfect information game tree. Dashed line connects nodes in the same information set.

(Figure 5.10 in [1])

The player 1 has two information sets in this game. The root node is in one information set, and both other choice nodes are in the second information set. Since a player cannot distinguish in which node he is, pure strategies must be defined as the Cartesian product $\times_{I_{j,k} \in I_i, \rho(I_{j,k})=i} \chi(I_{j,k})$. The actions are chosen in information sets instead of states. The same way a behavioral strategy is defined in imperfect information game.

The example of an imperfect information game (in Figure 2.4) demonstrates that behavioral strategies do not have the same expressivity as mixed strategies. Mixed strategy is a probability distribution over pure strategies. Pure strategies of the player 1 are L and R, and the player 2 has pure strategies U and D. It means that the player 1 plays with probability p the action L and

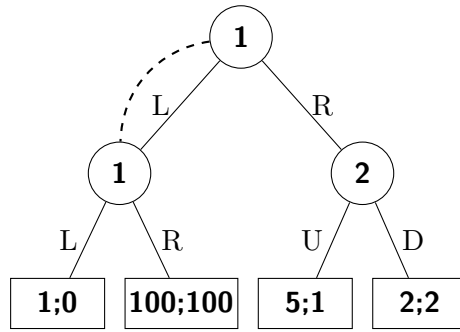


Figure 2.4: An imperfect recall, imperfect information game tree.
(Figure 5.12 in [1])

with probability $1-p$ the action R in all tree. The node (100;100) cannot be reached by a mixed strategy because we need to take for the first time the action L and for the second time the action R in the same information set. By way of contrast, we will reach the leaf node (100;100) with the probability of $\frac{1}{4}$ under the behavioral strategy, which chooses L and R uniformly with the probability of $\frac{1}{2}$.

2.2.4 Alternative Definition

Let $s_0, a_0, s_1, a_1, \dots, s_m, a_m, s$ be a path through a game tree (the s_j is a choice node, and the a_j is an action). The *history* h is an ordered set of actions a_0, a_1, \dots, a_m and $hist(s)$ is the history leading to the state s . For all $s \in Z$ the history $hist(s)$ is called *terminal history*. The set of all histories is named H' , and the set of all terminal histories is denoted Z' . Indeed holds $Z' \subseteq H'[2]$.

There exists a bijection between nodes and histories. Therefore, the set of terminal nodes Z can be identified with the set of all terminal histories Z' and the set of all nonterminal nodes H can be identified with the set of all nonterminal histories $H' \setminus Z'$. And finally, functions χ, ρ, γ, u can be redefined on sets H' and Z' instead of H and Z .

The root node corresponds to an empty history \emptyset . Let $h = hist(s)$, and a player plays an action a in the node s . Then $h' = ha$, and the history h is called a *prefix* of h' , denoted $h \sqsubseteq h'$. This relation has to be generalized with transitivity closure (if $h' = ha_1a_2$, then $h \sqsubseteq h'$) and reflexivity closure ($\forall h: h \sqsubseteq h$)[2].

We can also introduce an information set alternatively. If two nodes s, s' are in one information set, then the histories $hist(s)$ and $hist(s')$ are also in the same information set. Similarly $hist(I) = \{hist(s) | s \in I\}$. For history h , we also define $I(h)$ as the information set containing the history h [2].

A $z[I]$, where z is a terminal history, and I is an information set, denotes the history h , such that $h \sqsubseteq z$ and $h \in I$ [2].

2.2.5 Perfect Recall

The player i has a perfect recall, if for any s, s' from the same information set and for any paths $s_0, a_0, s_1, a_1, \dots, s_m, a_m, s$ and $s_0, a'_0, s'_1, a'_1, \dots, s'_m, a'_m, s'$, where s_0 is the root node, holds $m = m', \forall 0 \leq j \leq m : \rho(s_j) = i$, then s_j and s'_j are in the same information set and $a_j = a'_j$.

In other words, the player i has the perfect recall if all states of the player i in one information set have the same history of his nodes and chosen actions from the root node. It means that he does not forget the history of his nodes and actions.

If every player has a perfect recall, then the game is called the game of perfect recall. Otherwise, the game has an imperfect recall. It can be observed, that every perfect-information game is a game of perfect recall.

In games of perfect recall, a mixed strategy can be replaced by an equivalent behavioral strategy and vice versa. It means that behavioral strategies and mixed strategies have the same expressivity in the game of perfect recall.[1]

Chapter 3

Solving Games

In this chapter, we will introduce the concept of solving games. We assume that all players are rational and want to get the maximum utility. When there is only one agent, he can play the action, which gives him the maximal utility. The situation is much more complicated in games with two and more players. Therefore, the concepts of best response and Nash equilibrium are introduced.

The second problem is finding the strategies of the best response and Nash equilibrium. There are many options, which algorithm to use. We will focus on LP solving of two-players zero-sum games in the normal form, on Counterfactual regret minimization for the extensive-form games, and on Monte Carlo extension of CFR. These algorithms will be useful in the next parts of this text.

3.1 Solution Concepts

In games with more players, there we can not talk about the best strategy for the player, since the expected utility of the player depends on strategies of other players. Therefore, some outcomes are grouped into sets called solutions concepts, which in a certain perspective substitute ‘the best strategy.’ One of these solution concepts is Nash equilibrium.

3.1.1 Best Response

All players in the game are rational. That is why they want to maximize the expected utility. When a player knows strategies of opponents, he can simply find a strategy, which gives him the maximum expected utility. This concept is called a *best response* strategy.

Formally: Let $\sigma = (\sigma_1, \dots, \sigma_n)$ be a strategy profile. The strategy σ_i is the best response strategy of player i on other players strategies $\sigma_j; j \neq i$, if and only if for all strategy profiles $\sigma^* = (\sigma_1, \dots, \sigma_i^*, \dots, \sigma_n)$ holds $u_i(\sigma) \geq u_i(\sigma^*)$ [1].

3.1.2 Nash Equilibrium

Nash equilibrium is a strategy profile in which each player plays the best response strategy to all other players strategies. It means that no player can get a better expected utility by changing his strategy when strategies of other players stay the same[1].

We will demonstrate it on a simple example. We have two companies. Each one can discover new technologies or stay put. If one of them discovers technologies and the other does not, the first one gains a competitive advantage and gets the payoff $a > 0$ and the second one gets the payoff $-a$, otherwise, the utility of both is 0 . This game can be represented by the following matrix.

		Company 2	
		Discover	Stay
Company 1	Discover	0, 0	a, -a
	Stay	-a, a	0, 0

Figure 3.1: Payoff matrix of Companies competition game

There can be easily seen the Nash equilibrium in the first row and the first column. If one company changes strategy to stay, it gets the utility $-a$ instead of 0 , which is worse.

This was an example of pure-strategy Nash equilibrium. In the game rock, paper, scissors there is no such pure equilibrium. We must use the mixed strategies $\sigma_1 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ and $\sigma_2 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ to satisfy the condition of Nash equilibrium.

John Nash proved that in every game there exists at least one Nash equilibrium[3].

3.1.3 ϵ -Nash Equilibrium

ϵ -Nash equilibrium is a strategy profile, in which one of the players by changing his strategy can get maximum gain equal to ϵ .

Formally ϵ -Nash equilibrium is a strategy profile $\sigma = (\sigma_1, \dots, \sigma_n)$ for fixed ϵ if, for every player i and for all strategies $\sigma'_i \neq \sigma_i$ holds: $u_i(\sigma_1, \dots, \sigma_i, \dots, \sigma_n) \geq u_i(\sigma_1, \dots, \sigma'_i, \dots, \sigma_n) - \epsilon$. [1]

In every game, the ϵ -Nash equilibrium exists for any $\epsilon > 0$. If we go with ϵ to 0, ϵ -Nash equilibrium strategy profile goes to the Nash equilibrium strategy profile. It is useful for the computation of the Nash equilibrium. When we do not need the exact Nash equilibrium strategy profile, we can find ϵ -Nash equilibrium with any precision *epsilon*. Thus, it is not necessary to go through the whole continuous space of strategy profiles, and we still can get the desired accuracy[1].

3.2 Finding Nash Equilibria in Normal-Form Games

In the previous section, we have introduced the concept of Nash equilibrium. Now we will look at the basic algorithms for computing it in normal-form games. We will focus on the computation of the Nash equilibrium in two-players zero-sum games.

The minmax theorem shows that the expected utility of player i U_i^* in two-players zero-sum games is equal in all Nash equilibria[1]. Therefore, we can compute the expected utility U_1^* by finding a minmax strategy for the player 2. It leads to the following linear program (LP)[1]:

$$\min U_1^* \quad (3.1)$$

$$s.t. \sum_{k \in A_2} u_1(a_1^j, a_2^k) \cdot s_2^k \leq U_1^* \quad \forall j \in A_1 \quad (3.2)$$

$$\sum_{k \in A_2} s_2^k = 1 \quad (3.3)$$

$$s_2^k \geq 0 \quad \forall k \in A_2 \quad (3.4)$$

In this program s_2 and U_1^* are variables, and terms $u_1(\cdot)$ are constants. We minimize the utility U_1^* (3.1). Therefore, the constraint (3.2) will be satisfied with equality. It means that the utility U_1^* is the expected utility of the best response strategy of player 1 to a mixed strategy s_2^k . The linear program also finds such strategy of player 2, s_2^k , that the expected utility of player 1, U_1^* , is minimal. $U_2^* = -U_1^*$, which implies that the utility U_2^* is maximal. Therefore, it is the utility of the best response strategy. Because both U_1^* and U_2^* are the best response utilities, s_2^k is the mixed strategy of the second player in the Nash equilibrium. Formulas (3.3) and (3.4) guarantee that s_2 is a probability distribution.

We can write a dual problem to the previous linear program:[1]

$$\max U_1^* \quad (3.5)$$

$$s.t. \sum_{j \in A_1} u_1(a_1^j, a_2^k) \cdot s_1^j \geq U_1^* \quad \forall k \in A_2 \quad (3.6)$$

$$\sum_{j \in A_1} s_1^j = 1 \quad (3.7)$$

$$s_1^j \geq 0 \quad \forall j \in A_1 \quad (3.8)$$

There are variable s_1 , and U_1^* and terms $u_2(\cdot)$ are constants. The lines (3.5) and (3.6) find a strategy of player 1, s_1 , which maximizes the expected utility U_1^* against all possible strategies of player 2. Thus, it finds the mixed strategy of the player 1, s_1 , and the maximum utility U_1^* in the Nash equilibrium.

3.3 Finding Nash Equilibria in Extensive-Form Games

In this section, we will focus on computing Nash equilibria in extensive-form games. It is, in general, more complicated than in the normal form. We will describe an approximative algorithm, Counterfactual regret minimization, which iteratively updates strategies of all players. The strategies are ϵ -Nash equilibrium for $\epsilon \rightarrow 0$ when the number of iterations $\rightarrow \infty$. The second algorithm, which we will introduce is the Monte Carlo Counterfactual regrets minimization. Both these algorithms will be used in the next parts.

3.3.1 Counterfactual Regret Minimization

Counterfactual regret minimisation is an algorithm for computing Nash equilibria in extensive-form games. It iteratively modifies strategies of both players in self-play and converges to the Nash equilibrium. In this algorithm we define for each information set a *counterfactual value* as follows: [2]

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

This value represents the expected utility of player i in an information set I when he plays the strategy σ and his opponent plays the strategy σ_{-i} . Z_I is the set of all terminal histories z , such that $hist(I) \sqsubseteq z$. $\pi_{-i}^\sigma(z[I])$ is the probability, that the opponent will play actions from the history $hist(I)$ and $\pi^\sigma(z[I], z)$ is the probability of reaching the terminal history z using the strategy σ from the end of $hist(I)$ [2]. It gives us a sum of expected utilities for having reached the history from I weighted by the probability, that opponent will reach I [4].

The main idea of CFR is computing regrets iteratively. Regret is the difference of utilities between playing one best action and playing by actual strategy[4]. Specifically in CFR meaning an action regret in time t is the difference between the expected utility gained by playing strategy σ^t and choosing an action a in the information set I . Formally, the regret of not choosing an action a in the information set I :

$$r^{t+1}(I, a) = v_i(I, \sigma_{I \rightarrow a}^t) - v_i(i, \sigma^t)$$

A strategy $\sigma_{I \rightarrow a}^t$ is identical as σ^t , only in the information set I is chosen the action a with the probability of 1[2].

Action regrets are summed over time to cumulative regrets for each information set and each action:

$$Q^{t+1}(I, a) = \max\{0, Q^t(I, a) + r^{t+1}(I, a)\}$$

And $Q^0 = 0$ for every action, every player, and in every information set[2].

Finally, the new strategy is recomputed:

$$\sigma^{t+1}(I, a) = \begin{cases} \frac{Q^t(I, a)}{Q_{sum}^t(I)} & \text{if } Q_{sum}^t(I) > 0 \\ \frac{1}{|A(I)|} & \text{otherwise} \end{cases}$$

Where $Q_{sum}^t(I) = \sum_{a' \in A(I)} Q^t(I, a')$ and $A(I)$ is the set of actions possible in the information set I . This procedure is called Regret matching[2].

Finally, we can compute an average strategy profile $\bar{\sigma}^t(I, a)$. This average profile is the ϵ -Nash equilibrium with $\epsilon \rightarrow 0$, when $t \rightarrow \inf$ [2]. It is calculated:

$$\bar{\sigma}^T(I, a) = \frac{\sum_{t=1}^T \pi_i^{\sigma^t}(I) \cdot \sigma^t(I, a)}{\sum_{t=1}^T \pi_i^{\sigma^t}(I)}$$

Where $\pi_i^{\sigma^t}(I) = \sum_{h \in I} \pi_i^{\sigma^t}(h)$ [4].

3.3.2 MCCFR

MCCFR algorithm is Counterfactual regret minimization algorithm (3.3.1) modified by Monte Carlo approach. In Monte Carlo algorithms we do not compute the exact value, and we only estimate it by sampling. We compute a mean of values gained by samples weighted by probabilities of reaching the sample. When the number of used samples is high, the estimate has to be close to the real value. It is guaranteed by the law of large numbers [4].

Sampled counterfactual regret minimization is the same as CFR, except for the computation of counterfactual values. These are computed by the following equation:[4]

$$v_i(I, \sigma|j) = \sum_{z \in Q_j \cap Z_I} \frac{\pi_{-i}^{\sigma}(z[I]) \cdot \pi^{\sigma}(z[I], z) \cdot u_i(z)}{q(z)}$$

Where Q_j is a chosen set of terminal histories, so that $\bigcup_i Q_i = Z$. And $q(z) = \sum_{j: z \in Q_j} q_j$, where q_j is the probability of picking Q_j and $q(z)$ is the probability of selecting z as a part of Q_j .

In MCCFR outcome sampling we choose one terminal history in each iteration, and we update only the information sets in this selected history. The probability distribution over all samples z is marked $\pi^{\sigma'}(z)$ and σ' is called sampling profile[4].

In this algorithm, we traverse the history similarly as in CFR (in CFR we go through the whole tree). In each step, we compute regrets and update strategies. Almost all equations stay the same as in CFR. The counterfactual values must be normalized by the probability of choosing the sample[4].

$$v_i(I, \sigma) = \frac{\pi_{-i}^{\sigma}(z[I]) \cdot \pi^{\sigma}(z[I], z) \cdot u_i(z)}{\pi^{\sigma'}(z)}$$

Where z is a chosen terminal history.

Formally written computation of action regrets:[4]

$$\begin{aligned}
r(I, a) &= v_i(I, \sigma_{I \rightarrow a}) - v_i(i, \sigma) \\
&= \frac{\pi_{-i}^\sigma(z[I]) \cdot \pi^\sigma(z[I]a, z) \cdot u_i(z)}{\pi^{\sigma'}(z)} - \frac{\pi_{-i}^\sigma(z[I]) \cdot \pi^\sigma(z[I], z) \cdot u_i(z)}{\pi^{\sigma'}(z)} \\
&= \frac{\pi_{-i}^\sigma(z[I]) \cdot u_i(z)}{\pi^{\sigma'}(z)} \cdot (\pi^\sigma(z[I]a, z) - \pi^\sigma(z[I], z)) \\
&= W \cdot (\pi^\sigma(z[I]a, z) - \pi^\sigma(z[I], z))
\end{aligned}$$

The last step is the substitution $W = \frac{\pi_{-i}^\sigma(z[I]) \cdot u_i(z)}{\pi^{\sigma'}(z)}$

When the action is not a part of the history z , the probability $\pi^\sigma(z[I]a, z) = 0$, which means that $v_i(I, \sigma_{I \rightarrow a}) = 0$ and $r(I, a) = -v_i(i, \sigma)$. [4]

There are two more questions. The first question is how to make the sampling profile σ' . We used the most straightforward approach epsilon-on-policy. It is a mixture of a uniform strategy and $\sigma(I, a)$ in the ratio ϵ and $1 - \epsilon$. Therefore, $\sigma'(I, a) = \epsilon \cdot \text{Uniform}(I) + (1 - \epsilon) \cdot \sigma(I, a)$.

The second one is how to compute the mean strategy, $\bar{\sigma}$, when only few information sets are visited in each iteration. The most straightforward method is called *optimistic averaging*. In this approach, each information set remembers the number of the iteration, when the mean strategy has been last updated. This counter is named c_I . Initially c_I is equal to 0 for all I . Later, when information set is visited, the mean strategy is incremented by $(t - c_I) \pi_i^{\sigma^t}(I) \cdot \sigma^t(I, a)$, and c_I is set to t . This is the mean strategy update from CFR, weighted by the time since the last modification. It simulates using the newly computed strategy from the last visitation of I . Finally, all information sets have to be updated, and the cumulative mean has to be normalized. More information about this and other methods can be found in Marc Lanctot's Ph.D. thesis Section 4.4[4].

Chapter 4

NFGSS

This chapter will introduce the new model of games called Normal-form games with sequential strategies. In this two-players, zero-sum games both players have sequential strategies as in the extensive form, but they cannot observe the impact of actions of the opponent immediately. This model is useful for describing many real problems, typically a movement on the graph in time, where the player cannot observe the movement of the opponent.

4.1 Definition

Normal-form games with sequential strategies (NFGSS) are two-players zero-sum games. Each player has an acyclic Markov decision process (MDP) as his strategy space. It means that player i has a set of states S_i , a set of actions A_i and a stochastic transition function $T: S_i \times A_i \mapsto \Delta(S_i)$, where $\Delta: S_i \mapsto \mathbb{R}$ is a probability distribution over states[2][5]. Intuitively, a player moves between his states by choosing actions. If he chooses an action, the next state is given by a probability distribution. The pure strategy is a set of chosen actions in each state, and mixed strategy is a probability distribution over pure strategies. Utility in NFGSS is defined as follows[5]:

$$u_1(\sigma_1, \sigma_2) = \sum_{S_1 \times A_1} \sum_{S_2 \times A_2} \pi_1^\sigma(s_1, a_1) \cdot \pi_2^\sigma(s_2, a_2) \cdot U((s_1, a_1), (s_2, a_2))$$

Where σ_i is a strategy of a player i , $\pi_i^\sigma(s_i, a_i)$ is the probability, that the player i will reach the state s_i and will play the action a_i under the strategy σ_i . And $U(.,.)$ is the defined utility function for all state-action pairs of both players.

We will demonstrate it on a simple example:

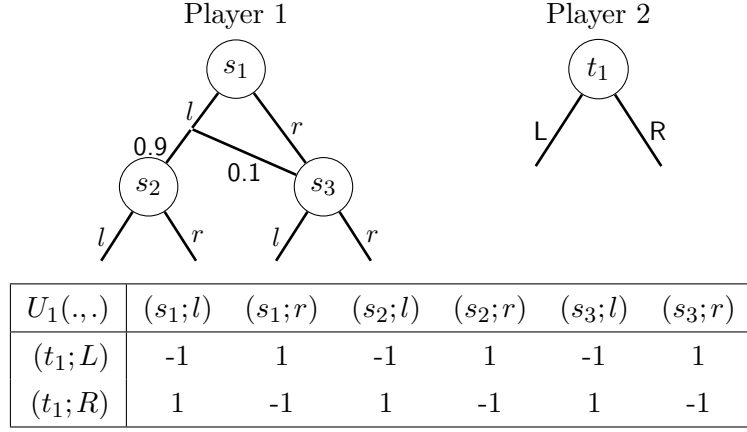


Figure 4.1: Example of NFGSS. At the top there are MDPs of both players and predefined utility is below them.

In this example, we have two players. The first player tries to sneak two illegal packets through a border, but he can carry only one at a time. There are two possible ways, the left, and the right. In the first attempt, he has a messy map. Thus there is the probability of 0.1, that he will choose the left way, but he will go by the right way. In the second attempt, the maps are already clear. The second player, the defender, chooses the crossing at the beginning and he cannot change it. The utility of the smuggler is written in the table.

4.2 Conversion of NFGSS to Normal-Form Game

NFGSS is basically in the normal form. The only thing we need is to extract a set of actions in the sense of the normal-form game actions and then compute the utility for these actions. An action in the normal-form game representation is identical as a pure strategy in NFGSS. Which means it is σ , a set of pairs (s, a) , where s is an MDP state, and a is a chosen action in it [5]. For these actions we can compute utility $u_1(\sigma_1, \sigma_2)$ with the following equation:

$$u_1(\sigma_1, \sigma_2) = \sum_{(s_1, a_1) \in \sigma_1} \sum_{(s_2, a_2) \in \sigma_2} U((s_1, a_1), (s_2, a_2)) \cdot \pi^T(s_1) \cdot \pi^T(s_2)$$

Where $U((s_1, a_1), (s_2, a_2))$ is the utility from the definition of NFGSS and $\pi^T(s_i)$ is the probability of reaching s_i defined by the transition function T .

From the example in Figure 4.1, the sets of actions for players are:

$$A_1 = \{((s_1, l), (s_2, l), (s_3, l)), ((s_1, l), (s_2, l), (s_3, r)), ((s_1, l), (s_2, r), (s_3, l)), ((s_1, l), (s_2, r), (s_3, r)), ((s_1, r), (s_2, l), (s_3, l)), ((s_1, r), (s_2, l), (s_3, r)), ((s_1, r), (s_2, r), (s_3, l)), ((s_1, r), (s_2, r), (s_3, r))\}$$

$$A_2 = \{(t_1, L), (t_1, R)\}$$

And the utility for the third action of player 1 and the first action of player 2 is:

$$\begin{aligned} u_1(((s_1, l), (s_2, r), (s_3, l)), (t_1, L)) &= \\ &= U((s_1, l), (t_1, L)) \cdot 1 + U((s_2, r), (t_1, L)) \cdot 0.9 + U((s_3, l), (t_1, L)) \cdot 0.1 \\ &= -1 \cdot 1 + 1 \cdot 0.9 - 1 \cdot 0.1 = -0.2 \end{aligned}$$

The disadvantage of this representation is the exponential size. It can also be seen in this example, that the MDP with only three states and branching factor 2 corresponds to 8 actions in the normal form representation.

4.3 Conversion of NFGSS to Extensive-Form Game

Each NFGSS can be converted to the extensive form with an imperfect recall [2]. The extensive form can be defined as a set of players, a set of actions, a set of histories, a set of terminal histories, a set information sets, and as a utility function. History is any sequence of actions lying on the path from the root to any state. Terminal history is any sequence of actions lying on the path from the root to any leaf. It is described in Section (2.2.4).

In an extensive-form game derived from NFGSS, the set of players stays the same. The set of actions is a union of state-action pairs of all players and chance actions. Chance actions are triple (s, a, s') with a probability derived from the transition function in the MDP. A history of the player i is a sequence of actions in his MDP, where action of each player is followed by a chance action. The set of histories is the Cartesian product of histories of all players. The set of terminal histories is a set of histories in which all players reach a terminal node in the MDP. Two histories are in the same information set if both lead to the same MDP state. The utility is defined for each terminal history as follows [2]:

$$u_1(z) = \sum_{(s_1, a_1) \in z_1} \sum_{(s_2, a_2) \in z_2} U((s_1, a_1), (s_2, a_2))$$

The algorithm can be shown on the game displayed in Figure 4.1. This game in the exponential form is displayed in the following figure.

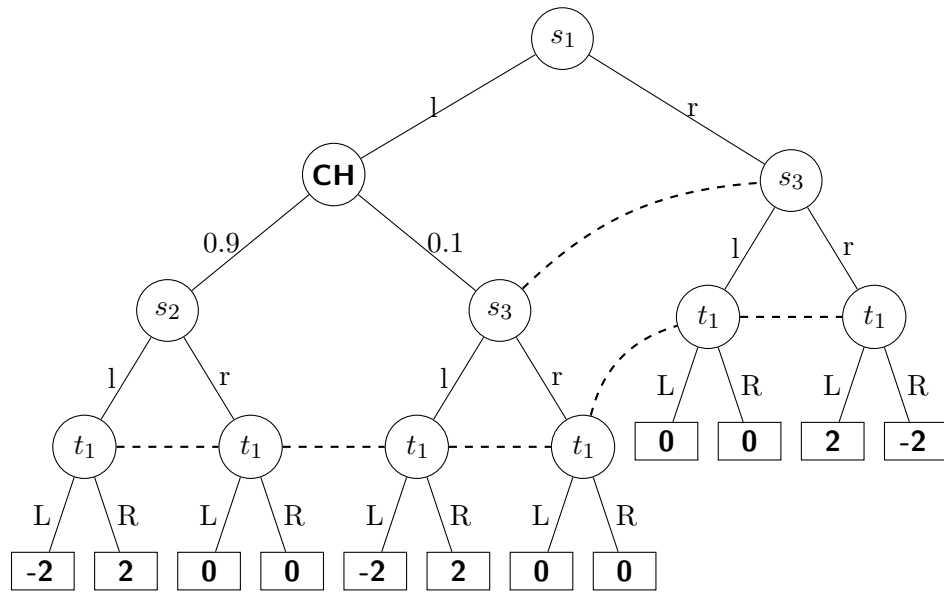


Figure 4.2: The imperfect-recall, imperfect-information, extensive-form representation of the game from Figure 4.1.

In this game, the state-action pairs of the first player and the chance nodes are at the beginning and the state-action pairs of the second player follow afterwards. The information sets are marked out by the dashed lines, and the utility of the history is written in terminal nodes.

The disadvantage of this representation is that these games generally have the imperfect recall. It is unpleasant because extensive-form games with imperfect recall are in general difficult to solve.

4.4 Generalized NFGSS

We wanted to express more realistic scenarios, such as fare evaders in the public transport or a graph searching game. These problems can be simplified as NFGSS, but it has disadvantages. The main problem is that NFGSS utility does not depend on history, which means that linearity is too strong an assumption. To be concrete, when the pursuer in the case of a graph searching game catches the evader, the evader gets a penalty, and the game ends.

4.4.1 Definition

Modified nonlinear NFGSS is defined the same way as NFGSS. It is a two-players zero-sum game, where each player has his own MDP as a strategy space. Strategies and transition probabilities are the same.

The only difference between NFGSS and nonlinear NFGSS is the definition of utility. Let z_i be a terminal sequence of action states in an MDP of player i , then $U(z_1, z_2) = c$, where $c \in \mathbb{R}$.

For example in the game defined in Figure 4.1, the first player has two

attempts. However, more realistic is, that when the smuggler is caught, he cannot smuggle the second package. It leads to the following nonlinear utility:

$U_1(.,.)$	$(s_1;l)$	$(s_1;l)$	$(s_1;l)$	$(s_1;l)$	$(s_1;r)$	$(s_1;r)$
	$(s_2;l)$	$(s_2;r)$	$(s_3;l)$	$(s_3;r)$	$(s_3;l)$	$(s_3;r)$
$(t_1;L)$	-1	-1	-1	-1	0	2
$(t_1;R)$	2	0	2	0	-1	-1

Table 4.1: An example of the definition of nonlinear utility in the game described in Figure 4.1. The utility is defined for the whole sequences in MDPs.

4.4.2 Conversion of Generalized NFGSS to the Normal Form

Conversion of modified NFGSS to the normal-form game can be made similarly as a transformation of ‘linear’ NFGSS into a normal-form game (4.2). The set of actions is the same, only the computation of utility cannot be identical. Therefore, we modify a form of actions for a better estimation of utility.

From the set of pairs (s, a) , σ , defined in Section (4.2), we extract a set Z , which contains all terminal histories $s_1, a_1, \dots, s_n, a_n, s_{n+1}$, where s_1 is the root node, $\forall i : (s_i, a_i) \in \sigma$, s_{n+1} is a terminal node, and $T(s_i, a_i, s_{i+1}) > 0$.

The sets A_i of all actions from the example defined in Figure 4.1 are:

$$A_1 = \{((s_1, l), (s_2, l), (s_3, l)), ((s_1, l), (s_2, l), (s_3, r)), ((s_1, l), (s_2, r), (s_3, l)), ((s_1, l), (s_2, r), (s_3, r)), ((s_1, r), (s_2, l), (s_3, l)), ((s_1, r), (s_2, l), (s_3, r)), ((s_1, r), (s_2, r), (s_3, l)), ((s_1, r), (s_2, r), (s_3, r))\}$$

$$A_2 = \{(t_1, L), (t_1, R)\}$$

The set of terminal histories Z can be written for each element of A_i . For example, the set Z for the third action of the first player is:

$$Z = \{((s_1, l), (s_2, r)), ((s_1, l), (s_3, l))\}$$

From the sets Z_i of both players it is straightforward to calculate the utility $u_1(Z_1, Z_2)$ by the following equation:

$$u_1(\sigma_1, \sigma_2) = u_1(Z_1, Z_2) = \sum_{z_1 \in Z_1} \sum_{z_2 \in Z_2} U(z_1, z_2) \cdot \pi^T(z_1) \cdot \pi^T(z_2)$$

Where $\pi^T(z_i)$ is the probability of reaching terminal history z_i defined by the transition function T as follows: $\pi^T(z_i) = \prod_{(s_j, a_j, s_{j+1}) \text{ part of } z_i} T(s_j, a_j, s_{j+1})$.

An example of the calculation of the utility of the third action of the first player and the first action of the second player:

$$\begin{aligned} u_1(((s_1, l), (s_2, r), (s_3, l)), (t_1, L)) &= u_1(\{((s_1, l), (s_2, r)), ((s_1, l), (s_3, l))\}, \{(t_1, L)\}) = \\ &= U(((s_1, l), (s_2, r)), (t_1, L)) \cdot 0.9 \cdot 1 + U(((s_1, l), (s_3, l)), (t_1, L)) \cdot 0.1 \cdot 1 \\ &= -1 \cdot 0.9 - 1 \cdot 0.1 = -1 \end{aligned}$$

4.4.3 Conversion of Generalized NFGSS to the Extensive Form

Conversion of nonlinear NFGSS to an extensive-form game can be the same as the transformation of standard NFGSS into the extensive form (4.3). Only the utility must be recalculated as follows:

$$u_1(z) = U(z_1, z_2)$$

4.5 Solving NFGSS

As in games in the normal form or the extensive form, we also want to find Nash equilibria in NFGSS. We have shown in the previous section, that NFGSS can be transformed into the normal form and the extensive form. It implies that NFGSS can be solved by algorithms for these two models. The first algorithm in this text is LP finding Nash equilibria for normal-form games. The other approach is using CFR algorithm on the extensive-form representation of NFGSS. We will show, that the CFR can be applied directly on NFGSS.

4.6 Best Response in NFGSS

As in normal-form games, the best response can also be computed in NFGSS on a strategy of opponent σ_{-i} . It is computed recursively by the following program[5]: (s is an MDP state. Externally, the function is called with the root state as s .)

Algorithm 1 Best Response in NFGSS

```

1: function BR( $s, \sigma_{-i}$ )
2:   if  $s$  is terminal then
3:     return 0
4:   for all  $a \in A_1(s)$  do
5:      $v_a \leftarrow \sum_{s_2 \in S_2, a_2 \in A_2} \pi_{-i}^\sigma(s_2, a_2) U((s, a), (s_2, a_2))$ 
6:     for all  $s' \in S_1$  s.t.  $T(s, a, s') > 0$  do
7:        $v_a += BR(s', \sigma_{-i}) \cdot T(s, a, s')$ 
8:    $maxAction \leftarrow argmax_{a \in A_1(s)} v_a$ 
9:   return  $v_{maxAction}$ 

```

This program for each action possible in a state s computes an expected utility (line 4), then it gets the expected utility of the best response recursively for each successor (line 6). Finally, the action with the maximum sum of the expected utilities is selected (line 8). Finally, the value of this action is returned.

4.6.1 LP Finding Nash Equilibria in NFGSS

The first baseline algorithm uses the standard LP for solving games in the normal form since every NFGSS can be represented as an exponentially large normal-form game. We have explained the transformation of NFGSS into the normal form in Section (4.2) and the algorithm for finding Nash equilibria in the normal-form games in Section (3.2).

This solution has a big problem. A number of actions in the normal form and consequently the size of LP grows exponentially with the size of an MDP. Therefore, this method is very demanding regarding space complexity.

There are other methods, which solves NFGSS using LP for solving games in the sequential form, as Compact-Strategy Double-Oracle algorithm [5]. This algorithm uses the linearity of NFGSS and so it is not usable next in this work.

4.6.2 CFR Algorithm on NFGSS

The second algorithm, Counterfactual regret minimization, can be adapted directly on NFGSS [2]. When we transform NFGSS to an extensive-form game, two histories are in the same information set if both lead to the same MDP state. It implies that the probability of reaching the information set is the probability of reaching the MDP state. The strategies of both players can also be updated separately because parts of histories of each player are mutually independent. Finally, all probabilities and expected utilities can be calculated iteratively from the corresponding values in predecessors or successors. More precisely it can be seen in the detailed code:[2]

Algorithm 2 CFR for NFGSS

```

1: procedure NFGSS-CFR
2:    $\forall i \in N; \sigma_i \leftarrow$  uniform strategy
3:    $\forall i \in N; Q_i \leftarrow 0, \bar{\sigma}_i \leftarrow 0$ 
4:   for iteration  $t \in (1, 2, \dots)$  do
5:     for  $i \in N$  do
6:       UpdateStateProbabilitiesMeanStrategies( $-i, t$ )
7:       UpdateActionRegretsCurStrategies( $i$ )
8:    $\forall i \in N; \text{Normalize}(\bar{\sigma}_i)$ 
9: procedure UPDATESTATEPROBABILITIESMEANSTRATEGIES( $i, t$ )
10:   $\forall s_i \in S_i; p(s_i) \leftarrow 0$ 
11:   $p(s_i^0) \leftarrow 1$ 
12:  for  $s_i \in S_i$  in topological order do
13:    for  $a_i \in A(s_i)$  do
14:      for  $s' \in T(s_i, a_i)$  do
15:         $p(s') += p(s_i) \cdot \sigma(s_i, a_i) \cdot T(s_i, a_i, s')$ 
16:         $\bar{\sigma}_i(s_i, a_i) += p(s_i) \cdot \sigma(s_i, a_i)$ 

```

```

17: procedure UPDATEACTIONREGRETSCURSTRATEGIES(i)
18:    $\forall s_i \in S_i; v(s_i) \leftarrow 0$ 
19:   for  $s_i \in S_i$  in reverse topological order do
20:     for  $a_i \in A(s_i)$  do
21:        $v(a_i) \leftarrow 0$ 
22:       for  $s' \in T(s_i, a_i)$  do
23:          $v(a_i) += T(s_i, a_i, s') \cdot v(s')$ 
24:       for  $s_{-i}, a_{-i} \in S_{-i} \times A_{-i}$  do
25:          $v(a_i) += p(s_{-i}) \cdot \sigma(s_{-i}, a_{-i}) \cdot U_i(s_i, a_i, s_{-i}, a_{-i})$ 
26:        $v(s_i) += \sigma(s_i, a_i) \cdot v(a_i)$ 
27:     for  $a_i \in A(s_i)$  do
28:        $Q(s_i, a_i) \leftarrow \max(0, Q(s_i, a_i) + v(a_i) - v(s_i))$ 
29:      $\sigma_i(s_i) \leftarrow \text{RegretMatching}(Q(s_i))$ 

```

In this pseudocode, we store for each MDP state the probability $p(s)$ of reaching it. For each action we store the cumulative action regrets Q , the cumulative mean strategy $\bar{\sigma}$, and the current strategy σ (it can be computed from action regrets instead of storing). Since the MDPs in NFGSS have to be acyclic, we store the states in topological order. When we traverse the states in topological order, we have a guarantee, that predecessors are always solved before the successors.

The algorithm updates strategies of both players alternately in each iteration (line 5). First, it updates the state probabilities and the mean strategy of the opponent player $-i$ (line 6). Then the action values, the cumulative regrets, and the current strategy of the player i are computed (line 7). After T iterations the computation ends and the final cumulative mean strategy $\bar{\sigma}$ has to be normalized (line 8).

The update of state probabilities and the update of the mean strategy is as follows. First, all state probabilities are set to 0 (line 10), only the root state has the probability of 1 (line 11). We iterate over states in topological order (line 12). For each action (line 13) and each successor (line 14), the probability is set to the probability of predecessor multiplied by the probability of choosing the action under the current strategy and by the transition probability (line 15). When the probability is calculated, we can update the mean strategy (line 16).

In the procedure update action regrets current strategies, we traverse states in reverse topological order (line 19). For each action (line 20), we compute the action value as a sum of all expected values of the successors multiplied by the transition probability (lines 22-23) and the marginal utility of that action multiplied by probabilities of the opponent of playing his actions (lines 24-25). The state value is a sum of all action values multiplied by the current strategy (line 26). Finally, for each action cumulative action regrets Q are computed (lines 27-28), and the current strategy is updated by regret matching (line 29).

Chapter 5

Solving Generalized NFGSS

In this chapter, we will focus on finding Nash equilibria in generalized NFGSS. The first baseline algorithm is the standard LP solving algorithm applied on the normal-form representation of generalized NFGSS, as we have shown in Section (4.4.2). However, NFGSS represented in the normal form is exponentially large. Therefore, we modify the MCCFR algorithm on NFGSS and consequently on generalized NFGSS.

5.1 Best Response in Modified NFGSS

Computation of the best response to a strategy of the opponent σ_{-i} in modified NFGSS is also similar as in ‘linear’ NFGSS. It can be written as the following recursive function: (s is the visited state, z is the sequence of states and actions leading to s)

Algorithm 3 Best Response in nonlinear NFGSS

```
1: function BR( $s, z, \sigma_{-i}$ )
2:   if  $s$  is terminal then
3:     return computeExpectedUtility( $z, \sigma_{-i}$ )
4:   for all  $a \in A_1(s)$  do
5:      $v_a \leftarrow 0$ 
6:      $z' \leftarrow (z, (s, a))$ 
7:     for all  $s' \in S_1$  s.t.  $T(s, a, s') > 0$  do
8:        $v_a \text{ += } BR(s', z', \sigma_{-i}) \cdot T(s, a, s')$ 
9:    $maxAction \leftarrow \text{argmax}_{a \in A_1(s)} v_a$ 
10:  return  $v_{maxAction}$ 
```

This program returns in terminal state s the expected utility given from the sequence z leading to s and strategy of opponent σ_{-i} (3). Formally:

$$\text{computeExpectedUtility}(z, \sigma_{-i}) = \sum_{z_{-i} \in Z_{-i}} U(z, z_{-i}) \cdot \pi_{-i}^\sigma(z_{-i})$$

Where $\pi_{-i}^\sigma(z_{-i}) = \prod_{(s,a) \in z_{-i}} \pi_{-i}^\sigma(s, a) \cdot \prod_{(s_j, a_j, s_{j+1}) \text{ part of } z_{-i}} T(s_j, a_j, s_{j+1})$.

If s is not terminal (lines 2-3), the program computes the best response value for each action (lines 4-8). It is calculated as the value returned by the recursive calling of this function for all successors, normalized by transition probability (line 8). Finally, the action with the maximum expected value is selected (line 9).

This function finds the best response against σ_{-i} . In a terminal node, we get an expected utility and then in each node we choose an action, which leads to the maximal expected utility gained by a next sequence. Formally, this argument can be done by the mathematical induction.

5.2 MCCFR algorithm application for NFGSS

The main part of our work is to adapt MCCFR algorithm for NFGSS and utilize the property, that we need only the utility of entire samples. With these prerequisites, we use the major part of the CFR algorithm adapted on NFGSS (Algorithm 3). Only the procedure *UpdateActionRegretsCurStrategies* uses the linearity of the utility in NFGSS. Therefore it must be modified in the MCCFR way.

Each NFGSS can be transformed into an extensive-form game. In the method of turning NFGSS into the extensive form (4.3), the terminal history of the game is a terminal sequence of moves in an MDP of the player i followed by a terminal sequence of steps of the player $-i$. Thus, the part of the player i we can call z_i and the part of the opponent z_{-i} .

In MCCFR for extensive-form games, we get a set of terminal histories updated in one step, and we call it Q_j . We use the property of splitting history on z_i and z_{-i} , and we take as Q_j a sequence z_i and all possible continuations of it. These continuations are all possible terminal sequences of moves in an MDP of the opponent. And it gives us, that the probability of taking the set Q_j is $q(z) = q_j = \pi^\sigma(z_i)$, where $\pi^\sigma(z_i)$ is the probability of reaching the sequence z_i by the strategy σ . It is because all histories from Q_j start by the sequence z_i .

With this knowledge, we can rewrite the computation of counterfactual value:

$$\begin{aligned}
v_i(I, \sigma|j) &= v_i(I, \sigma|z_1) \\
&= \sum_{z \in Q_j \cap Z_I} \frac{\pi_{-i}^\sigma(z[I]) \cdot \pi^\sigma(z[I], z) \cdot u_i(z)}{q(z)} \\
&= \sum_{z \in Q_j \cap Z_I} \frac{\pi_{-i}^\sigma(z[I]) \cdot \pi_{-i}^\sigma(z[I], z) \cdot \pi_i^\sigma(z[I], z) \cdot u_i(z)}{q(z)} \\
&= \sum_{z \in Q_j \cap Z_I} \frac{\pi_{-i}^\sigma(z) \cdot \pi_i^\sigma(z[I], z) \cdot u_i(z)}{q(z)}
\end{aligned}$$

$$\begin{aligned}
&= \sum_{(z_i, z_{-i}) \in Q_j \cap Z_I} \frac{\pi_{-i}^\sigma(z_{-i}) \cdot \pi_i^\sigma(z_i[I], z_i) \cdot u_i(z_i, z_{-i})}{\pi^{\sigma'}(z_i)} \\
&= \sum_{z_{-i}} \frac{\pi_{-i}^\sigma(z_{-i}) \cdot \pi_i^\sigma(z_i[I], z_i) \cdot u_i(z_i, z_{-i})}{\pi^{\sigma'}(z_i)} \\
&= \frac{\pi_i^\sigma(z_i[I], z_i)}{\pi_{-i}^{\sigma'}(z_i)} \sum_{z_{-i}} \pi_{-i}^\sigma(z_{-i}) \cdot u_i(z_i, z_{-i})
\end{aligned}$$

Similarly, as in general MCCFR Outcome sampling, we can make a substitution $W = \frac{\sum_{z_{-i}} \pi_{-i}^\sigma(z_{-i}) \cdot u_i(z_i, z_{-i})}{\pi^{\sigma'}(z_i)}$. The computations of action regrets can be done in the same way as in MCCFR Outcome sampling, described in Section (3.3.2), only W must be substitute differently and terminal history z is restricted on sequence z_i .

The numerator of the W , $\sum_{z_{-i}} \pi_{-i}^\sigma(z_{-i}) \cdot u_i(z_i, z_{-i})$, can be computed as the expected utility gained from a pure strategy of the player i , which leads to the z_i , and the strategy σ_{-i} of the opponent player $-i$.

Formally:

$$\sum_{z_{-i}} \pi_{-i}^\sigma(z_{-i}) \cdot u_i(z_i, z_{-i}) = \sum_{s_i, a_i \in z_i} \sum_{s_{-i} \in A_{-i}} \sigma_{-i}(s_{-i}, a_{-i}) \cdot U((s_i, a_i), (s_{-i}, a_{-i}))$$

The modified function UpdateActionRegretsCurStrategies is depicted in the following pseudocode:

Algorithm 4 MCCFR in NFGSS

```

1: procedure UPDATEACTIONREGRETSCURSTRATEGIES( $i$ )
2:    $(p^{\sigma'}, z_i) \leftarrow \text{getRandomSample}(i)$ 
3:    $W \leftarrow \text{computeExpectedUtility}(z_i, \sigma_{-i}) / p^{\sigma'}$ 
4:    $p \leftarrow 1$ 
5:   for all  $(s_i, a_i) \in z_i$  in reverse topological order do
6:     if  $\text{isDefined}(s_{succ})$  then
7:        $p * = T(s_i, a_i, s_{succ})$ 
8:        $s_{succ} \leftarrow s_i$ 
9:       for all  $a \in A(s_i)$  do
10:        if  $a = a_i$  then
11:           $Q(s_i, a) += W \cdot p$ 
12:           $Q(s_i, a) -= W \cdot p \cdot \sigma_i(s_i, a)$ 
13:           $Q(s_i, a) \leftarrow \max(0, Q(s_i, a))$ 
14:         $p * = \sigma(s_i, a_i)$ 
15:         $\sigma_i(s_i) \leftarrow \text{RegretMatching}(Q(s_i))$ 

```

First, we get a random sample in the MDP of the player i and compute the probability of selecting it (line 2). The value and the probability are used for computing the substituent W (line 3), and in the variable p there is saved the probability of reaching the sample from the actual state. The

states from the sample are iterated from the leaf to the root state (line 5). If the state s is not the leaf, p is multiplied by the transition probability (lines 6-7). Then we calculate the cumulative action regret for each action (lines 9-13). If the action a is in the sample, W multiplied by p , the probability of reaching the leaf state, is added to the action regret (lines 9-10). Then the W multiplied by p and by the strategy σ is subtracted from the action regret (line 11). Finally, the probability p is multiplied by the probability of choosing the action from the sample by the current strategy σ (line 14), and the current strategy is updated by the regret matching (line 15).

■ 5.2.1 MCCFR Algorithm for Modified NFGSS

The CFR algorithm for NFGSS needs to know the utility for each state-action pair. Thus, it cannot be modified to generalized NFGSS. In contrast, MCCFR algorithm needs to know the utility of only terminal histories. Therefore, the computation of counterfactual values can be rewritten to a form with nonlinear NFGSS utility $U(z_1, z_2)$:

$$\begin{aligned} v_i(I, \sigma | j) &= \frac{\pi^\sigma(z_1[I], z_1)}{\pi^{\sigma'}(z_1)} \sum_{z_2} \pi^\sigma(z_2[I], z_2) \cdot u_i(z_1, z_2) \\ &= \frac{\pi^\sigma(z_1[I], z_1)}{\pi^{\sigma'}(z_1)} \sum_{z_2} \pi^\sigma(z_2[I], z_2) \cdot U(z_1, z_2) \cdot \text{sign}(i) \end{aligned}$$

Where $\text{sign}(1) = 1$ and $\text{sign}(2) = -1$.

Pseudocode stays the same, only the function *computeExpectedUtility*(z_i, σ_{-i}) must be defined with respect to the nonlinear utility. In detail it is described in Section (5.1).

It is not formally shown that this algorithm converges to Nash equilibrium. The experiments suggest that it can work for a reasonable large class of practical problems.

Chapter 6

Experimental Evaluation

In this chapter, we will present results of experiments on Monte Carlo CFR used on nonlinear NFGSS. First, we will state all circumstances under which the experiments were performed, then we will introduce game instances on which the algorithms run, and finally, we will discuss the outcomes and possible reasons for them.

6.1 Used Software and Resources

The implementation is created as an extension of Game-Theoretical Library¹ created by Game Theory group from Artificial Intelligence Center² at Czech Technical University in Prague. GT Library is written in Java, containing, among other things, many algorithms for solving normal-form and extensive-form games and algorithms for finding Nash equilibria in NFGSS as Double-Oracle or Counterfactual regret minimization. There are also implemented some domains of NFGSS as Ticket inspection game, Transit game, and Border protection game i.e. Border patrolling game, which we have used in the experiments.

On these bases, the system for finding Nash equilibria in NFGSS by conversion to the normal-form and solving LP has been created. In our implementation, the linear programs are solved in the IBM ILOG CPLEX Optimization Studio³. It is a large optimization package, which solves mainly integer and linear programming problems using various methods. We call it through an integrated interface from Java, and we use only default methods for solving LP described in Section (3.2).

Next, CFR has been reimplemented and transformed to MCCFR, which contains the implementation of the best response algorithm for NFGSS and generalized NFGSS. Finally, Transit game, Border protection game, and Ticket inspection game have been modified and extended by nonlinear utility functions.

The versions of used software are listed in the following table:

¹Download: <http://jones.felk.cvut.cz/repo/gtlibrary>

²More info: <http://aic.fel.cvut.cz/>

³<http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/>

Software	Version
Java	1.8.0_60
GT Library	from 1. 11. 2016
IBM ILOG CPLEX	12.4

Table 6.1: Table of versions of the software used in the experiments.

The experiments were performed on distributed computing infrastructure managed by virtual organization MetaCentrum⁴, to which we would like to give thanks once again for the provided grid services. The tests were performed mainly on computers from clusters listed in the table:

Cluster	CPU	SPECfp2006 performance per core
doom	2x 8-core Intel Xeon E5-2650v2 2.60GHz	33.9
gram	2x 8-core Intel Xeon E5-2670 2,6GHz	30.3
ida	2x 10-core Intel E5-2650v3 3GHz	33.2
lex	2x 8-core Intel Xeon E5-2630v3 2.40GHz	35.6
manegrot	4x 8-core Intel Xeon E5-4627v2 (3.30GHz)	34.375
meduseld	4x 14-core Intel Xeon E7-4830 v4 (2.00GHz)	26.6
tarkil	2x 12-core Intel Xeon E5-2650v4 (2.20GHz)	32.9
zubat	2x 8-core Intel Xeon E5-2630v3 2.40GHz	33.9

Table 6.2: Table of the clusters used in the experiments.⁵

6.2 Game Domains

Three game domains have been expanded by the implementation of nonlinear utility functions. Modified games are Border protection game, Transit game, and Ticket inspection game.

⁴More info: <https://metavo.metacentrum.cz/en/index.html>

⁵All clusters with detailed info: <https://metavo.metacentrum.cz/pbsmon2/nodes/physical> [6]

6.2.1 Transit Game

Transit game domain was used for evaluation in [2] and [5]. It is a basic search game played on an undirected graph, which is formed as a grid displayed in Figure 6.1. It is determined by three parameters: width, length, and a number of steps. When the number of steps is reached, the game ends. At each step, the player can either try to move or stay in place. Every try to move fails with the probability of 0.1. The players move simultaneously.

There are two players: an evader and a defender. The evader starts at any node most to the left. It is formulated as starting at the node s_e on time -1. His goal is to cross the grid from left to right without meeting the defender. If he successfully gains the right side of the grid, he gets utility 1. With every step, he gets a small penalty 0.02.

The defender starts on the grayed node s_d in the middle of the most bottom side of the graph. If he catches the evader, he gets payoff 1 and the game ends, and all next steps are evaluated with 0 utility. When the defender does not meet the evader during the game, and he does not return to the base, which corresponds to the starting node, he gets a penalty 20.

The game has to be zero-sum. Hence, a gain of the defender is a penalty of the evader and vice versa.

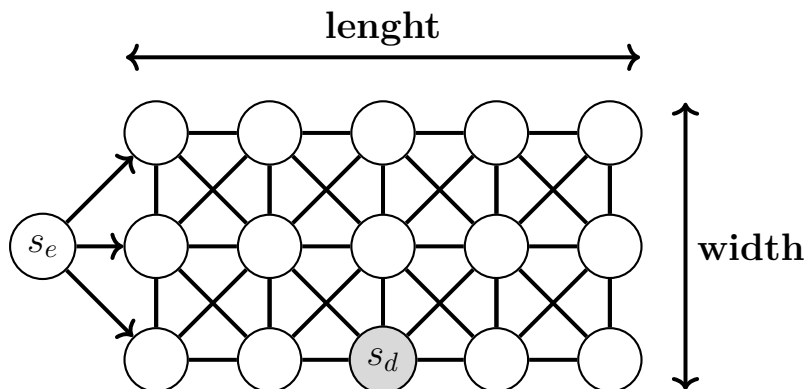


Figure 6.1: Scheme of a graph of Transit game.

6.2.2 Border Protection Game

Border protection game was used for evaluation in [5]. It is played on a directed graph shown in Figure 6.2 in three variants. It is determined by two parameters: width and a number of steps. As well as in Transit game, the number of steps determines the end time of the game.

There are also two players: an evader and a patroller. The evader starts at the leftmost node E and attempts to safely cross the graph to the rightmost node D. The evader gains the utility 2 for reaching the goal node. The evader moves by every step, only in grayed areas he can stay in the same node.

The patroller controls two units, which move simultaneously in the intermediate nodes. These nodes are in gray shaded areas in Figure 3.2. The units can start in any one of them, and they can in each step either move to the

neighbor node or stay at the same place. The patroller can observe traces, which the evader has left behind in previous steps with the probability of 0.1. If the patroller sees a trail, he gets the utility 0.1. If he catches the evader, he gets the utility 1 and the game ends.

As in Transit game, this domain has to be zero-sum. Hence, a gain of the defender is a penalty of the evader and vice versa.

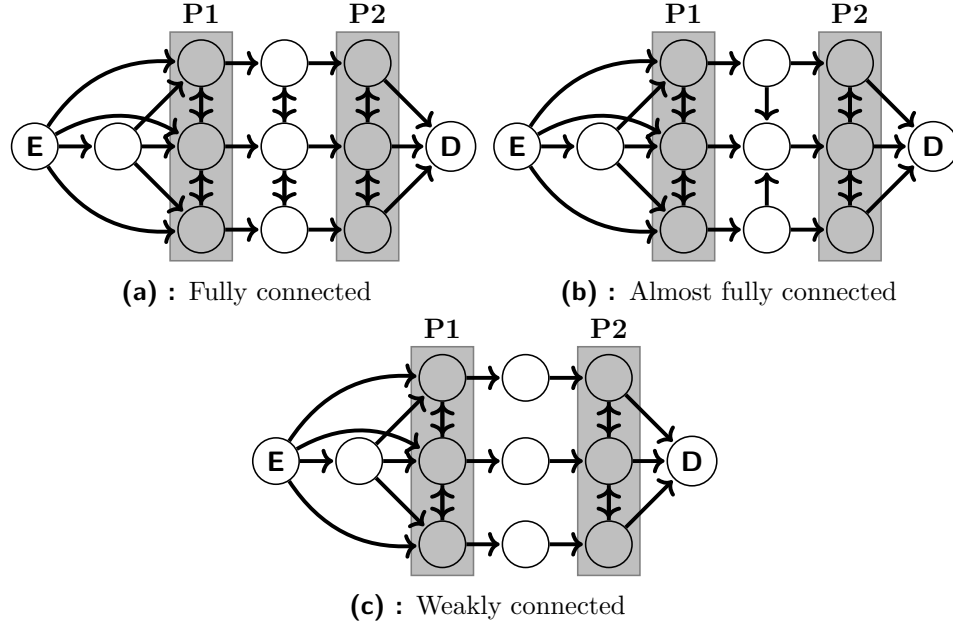


Figure 6.2: Graphs of Border protection game with width 3.

6.2.3 Ticket Inspection Game

Ticket inspection game was used for evaluation in [2]. It models scheduling ticket inspectors shift of specified length on a single train line. We define this line by a number of stops, a number of trains from both sides, and an approximate number of passengers. Trains are generated non-uniformly with tangent function. It means that at the beginning and at the end of the shift trains are generated with a lower frequency than in the peak around the second third of the shift. There are defined passenger trips as a number of passengers that take each train between each pair of stations. The trips are also generated non-uniformly from predefined popularities of stations, train timetable, and the approximate overall number of passengers.

The ticket inspector starts in the middle of the train line on a station. On a station, he can either take a train or check tickets of passengers on the station, which takes 15 minutes. When he is on a train, he can either check the tickets until the next stop or exit the train. The inspector checks 5 passengers per minute and passengers are present in the station 3 minutes before they take a train and 3 minutes after they exit it. The inspector misses the train or is not able to leave the train with the probability of 0.1. It is

due to unexpected delays. The MDP of ticket inspector starts with an action when he receives money for purchased tickets. Afterwards, the MDP contains a schedule of the shift of the inspector.

The MDP of passenger trips is drawn in Figure 6.3. It starts with the dummy action with a successor for each trip. The transition probability is defined as a probability, that random passenger takes the trip. In each of these successors, the passenger can decide, if he buys a ticket or dodges the fair.

The ticket price is 1.50\$ and fine for driving without a ticket is 100\$. After the inspector imposes a fine of the passengers in a train, he must impose a fine of the passengers in another train. Otherwise, he gets nothing from the passengers.

This game is clearly zero-sum because what a passenger pays, the inspector gets.

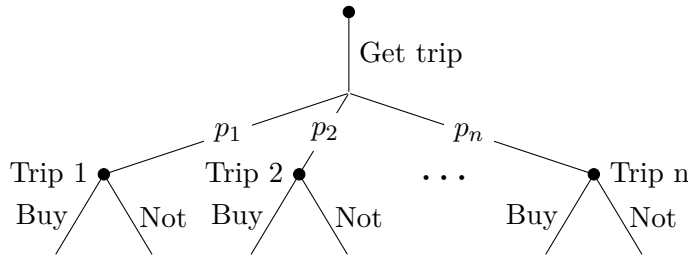


Figure 6.3: MDP of passenger in Ticket inspection game.

6.3 Nash Equilibrium in NFGSS Evaluated on Generalized NFGSS

NFGSS has several methods for finding Nash equilibria. In this section, we will discuss, if Nash equilibrium strategies in NFGSS give a good approximation of the Nash equilibrium in generalized NFGSS.

We solved NFGSS by CFR algorithm on the absolute precision 0.01 (a difference of the best response values), and we computed the best response values in generalized NFGSS on the found strategies. The precision ϵ guarantee that we found ϵ -Nash equilibrium in modified NFGSS.

In Border protection game, we tested games on the weakly connected graph with the width equal to 3. When the number of steps was smaller than 4, the strategies were quite close to Nash equilibrium in the generalized NFGSS. The difference values were at most 3.68% of the utility. The difference in generalized NFGSS was on the weakly connected graph with three steps sometimes even smaller than in ‘linear’ NFGSS ($0.34\% < 0.71\%$). When the number of steps was greater than or equal to 4, the strategies computed by CFR on ‘linear’ NFGSS differed considerably from Nash equilibrium in nonlinear NFGSS. The differences between the best response values ranged between 9% and 11%. The definition of nonlinearity in Border protection game cause this changes. The smuggler can reach the end of the plane after 4

steps, and it is the moment when the nonlinearity plays the significant role. It is almost inconsequential, if the player was caught twice or once, the strategy would be close to Nash equilibrium.

Steps	2	3	4	5	6	7
Precision [%]	3.68	0.34	10.98	9.58	9.66	9.07

Table 6.3: Nash equilibria from ‘linear’ approximation evaluated on generalized Border protection game on the weakly connected graph with width 3.

In Transit game, we tested games with many possible combinations of the width, the length and the number of steps. When the game configuration was not small (has more than two steps), the differences ranged from 1.3% to 2.1%. The multiple captures allowed in the ‘linear’ version of this game has a significant impact on the optimality in the strategy profile in the nonlinear version. It is caused by the penalty for step, the penalty for not returning to the basis and the height interconnection of the graph.

Length	2	3	4	5
Precision [%]	1.36	1.29	1.56	2.08

Table 6.4: Nash equilibria from ‘linear’ approximation evaluated on generalized Transit game with the width 2 and with the number of steps equal to the length + 1.

In Ticket inspection game, we tested games with the variable number of stations and the variable number of trains. The approximative number of passengers was always set on thousand times the number of trains. The results on this game domain differed a lot. Sometimes, the differences between the best response values take values around 0.05%. On the other hand, the differences acquire values up to 0.5% in some examples. In Ticket inspection game, the nonlinearity avoids checking passenger two times in the same train in a row. It is not a radical change, and usually, it has not a big impact on the best strategies. Sometimes, we cannot rely on linear NFGSS as a heuristic, and we must solve generalized NFGSS to get a reasonable result.

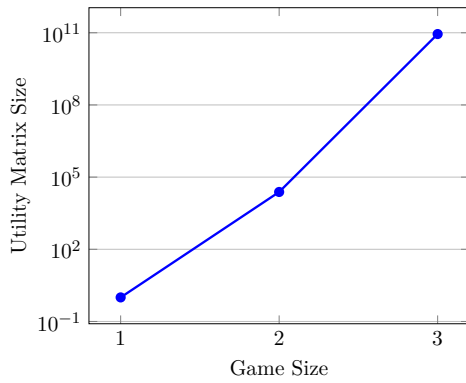
Trains	1	2	3	4
Precision [%]	0.048	0.48	0.05	0.51

Table 6.5: Nash equilibria from ‘linear’ approximation evaluated on generalized Ticket inspection game with 3-hour long shift, with the number of stations equal to the number of trains +1, and with the approximative number of passengers equal to the $1000 \cdot$ the number of trains.

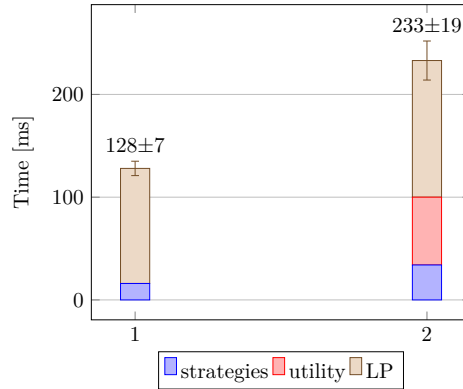
6.4 LP Solver

In this section, we describe experimental results for the standard LP for solving normal form games, which is outlined in Section 4.6.1. This algorithm is the baseline, and it is not expected to be usable for problems that are not negligibly small.

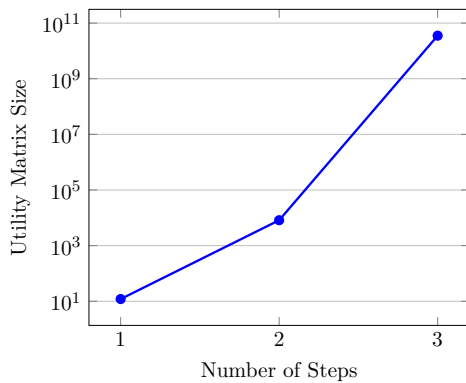
We used all three domains. Border protection game was run on the weakly connected graph with the width 2 and with the variable number of steps. Transit game had a size w , where the length of the grid was equal to the width and the size w , and the number of steps was equal to $w + 1$. Ticket inspection game was run on the three hours (180 min) long shift, with the variable number of stations, the number of trains was constantly 2, and the approximate number of passengers was 2000. We ran the experiments five times to decrease the statistical error. The matrix size is always the same, and the standard deviations of times are in graphs.



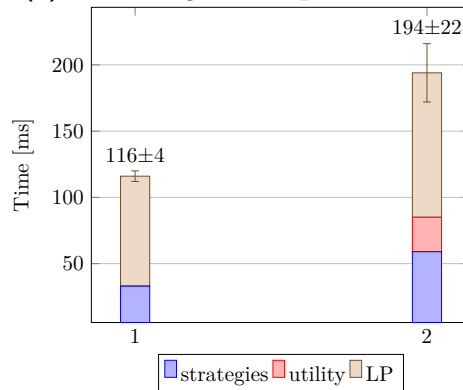
(a) : Transit game utility matrix size



(b) : Transit game computation time



(c) : Border protection game utility matrix size



(d) : Border protection game computation time

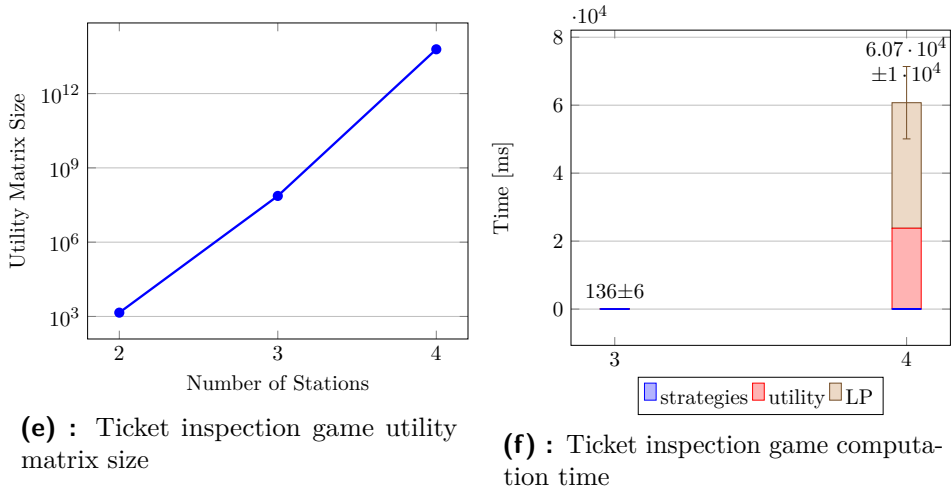


Figure 6.4: Sizes of utility matrices and computation times for Transit game (a, b), Border protection game (c, d), and Ticket inspection game (e, f). In the time analysis chart, the labels mean: ‘strategies’ - the part of getting all strategies, ‘utility’ - the part of getting utility, and ‘LP’ - the part of solving LP.

In the graphs, we can see, that the size of the payoff matrix grows quickly, which was expected. It corresponds to the time necessary for the computation. TG with the size 3 and in TIG with four stations has huge memory requirements, and we have not enough resources to meet it. In the TG of size 3, the utility matrix has size 338339×261307 . It means that we need more than $8.841 \times 10^{10} \cdot 8 \approx 7.073 \times 10^{11}$ bytes, which is approximately 707 Gigabytes. In the TIG with four stations, the utility matrix has size 3665243×16777216 . Thus, it requires even more memory (≈ 492 Terabytes). The payoff matrix in the BPG with three steps has size 20×1761607680 , and we have not enough resources to store all strategies.

6.5 Analysis of MCCFR for Generalized NFGSS

This section describes the experiments, which were performed with the MCCFR algorithm on modified NFGSS games. We examined the progress in the strategy, and we tried to find out, whether the algorithm converges.

We ran the algorithm on all three domains with different sizes. The computation was limited by a maximal number of iterations equals to 10^9 . Based on the previous experiments, we set the epsilon from the epsilon-on-policy to 0.1 or 10%. And the precision of the final result was set to 0.01, which means the best response values on strategies of players differ maximally by 0.01. In TG it corresponds to the half of the penalty for a step, in BPG it is a tenth of the penalty for observing a trail and in TIG the error tally with one cent per passenger.

6.5.1 Border Protection Game

Border protection game was used in all three variants: weakly connected, almost fully connected, and fully connected. It was run with width three and a multiple number of steps. There are three examples of convergence graphs. It shows the average values of five runs. The standard deviation was in all iterations and in all tests smaller than 0.03.

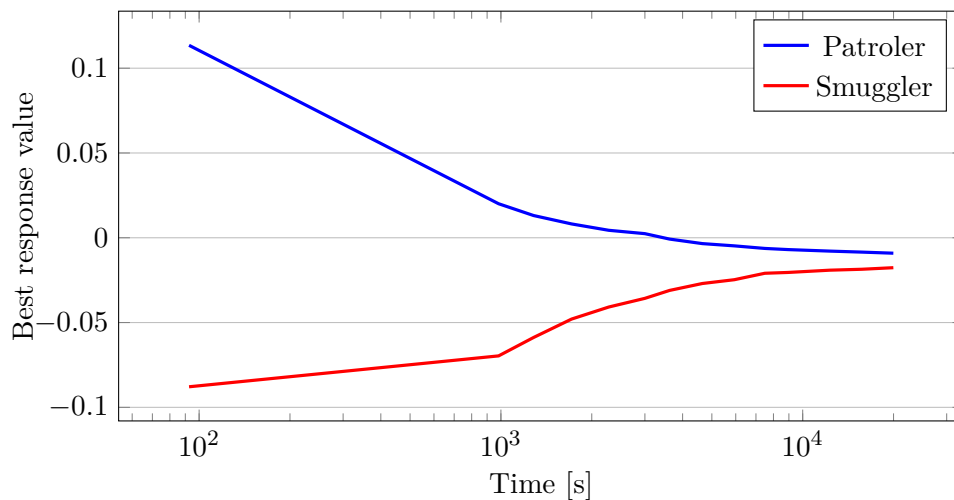


Figure 6.5: Convergence chart of the BPG with the weakly connected graph, width 3, and 4 steps.

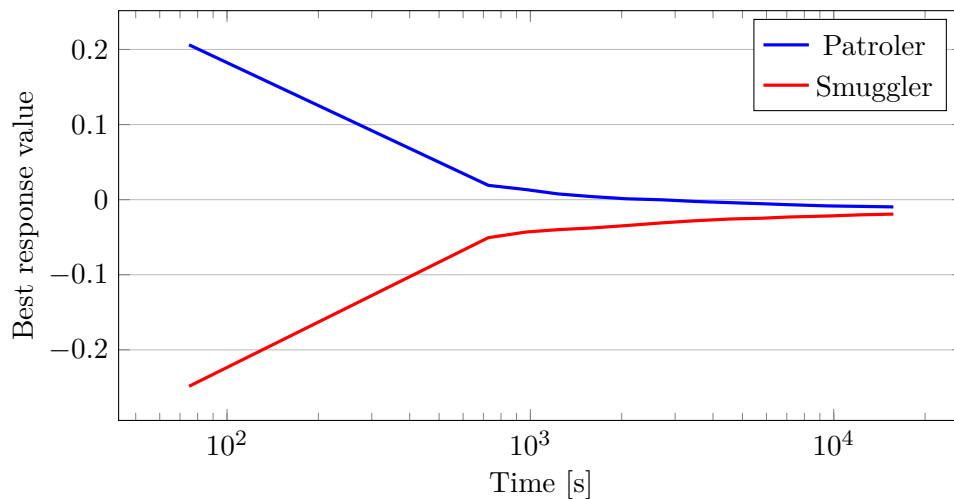


Figure 6.6: Convergence chart of the BPG with the almost fully connected graph, width 3, and 4 steps.

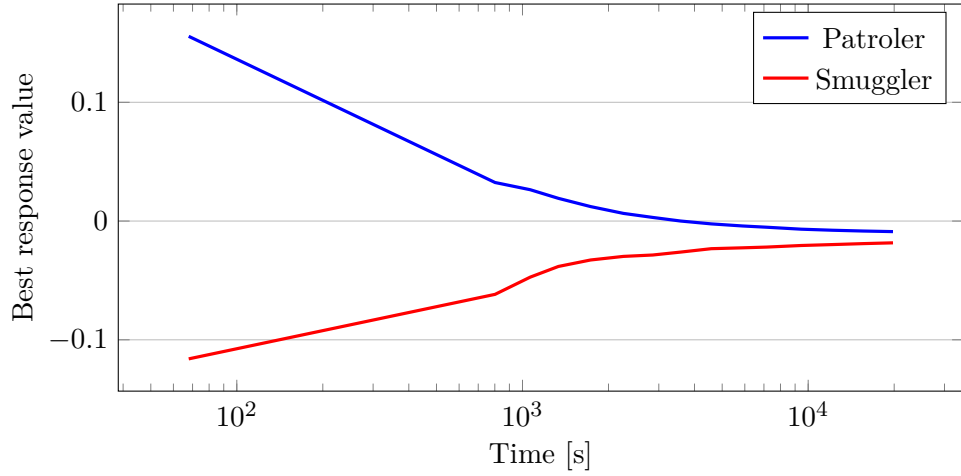


Figure 6.7: Convergence chart of the BPG with the fully connected graph, width 3, and 4 steps.

In the graphs and also in all other tested Border protection games the best response values for both players apparently converges to the same values, which means that strategies found by the MCCFR algorithms was in ϵ -Nash equilibrium for $\epsilon \rightarrow 0$. In these three examples, the algorithm converges to the desired precision (0.01) after ca. 5.5 hours (2.8×10^6 iterations), 4.4 hours (4.8×10^5 iterations), and 5.5 hours (3×10^6 iterations) respectively.

When we compare the MCCFR algorithm with the approximation by ‘linear’ NFGSS, MCCFR gives us better results. The smallest difference gained by the approximation was 9.07% (unless we include the smallest games), while MCCFR gives us the precision, which we want to.

Another question is the scalability of the algorithm in this game. All experiments were run five times, the average values and the standard deviations are presented. The percentages differed not more than by 4%.

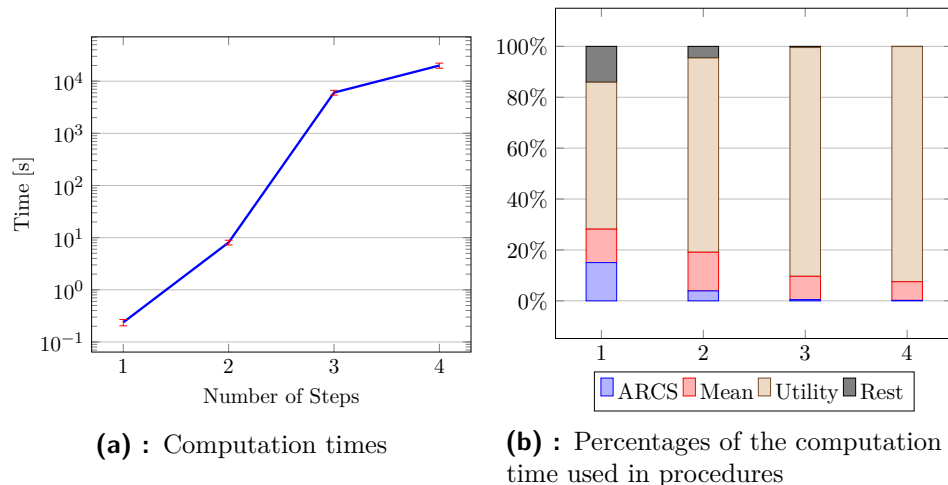


Figure 6.8: Scalability of MCCFR in Border protection game with weakly connected graph

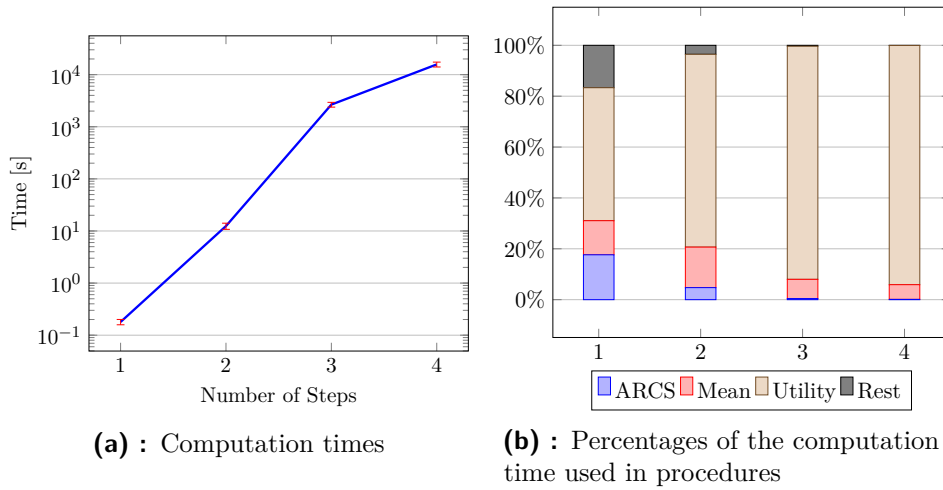


Figure 6.9: Scalability of MCCFR in Border protection game with almost fully connected graph

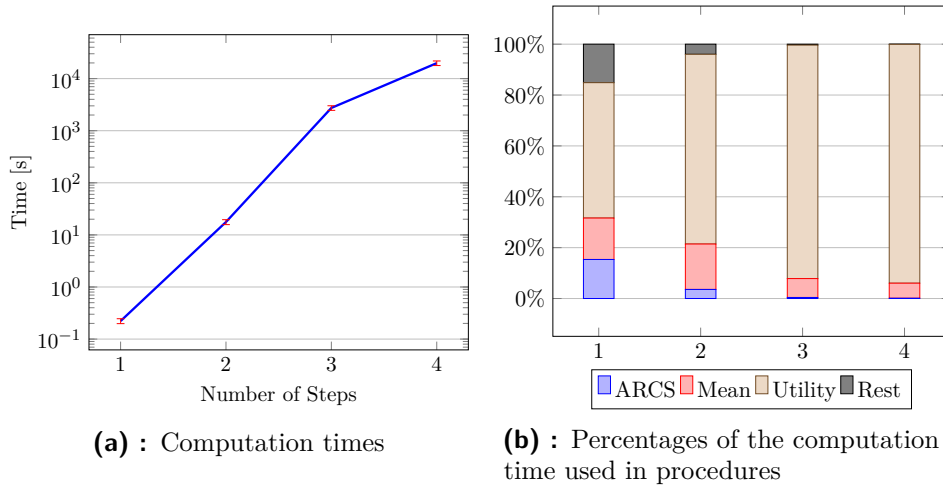


Figure 6.10: Scalability of MCCFR in Border protection game with fully connected graph

The previous graphs shows the computation times and percentages of time used in methods for Border protection game with the weakly connected graph (6.8), Border protection game with the almost fully connected graph (6.9), Border protection game with the fully connected graph (6.10). In the method analysis chart, the labels mean:

- ‘ARCS’ - the method of computation action regrets and current strategies (without computeExpectedUtility)
- ‘Mean’ - the method for updating of the state probabilities and computing mean strategies
- ‘Utility’ - the function computeExpectedUtility
- ‘Rest’ - overhead time for printing results and so on

The time needed to solve the game grows quickly. The first factor, which plays a role in convergence time, is the number of iterations necessary for the solving the game. This number is connected with the number of samples through both MDPs. In the CFR algorithm, we update values in all MDP in one iteration. Unlike it, MCCFR updates strategies only in one sample in an iteration. Therefore, we need the number of iterations from CFR multiplied by the number of samples in the MDP. This estimate is very inaccurate because it profoundly depends on the probability of choosing the sample. The number of iterations needed to solve the game grows typically with the size of the game, but it is not the rule. For example, we needed approximately 6.7×10^6 iterations in Border patrolling game with the weakly connected graph with 3 steps and 2.3×10^6 iterations in the same game with 4 steps. It is caused by not uniformity in selecting a sample.

The reason for the bad scalability can be seen in right graphs. The main part of the computation time is spent in the `computeExpectedUtility` function. In this function, we evaluate a chosen sample against all the MDP of the opponent. We go through the MDP and evaluate the states against the states in the sample. When the probability of reaching the successor is zero, we drop out all continuations of this path. It is also done when the game has to end (the defender catches the attacker).

6.5.2 Ticket inspection game

We run Ticket inspection game with a various number of stations and trains. The approximative number of stations was set on thousand times the number of trains. The following convergence graph shows an example of the average of five runs. The error was always smaller than 1.5%.

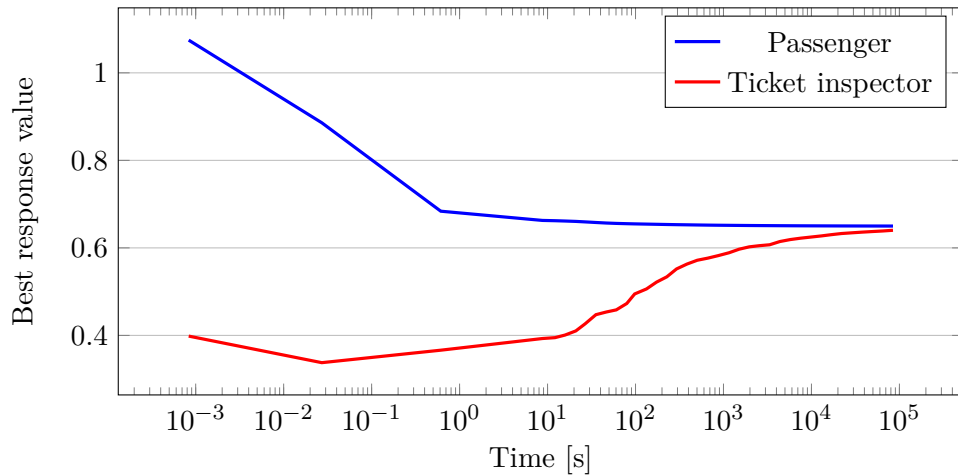


Figure 6.11: Convergence chart of the TIG with two stations, two trains, and approximative 2000 passengers.

The graph and the other tested configurations of this game indicate that the mean strategy computed by MCCFR algorithm converges to the Nash

equilibrium also in Ticket inspection game. More specifically, in the previous example, the mean strategy reaches desired accuracy (0.01) after about 23.5 hours (784×10^6 iterations).

When we compare the MCCFR algorithm with the approximation by ‘linear’ NFGSS, MCCFR gives us better results. However, the results given by ‘linear’ approximation are sometimes quite good (the difference of best responses is about 0.05%) and the computation time is significantly lower ($10^{-2}s < 10^1s$). The problem is, we do not know in which case the approximation give us the good result, while MCCFR gives us always the precision, which we want to.

Another question is the scalability of the algorithm in this domain. All experiments were run five times, the average values and the standard deviations are presented. The percentages differed not more than by 4%.

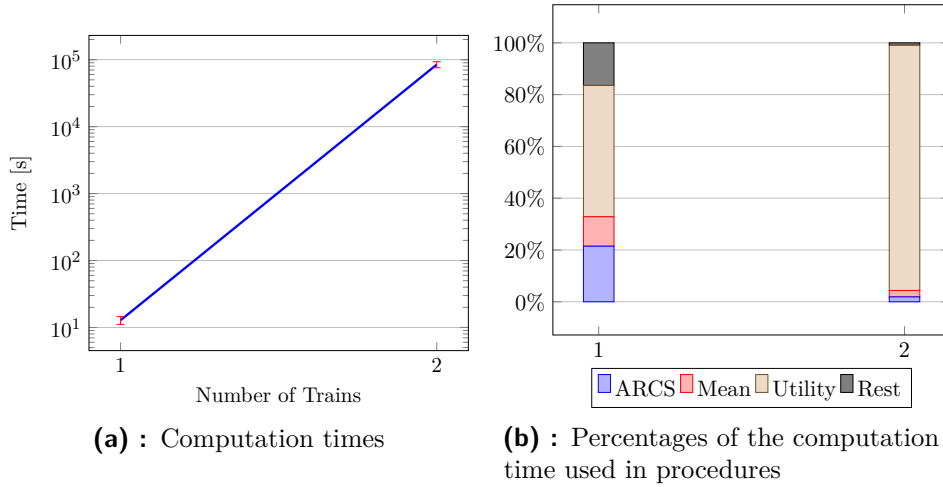


Figure 6.12: Scalability of MCCFR in Ticket inspection game

Ticket inspection game was in the previous figure used with the changing number of trains, the number of stations was equal to the number of trains plus one, and the approximative number of passengers was equal to the number of trains multiplied by thousand.

The graph of the scalability shows the same phenomena, which were discussed in the previous section. However, the computation of the utility in Ticket inspection game is more time complex than in Border protection game because the nonlinearity does not drop out any paths through MDP. It only skips some values.

6.5.3 Transit game

Transit game has three degrees of freedom: width, height, and a number of steps. We ran multiple tests With various settings. Data displayed in following charts are averages of five runs with different random seeds. The error was always smaller than 1%. The run games are marked as follows: TG a number of steps, a length, a width. For example, TG 2 1 3 means Transit game with two steps and on a grid with width 3 and length 1.

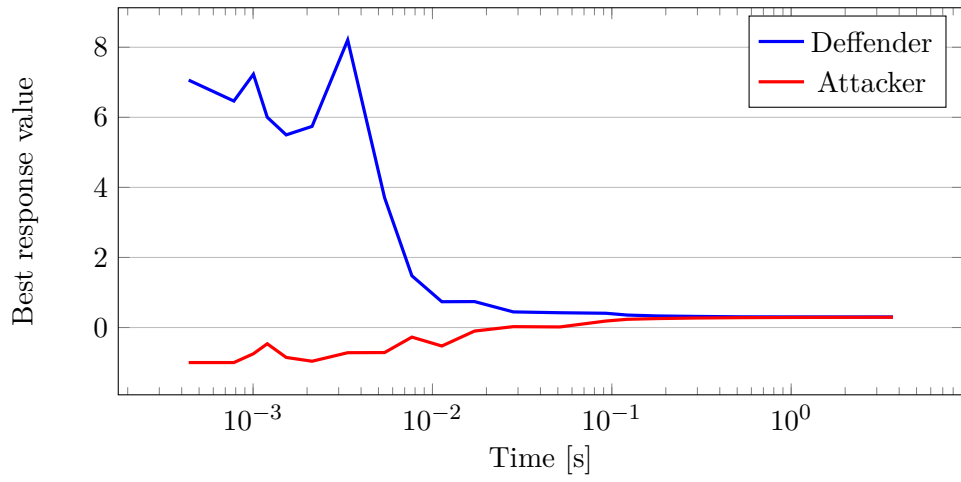


Figure 6.13: Convergence chart of the TG 2 2 2.

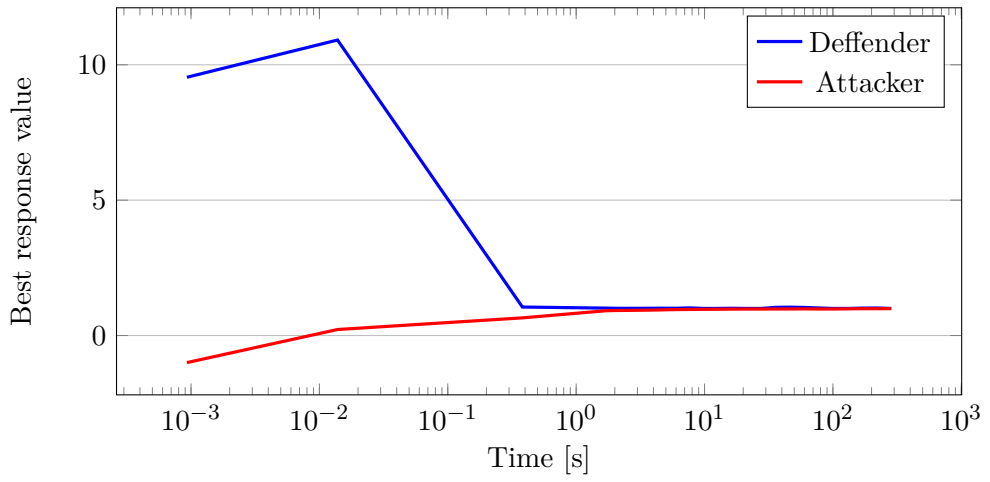


Figure 6.14: Convergence chart of the TG 3 2 3.

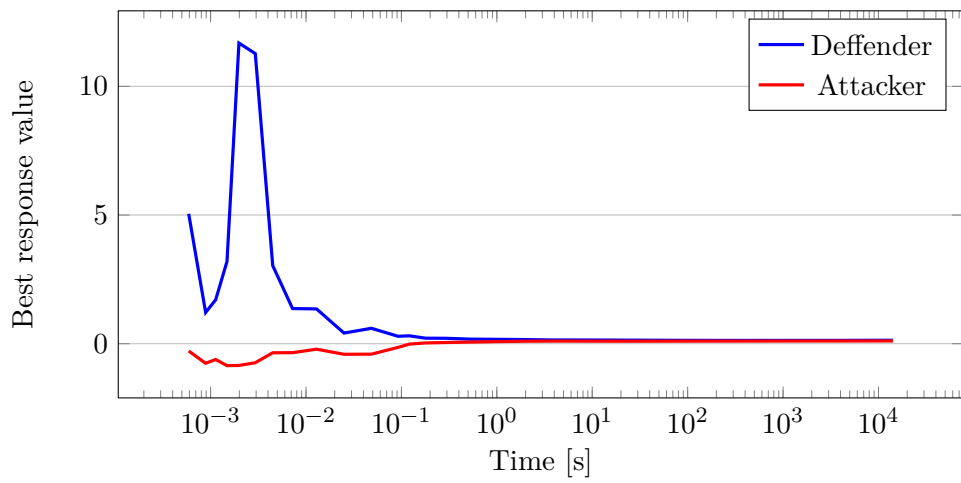


Figure 6.15: Convergence chart of the TG 3 2 2

The first two examples converge to the accuracy 0.01 respectively after ca. 10^6 iterations and 4 seconds, ca. 24×10^6 iterations and 4.75 minutes.

On the third graph, there is drawn a progress of best response values against the mean strategies of both players. It seems that it converges quickly as the previous examples. However, the required precision is not reached even after 10^9 iterations, which was set as a limit. When we look at it in detail (Figure 6.16), we can see that the accuracy oscillates around the value of 0.11%. The best-attained strategy has the error equal to 0.106%. It was reached after ca. 45 minutes and 262×10^6 iterations.

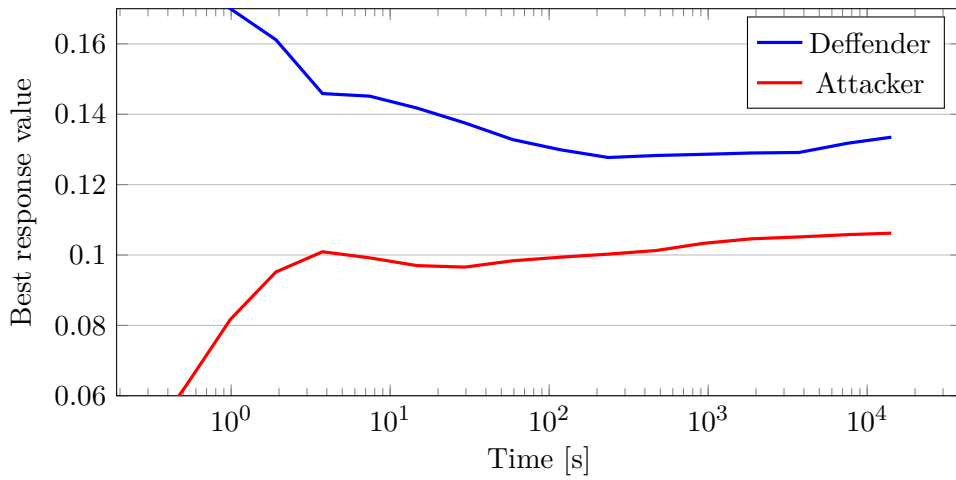


Figure 6.16: Convergence chart of the TG 3 2 2 in the detail.

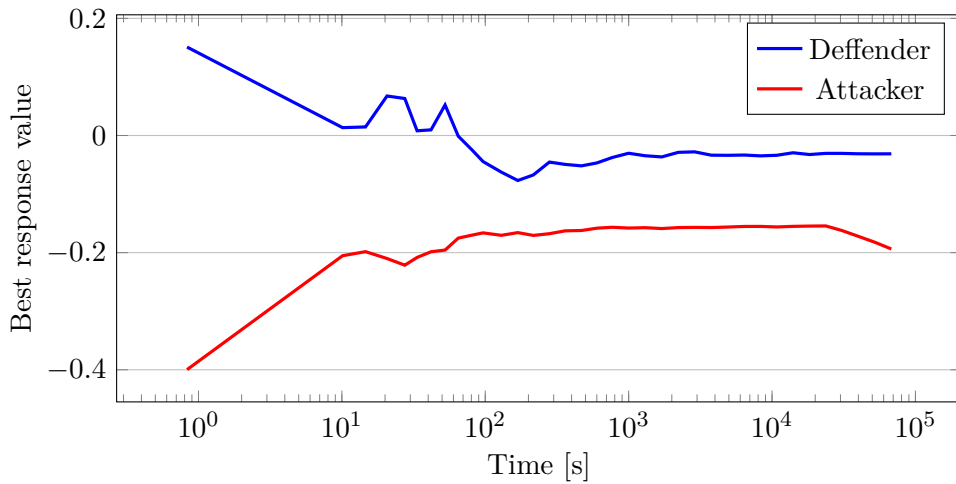


Figure 6.17: Convergence chart of the TG 4 3 2

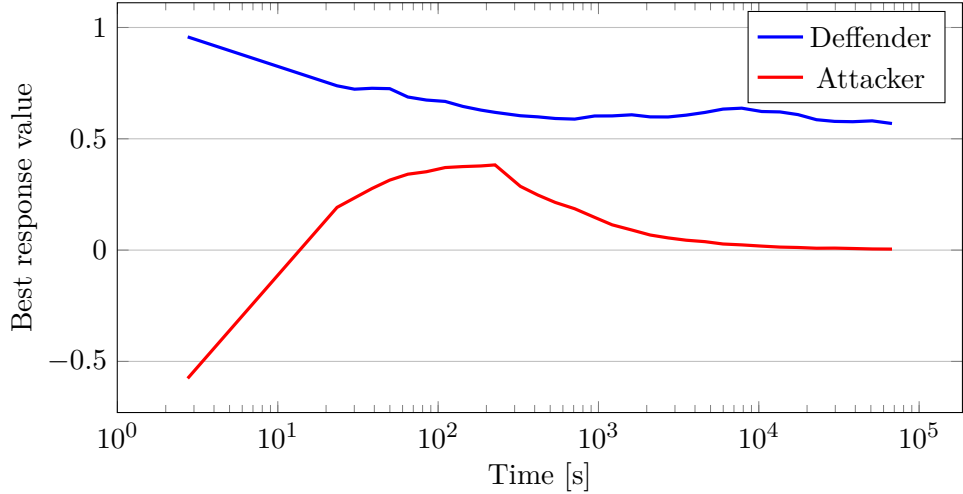


Figure 6.18: Convergence chart of the TG 4 3 3

In the last two examples, it is even more clear that the MCCFR algorithm does not converge to Nash equilibrium. TG 4 3 2 reaches the minimal error equal to 0.42% after approximately 2.4×10^6 iterations and 217 seconds. TG 4 3 3 has the minimal difference even worse. It is equal to 1.2%, and it is reached after ca. 0.9×10^6 iterations and 184 seconds.

The main question is, what causes this divergence. We have measured, that in these examples, there are differences in values gained by the same action according to a history and this difference does not disappear with the time.

The difference can in the nonlinear version of Transit game appear when the attacker has two histories h_1 , h_2 , which lead to the same information node. In h_1 the player is caught with the probability of ϵ , and in h_2 he can not be caught. Therefore, an action a is after the history h_1 evaluated only against a part of the MDP of the opponent. By way of contrast, the same action a is after history h_2 evaluated against all the MDP of the opponent.

We show this phenomenon in the following example. We have a nonlinear Transit game, where the attacker gets only the utility 1 for reaching the most right node and the utility -1 when he is caught. We play this game on a 3×3 grid, with three steps, and there is no uncertainty. The parts of the MDPs of both players, where the strategy probability is nonzero, are in Figure 6.19. In the figure, strategy probabilities are drawn next to actions, and there is marked an action a .

When the attacker plays the first history (probability p), he is caught with the probability of ϵ at the time 1, and he is caught with the probability of $1 - \epsilon$ at the time 2. When the player plays the second history (probability $1-p$), he is caught with the probability of $1 - \epsilon$, and with the probability of ϵ , he reaches the right node (1,2) safely. The utility gained by a after the first history is -1 and after the second history, it is $\epsilon \cdot 1 + (1 - \epsilon) \cdot (-1)$. When ϵ is bigger than 0, these utilities are unequal.

MCCFR algorithm does not converge in Transit game. We can use the strategy profile, which is nearest to Nash equilibrium, as an approximation.

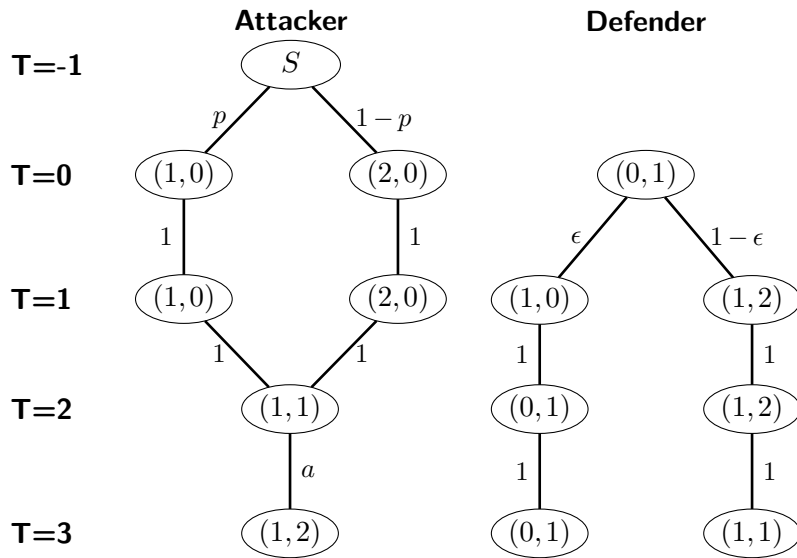


Figure 6.19: An example of a situation in Transit game, where utility gained by action a depends on the previous history.

Then we can compare this estimate against the ‘linear’ approximation by NFGSS. MCCFR is always better in this comparison. For example, TG 3 2 2 has the best strategy profile with the error 0.106% and the same instance gets from approximation by NFGSS the error 1.36%. TG 4 3 2 has the best difference gained by MCCFR equal to 0.42% and by CFR in NFGSS get the precision 1.29%. Finally, TG 4 3 3, which is the worst case for MCCFR, has the difference by MCCFR equal to 1.2% and the ‘linear’ approximation gives us 3.7%, which is also the biggest error from all tested instances.

The last question is the scalability of the algorithm on this domain. Since MCCFR does not converge in this domain, we stopped the computation after 10^9 iterations. All experiments were run five times, the average values and the standard deviations are presented. The percentages differed not more than by 4%.

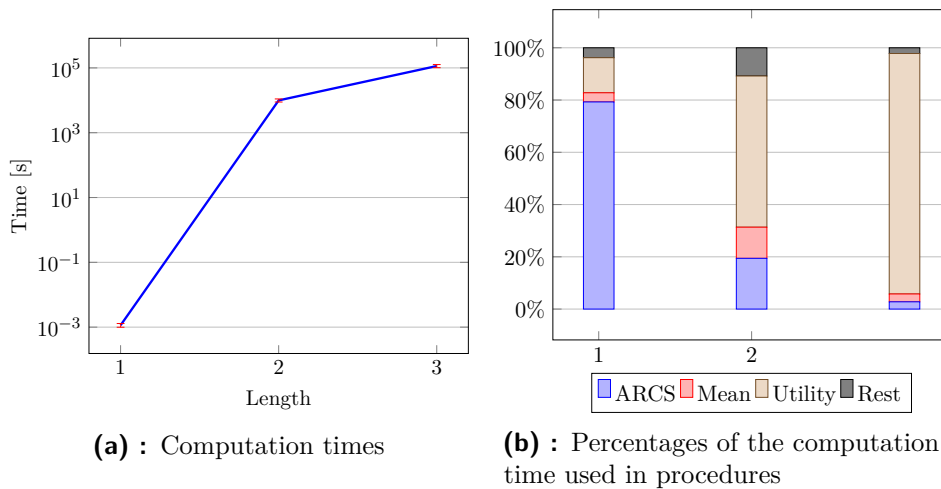



Figure 6.20: Scalability of MCCFR in Transit game

Transit game confirms, that the time complexity grows quickly with the size of the game. From the right graph, we can see that Transit game has the `computeExpectedUtility` function less time-consuming. It is because we need to evaluate the state from the MDP against only the state with the same timestamp from the sample.



Chapter 7

Conclusions

The goal of the work was to solve generalized NFGSS in the meaning of finding Nash equilibria in them. As the baseline algorithm, we used the standard LP for solving normal-form games. It is possible because each NFGSS can be expressed as an exponentially large game in the normal form.

The second approach to solving generalized NFGSS used no-regret learning algorithms. We had started with the Counterfactual regret minimization (CFR) algorithm adapted for NFGSS. However, it could not be used on generalized NFGSS because it needs to know the utility of single actions. This is not feasible in generalized NFGSS since the utility function there is defined only for terminal histories. Therefore, we adapted the Monte Carlo Counterfactual regret minimization (MCCFR) algorithm for NFGSS. This algorithm needs to know the utility of completed games, which in our implementation corresponds with terminal histories in NFGSS. It implies that we could use it directly on generalized NFGSS.

For testing, we modeled three domains of generalized NFGSS: Transit game, Ticket inspection game, and Border protection game. On these domains, we run multiple experiments with the LP solver and MCCFR. Finally, we compared the attained results with results given by evaluation of Nash equilibria from NFGSS in generalized NFGSS.

MCCFR algorithm always converged in Ticket inspection game and Border protection game. In Transit game the algorithm diverges since we discovered considerable utility dependency on the previous history. However, the best strategy found by MCCFR was a far better than results of the ‘linear’ approximation of generalized NFGSS.

The scalability of LP method was bad, because of the exponential size of the utility matrix of the representation in the normal form. MCCFR algorithm managed to compute more problems, but the largest solved problems had only tens or at most hundred of states in one MDP. The bad scalability is caused by getting utility function, which is called in each iteration, and the computation time of it grows with the number of paths through MDP.

7.1 Future work

MCCFR proved to be promising as a solver or an approximative solver of generalized NFGSS. We discovered a significant deficiency in scalability. The current bottleneck is the computation of the expected utility. It would be great to speed up this method by, for example, finding a more efficient algorithm or by parallelization of the computation. The other possibility is to find some approximation of this function, which would speed up the computation without significant inaccuracy. For example, we can drop out the samples with reaching probability smaller than $\epsilon \geq 0$. (In actual version $\epsilon = 0$).

Furthermore, for specific domains, we would like to identify some more heuristics to improve the scalability of the algorithm.



Appendix A

Bibliography

- [1] Y. Shoham and K. Leyton-Brown, *Multiagent Systems : Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, rev 1.1 ed., 2009.
- [2] V. Lisý, T. Davis, and M. Bowling, “Counterfactual regret minimization in sequential security games,” in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
- [3] J. Nash, “Non-cooperative games,” *Annals of Mathematics*, vol. 54, pp. 286–295, Sept. 1951.
- [4] M. Lanctot, *Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision Making in Large Extensive-Form Games*. PhD thesis, University of Alberta, 2013.
- [5] B. Bošanský, A. X. Jiang, M. Tambe, and C. Kiekintveld, “Combining compact representation and incremental generation in large games with sequential strategies,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [6] “Vo metacentrum - physical machines.” <https://metavo.metacentrum.cz/pbsmon2/nodes/physical>. Accessed: 2017-04-22.

Appendix B

CD Content

The enclosed CD contains following files and directories:

- **silhapro.pdf** - the text of this work
- **source** - folder with the implementations in Java

Running the Implementations

The source folder contains the whole GTLibrary supplemented by our source code of MCCFR algorithm, transformation NFGSS to the normal form and solving the standard LP, and implementations of nonlinear domains: Transit game, Border protection game, and Ticket inspection game.

Executing of it requires third party software. All of them can be downloaded for free (or at least in an academical license):

- **Java**¹
- **IBM ILOG Cplex**²

Our code is located in following packages:

- **cz.agents.gtlibrary.nfg.MDP.mccfr** - the whole package
- **cz.agents.gtlibrary.nfg.MDP.exponential** - the whole package
- **cz.agents.gtlibrary.nfg.MDP.implementations** - the abstract class MDPConfigNonlinearImpl
- **cz.agents.gtlibrary.nfg.MDP.domain.transitgame** - the TGConfig supplemented by extension of the MDPConfigNonlinearImpl class
- **cz.agents.gtlibrary.nfg.MDP.domain.bpg** - the BPCConfig supplemented by extension of the MDPConfigNonlinearImpl class
- **cz.agents.gtlibrary.nfg.MDP.domain.tig** - the TIGConfig supplemented by extension of the MDPConfigNonlinearImpl class

¹<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

²<https://developer.ibm.com/academic/>

The standard LP solver can be executed by running `MDPExponentialFormAlgorithm` from package `cz.agents.gtlibrary.nfg.MDP.exponential` with following options:

[domain] [parameters of the domain] [*Boolean*: use nonlinear utility]

The MCCFR algorithm can be executed by running `MCCFR` from package `cz.agents.gtlibrary.nfg.MDP.mccfr` with following options:

[domain] [parameters of the domain] [*Boolean*: use nonlinear utility] [*Double*: final precision] [*Double from 0 to 1*: ϵ from epsilon-on-policy] [*Integer*: a maximal number of iterations] [*Boolean*: use CFR] [*Boolean*: evaluate on nonlinear]

The ‘use CFR’ and ‘evaluate on nonlinear’ options are available only with the linear utility. With the ‘use CFR’, is Nash equilibrium found by CFR, and the ‘evaluate on nonlinear’ computes the best response values in generalized NFGSS against the found strategy in NFGSS.

Possible domains with options are:

[TG] [*Integer*: a number of steps] [*Integer*: length] [*Integer*: width]

[BP] [*Integer*: a number of steps] [*String*: graph file]

[TIG] [*Integer*: max time] [*Integer*: number of stops] [*Integer*: number of trains] [*Integer*: approximative number of passengers]

BACHELOR PROJECT ASSIGNMENT

Student: Prokop Šilhavý

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: No-regret Learning in Generalized Normal-Form Games with Sequential Strategies

Guidelines:

In some sequential games, players have limited observations about the actions of the opponent. These games can be modeled as normal-form games with sequential strategies (NFGSS), where the strategy space of each player is modeled as a finite acyclic Markov decision process. Existing algorithms for finding optimal strategies in these games typically assume a specific linear structure of utilities that allow these games to be solvable by linear programming or no-regret learning algorithms. However, such an assumption is not always present in real-world scenarios. The main goal of the student is to (1) analyze these more general NFGSS and no-regret learning algorithms based on counterfactual regret minimization (CFR), (2) analyze the possibility of implementing CFR-based algorithms for these more general NFGSS despite the lack of theoretical guarantees, and (3) provide an experimental evaluation of the quality of CFR strategies in at least three different games.

Bibliography/Sources:

- [1] Branislav Bosansky, Albert Xin Jiang, Milind Tambe, and Christopher Kiekintveld. Combining Compact Representation and Incremental Generation in Large Games with Sequential Strategies. In Proceedings of AAAI. 2015
- [2] Viliam Lisy, Trevor Davis, Michael Bowling. Counterfactual Regret Minimization in Sequential Security Games. In Proceedings of AAAI. 2016
- [3] Marc Lanctot. Monte Carlo Sampling and Regret Minimization For Equilibrium Computation and Decision-Making in Large Extensive Form Games. PhD Thesis, University of Alberta, 2013

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

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 12, 2017