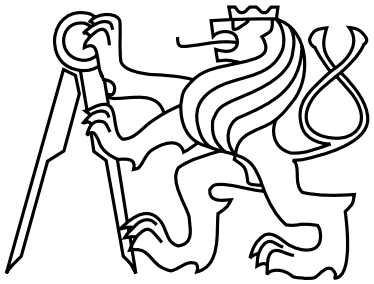
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics



Bachelor's Project

Stackelberg Equilibria in Normal Form Games with Sequential Strategies

Eduard Rindt

Supervisor: Mgr. Branislav Bošanský Ph.D.

Study Programme: Otevřená Informatika

Field of Study: Informatika a počítačové vědy

May 21, 2015

Prohlášení

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

V Praze dne 22. 5. 2015 .....

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Eduard R i n d t

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Stackelbergovo equilibrium v normálních hrách se sekvenčními strategiemi

Pokyny pro vypracování:

V některých herně-teoretických modelech mají hráči pouze omezenou možnost pozorovat akce oponenta. Tyto hry tak můžeme modelovat jako standardní hry v normální formě, kde prostor strategií každého hráče odpovídá konečnému acyklickému markovskému rozhodovacímu procesu. Existující algoritmy pro tuto třídu her typicky předpokládají, že cíle hráčů jsou striktně kompetitivní (tj. jedná se o hru s nulovým součtem).

Úlohou studenta je relaxace tohoto předpokladu, analýza algoritmů pro výpočet tzv. Stackelbergova equilibria v sekvenčních hrách s nenulovým součtem a adaptace těchto algoritmů pro tuto třídu her.
Druhým cílem práce je prozkoumání možností iterativních metod pro výpočet Stackelbergova equilibria s cílem zrychlit výpočet equilibria.

Seznam odborné literatury:

[1] Branislav Bosansky and Jiri Cermak: Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In Proceedings of AAAI (to appear). 2015.
[2] 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(to appear). 2015
[3] Zhengyu Yin and Milind Tambe: A unified method for handling discrete and continuous uncertainty in Bayesian Stackelberg games. In Proceedings of AAMAS. 2012.
[4] Yoav Shoham, and Kevin LeytonBrown: Multiagent systems: Algorithmic, gametheoretic, and logical foundations. Cambridge University Press, 2009.

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

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 14. 1. 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Eduard R i n d t

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Stackelberg Equilibria in 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, 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 the game is strictly competitive (i.e., zero-sum).

The goal of the student is to relax this restriction, analyze the existing algorithms for computing Stackelberg equilibria in general-sum extensive-form games, and adapt these algorithms for normal-form games with sequential strategies.
The second goal is to investigate the possibility for using iterative methods to improve the computation time of the algorithm.

Bibliography/Sources:

[1] Branislav Bosansky and Jiri Cermak: Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games. In Proceedings of AAAI (to appear). 2015.
[2] 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(to appear). 2015
[3] Zhengyu Yin and Milind Tambe: A unified method for handling discrete and continuous uncertainty in Bayesian Stackelberg games. In Proceedings of AAMAS. 2012.
[4] Yoav Shoham, and Kevin LeytonBrown: Multiagent systems: Algorithmic, gametheoretic, and logical foundations. Cambridge University Press, 2009.

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

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

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 14, 2015

Abstract

In Stackelberg games, one agent must commit to a strategy before the other agents compute their strategies, allowing them to always play the best response to first player's strategy. Standard models of computing optimal strategy of the first player in sequential games are challenging to compute. In this work, we will focus on Stackelberg games and their solution in normal-form games with sequential strategies, which is a simplified form of a sequential games, in which the players have no information about actions of their opponents throughout the game.

Keywords: Game Theory, Stackelberg games, Normal Form games with sequential strategies

Abstrakt

Ve Stackelbergových hrách se musí jeden z agentů přihlásit ke své strategii ještě před tím, než ostatní hráči spočítají své strategie. To jim dává možnost hrát svou nejlepší strategii proti prvnímu hráči. Standartní modely výpočtu optimální strategie prvního hráče v sekvenčních hrách jsou výpočetně náročné. V této práci se zaměříme na Stackelbergovy hry a jejich řešení ve hrách v normální formě se sekvenčními strategiemi, což je zjednodušená forma sekvenční hry, v níž žádný z hráčů nemá v průběhu hry žádné informace o akcích svých protihráčů.

Klíčová slova: Teorie her, Stackelbergovy hry, hry v normální formě se sekvenčními strategiemi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Stackelberg games have been recently given a lot of focus, mostly because their importance for security models, where the leader is trying to find the optimal strategy to secure an object from possible attackers. In this case, the leader is defender of the object, who will commit to a strategy (e.g. a schedule of patrol etc.). This strategy will be observed by the attacker (or follower), who will then plan an attack as a best response to the observed defensive strategy. Stackelberg games are used in many security applications (e.g. IRIS for Federal Air Marshals Service [8], GUARDS for the Transportation Security Administration [9], ARMOR at Los Angeles International Airport [8])

For normal-form Stackelberg games (in which every player plays 1 action) there is an efficient algorithm solving this problem [1][3]. There also exists an algorithm for Stackelberg extensive-form games (exploiting compact sequence-form representation) [5]. This algorithm, however, does not scale well into large games, as computing Stackelberg equilibrium is an NP-hard problem in extensive-form games[11].

Normal-form game with sequential strategies [6] is a simplified extensive form game, in which states of the game are not represented by one game tree common to all agents. Instead, there is one Markov decision process for every agent participating in the game. That allows us to represent situations, where an agent does not have any information about movements of other agents, until the game reaches a terminal state, or he chooses to disregard this information. Also, normal-form games with sequential strategies do not need perfect recall (perfect memory about what an agent has played so far) in the game.

There is an existing algorithm solving Stackelberg normal-form games with sequential strategies (LP for NFGSS [6]), but it requires a strictly competitive game (or a zero-sum game). Therefore, the main goal of this thesis is to provide a new algorithm finding Stackelberg equilibrium in general-sum normal-form games with sequential strategies and then investigate the possibility of using iterative algorithms for solving this class of games.

1.1 Outline

In the second chapter of this work, we will describe the basics about game theory. It is necessary for understanding the main goal of this work. We will describe outcomes of the game and their comparison, strategies of players and the concepts of Nash and Stackelberg

equilibria using those strategies. Then we will describe several ways to classify a game and existing algorithms for solving different types of Stackelberg games, on which will we then focus.

In Chapter 3, we will first describe normal-form games with sequential strategies in more detail. We will then focus on the problem of finding strong Stackelberg equilibrium in general-sum normal-form game with sequential strategies. That can be done by formulating the problem as a set of linear programs [1] by transforming the game into a normal form game, and by solving those LPs, we would find a solution. This approach is probably the easiest one for implementation, but it doesn't scale well to large games, as there will be an exponential number of linear programs based on length of strategies in sequential games. Once we will formulate the problem as set of linear programs, we can transform these programs into a mixed integer linear program (MILP) representation [3] (similarly to solving Stackelberg extensive form games [5]), which will also find strong Stackelberg equilibrium in general-sum normal-form games with sequential strategies. This MILP will consist of set of constraints with linear size depending on number of actions and states of all agents.

We will then review the iterative algorithm solving this class of games[6]. The algorithm iteratively searches the space of player's strategies and prunes as many branches as possible. For that, it needs a fast algorithm for computing a tight upper bound on the expected utility of the leader. We will therefore present two variants of a simplified LP created from the MILP algorithm by relaxing some constraints. Given a sub-game of Stackelberg normal-form game with sequential strategies (NFGSS), these upper bound algorithms yield a reasonable upper bound of leader's (agent committing himself to a strategy) utility for this sub-game.

Chapter 2

Game theory

Game theory is a mathematical discipline describing interaction of two or more agents solving a given problem (for the rest of the work reffered to as the game) [4]. We will focus on non-cooperative game theory, which means that every agent (often referred to as player) is only concerned about his outcome. The outcome of the game is typically modeled as a *utility function*.

2.1 Utility and utility function

Formally, utility for a player p is any complete transitive relation $\succeq$ over outcomes of the game such that these outcomes can be partially ordered using $\succeq$ [4]. For two outcomes a, b and relation $\succeq$, we say that:

- p prefers outcome a to outcome b, if $a \succeq b$ and not $b \succeq a$
- p is indifferent between outcomes a, b, if $a \succeq b$ and $b \succeq a$

For the rest of this work, we will use only following concept. A utility function $u_p : O \mapsto \mathbb{R}$ (where O is the set of possible outcomes of the game) evaluates every outcome for player p with a real number. Player p then measures his utility using relation $\geq$ on these numbers.

In non-cooperative game-theory, player p seeks to maximize his utility, that means to reach such an outcome a, for which no reachable outcome b is strictly preferred to a. In terms of a utility function, it means to reach a reachable outcome with the highest value of utility function. Generally, this is not the outcome with the highest utility value of all outcomes, because this outcome might not be reachable due to the actions of the other players (that means that other players will play - in pursue of maximizing their utility value - such actions, that will make reaching this outcome impossible).

Utility values for the outcomes are usually represented by *payoff matrix* M, an n-dimensional matrix, such that $M(s_1, ..., s_n)$ is a tuple $(u_1(o), ..., u_n(o))$, where o is outcome of the game, in which player 1 played strategy s_1, player 2 played s_2, ... and player n played strategy s_n. This means, that outcome o yields utility value of $u_1(o)$ for player 1, $u_2(o)$ for player 2 etc.

	c	d
a	2;3	1;0
b	0;1	1;1

Example of a payoff matrix:

Let us assume for the rest of the work, that player 1's actions are equivalent to rows and player 2's to columns. In this example player 1 can play actions a or b, while player 2 can play c or d. When player 1 plays a and 2 plays c, an outcome is reached, in which player 1 gets utility equal to 2 and player 2 gets utility equal to 3.

2.2 Actions and strategies

An *action* performed by a player in a state of the game advances the game to another state (e.g., in chess, a state is a position of figures on a chess-board; an action - moving one piece changes the position, creating another state). In some games, players act simultaneously and the resulting state depends on the combination of actions of all players (rock, paper, scissors). In other games players choose actions sequentially and each action moves game to another state (chess). An action taken in a state of the game will definitely yield a new state, but there are games, in which that state is not uniquely determined, instead of which the actions leads to a probability distribution over several possible states (often referred to as stochastic actions, or having the element of nature).

A *strategy* of a player is the probability distribution over all possible actions for every state of the game, in which this player takes action.

The simplest way to create a strategy is to choose one action for every state and always play this action (with probability 1). This is called a *pure strategy.*

The more general concept means to choose the decision in every state randomly depending on a probability distribution over all actions available in that state. This is called a *mixed strategy.*

For every mixed strategy s of player p, we define *support* of s as the set of pure strategies, that are played in s with non-zero probability. Let us denote S_p set of all possible strategies for player p.

Set of strategies $R = \{s_1, s_2, ..., s_n\}$, where s_p is a strategy (mixed or pure) for player p, is called a *strategy profile.* Let us denote the set $R_{-p} = \{s_1, s_2, ..., s_{p-1}, s_{p+1}, ..., s_n\}$ as strategy profile without player p's strategy and the whole strategy profile as $R = (R_{-p}, s_p)$.

Because strategies with random element (mixed strategies or stochastic actions) may yield different utility for a player every time they are played, we will need to consider this fact in the utility function. For these strategies, we will define the value of the utility function as:

$$u_p(s_p, R_{-p}) = \sum\nolimits_{o \in O} P_{s_p, R_{-p}}(o)\ N(o)\ u_p(o)$$

where

- s_p is strategy of player p, R_{-p} is strategy profile without player p's strategy.

- $u_p(s_p, R_{-p})$ is the utility function of player p, strategies s_p and R_{-p}
- O is set of possible outcomes
- $P_{s_p,R_{-p}}(o)$ is the probability that outcome o will be reached when player p plays strategy s_p and all other players play their strategies from R_{-p}
- $C(o)$ is the probability that o will be reached due to the stochastic actions and it is equal to the product of the probabilities of actions needed to reach o, assuming all players played their actions corresponding to history of o.
- $u_p(o)$ is value of utility function for outcome o and player p.

$u_p(s_p, R_{-p})$ is equal to the expected value of probability distribution over utilities of possible outcomes given the mixed strategy s_p and stochastic actions and is calculated as average utility value yield by this strategy profile.

2.2.1 Best response and Nash equilibrium

Consider player p, his strategy s_p and strategy profile without p's strategy $R_{-p} = \{s_1, s_2, ..., s_{p-1}, s_{p+1}, ...s_n\}$. If for every (generally mixed) strategy of player p (denote it s'_p) is $u_p(s_p, S_{-p}) \geq u_p(s'_p, R_{-p})$, we say that s_p is player p's *best response* to S_{-p} [4]. Usually, best response to a given set of strategies of other players is not unique.

Strategy profile $R = (s_1, ..., s_n)$ is called *Nash equilibrium* if for every player p is s_p p's best response to R_{-p}.

2.3 Stackelberg games

In *Stackelberg games*, we seek a different solution concept called Stackelberg equilibrium. It is different in terms of computing strategies and possible outcomes. In the following paragraph, we will describe main difference between Stackelberg and the other games. From now on, we will consider game of only two players (it can be extended to n-player case; e.g. see [7]).

We have so far assumed, that all the players have the same role in the game and the same possibilities. For some problems, this is not exactly desired assumption. In Stackelberg games, one player is called the *leader* while the other one is called the $follower$. We will use index 1 for the leader, 2 for the follower and we will denote S_1 set of strategies of the leader and similarly S_2 strategies of the follower. Before the beginning of the game, the leader is forced to commit himself to a strategy he will play, the follower observes this commitment and plays his best response to the leader's committed strategy.

2.3.1 Strong Stackelberg equilibrium

We now define *Strong Stackelberg equilibrium* (SSE). In this concept, the leader knows that follower will be playing his best response to whichever strategy the leader will commit to. Therefore, leader needs to commit himself to such a strategy, which will with it's best response from the follower yield maximal utility for the leader. We assume, that the follower prefers outcome with higher utility value of the leader over the outcome with lower utility, assuming that both outcomes yield the same utility value for the follower. Formally, let us denote K the set of all possible strategy profiles for both players ($K = \{(s_1, s_2)|s_1 \in S_1, s_2 \in S_2\}$), $K^* \subseteq K$ the set of all strategy profiles, in which strategy of the follower is a best response to leader's strategy ($K^* = \{(s_1, s_2)|s_1 \in S_1, s_2 \in {S_2}^*(s_1)\}$, where ${S_2}^*(s_1) \subseteq S_2$ is set of follower's best responses to s_1), $U_1 : K \mapsto \mathbb{R}$ leader's utility function. Strong Stackelberg equilibrium is any strategy profile s, such that $s = \underset{s \in S^*}{\operatorname{argmax}}\, U_1(s)$, which means, that leader is playing such strategy, which - when combined with it's best response played by the follower - yields the highest utility value for the leader.

The following example [3] shows, that Nash and Stackelberg equilibrium are generally not the same. Given that both players move simultaneously, the only Nash equilibrium will be reached - player 1 plays a and player 2 plays c. However, if player 1 is able to commit himself to a strategy first, the Stackelberg equilibrium will be reached - 1 plays a or b with probability 0.5 and 2 plays d.

	c	d
a	2;1	4;0
b	1;0	3;2

Since a mixed strategy s_2 of the follower is a probability distribution over his pure strategies and for a fixed leader's strategy s_1 the value of U_1 is given by the same probability distribution over U_1 of s_1 and those pure strategies, it is clear, that $U_1(s_1, s_2)$ will be always lesser or equal than $U_1(s_1, s_2')$, where s_2' is such pure strategy used in s_2, that yields maximal $U_1(s_1, s_2')$. Therefore, at least one SSE exists (and it always exists [10]), such that follower plays a pure strategy. That is very helpful when computing SSE, as the leader can consider only follower's pure strategies, for every such strategy find his own strategy so that this strategy profile is element of S^* and then select the one that yields maximal utility for him.

Special case of Stackelberg games are *Bayesian Stackelberg games* [3], in which the leader doesn't exactly know, which follower type he is facing - there is a finite set of follower types, each of them has different utility function, and known probability distribution over follower types occurrences. That can be solved by setting utility values for all outcomes equal to sum of equivalent utility values multiplied by probability of equivalent follower type's occurrence over all follower types. Computation of SSE in Bayesian Stackelberg game is, however, an NP-hard problem [3].

2.4 Game classification based on utility dependence

Since every player is trying to maximize his utility, it is clear, that player interactions and behavior will reflect the dependencies between utilities of players in possible terminal states. We will describe several common categories.

2.4.1 Constant-sum games

A Game of two players, in which exists a real constant q, such that for every outcome of the game (terminal states of a game) the sum of utilities of both players is equal to q, we talk about *constant-sum game*.

Example of a constant-sum game with q = 2:

	c	d
a	1;1	0;2
b	2;0	3;-1

If q = 0, we talk about *zero-sum game*. The behavior of agents in zero-sum games is strictly competitive. However, since adding a real constant to utility value of every outcome doesn't change the optimal strategy, every constant-sum game can be modified to a zero-sum game and vice versa. Therefore, agents will behave strictly competitive in every general-sum game. Rock, paper, scissors as an example of zero-sum game:

	rock	paper	scissors
rock	0;0	-1;1	1;-1
paper	1;-1	0;0	-1;1
scissors	-1;1	1;-1	0;0

2.4.2 General-sum games

A *general-sum game* is such game, in which there is no obvious dependence between utilities of different players in different outcomes. We cannot presume cooperative or competitive behavior of agents, only that they will maximize their utility.Example of strictly cooperative general-sum game:

	c	d
a	4;3	2;2
b	-1;0	3;1

2.5 Game classification based on representation

Representation of the game is a mathematical model, in which the game can be described and solved. Obviously, more complicated games will need more complex and more difficult model to represent them. We will describe several most frequently used representations.

2.5.1 Normal-form games

Normal-form game is the simplest form of game. In this model, every player chooses one pure strategy available for him to play and once all players have made their actions, each of them gains an utility depending on combination of those actions. Formally, game in Normal form is a tuple (N, A, u), where

- N is a set of 2 players (agents), in this work indexed by p.
- $A = A_1 \times A_2 \times ... \times A_n$ is a space of actions, A_p refers to possible actions of p-th player. A_p also corresponds to p's set of pure strategies.
- u is a tuple $(u_1, u_2, ..., u_n)$, where $u_p : A \mapsto \mathbb{R}$ is a utility function of p-th player.

The "Rock, paper, scissors" example used above to present a zero-sum game is also a normal form game.

The algorithm for finding SSE in normal-form games [1] consists of computing one linear program for every pure strategy of the follower. Every linear program computes a strategy profile, in which strategy of the leader is optimal, such that the follower's fixed pure strategy is best response to strategy of the leader. From those strategy profiles computed by the linear programs, the one with the highest expected utility of the leader is chosen.

2.5.2 Extensive form games

Extensive-form games are used to effectively model finite sequential games, allowing stochastic actions, strategies of more than one action and imperfect observation of players. Formally, the game in the extensive form is a tuple $(N, H, Z, A, \rho, u, C, I)$, where

- N is a set of 2 players (agents) indexed by p.
- H is a finite set of states of the game, usually represented by a game tree. Each state is unique and holds information about all actions taken by all players (plus Nature) so far.
- $Z \subseteq H$, is a set of terminal states of the game. By reaching any terminal state, the game ends and each player is given his utility value, depending on which terminal state was reached.
- A is set of all actions. Mostly, we will care just about a subset of actions available for a player acting in state $h \in H$. This subset will be denoted as $A(h)$.

- $\rho : H \mapsto N \cup \{r\}$ is a function, which assigns each state of the game a player acting in it. r represents the element of Nature, meaning that if $\rho(h) = r$, one possible action will be chosen in state h due to a known probability distribution over $A(h)$.
- u is a tuple $(u_1, u_2, ..., u_n)$, where $u_p : Z \mapsto \mathbb{R}$ is a utility function of p-th player.
- $C : H \mapsto [0, 1]$ is a function assigning each $h \in H$ a probability that state h will be reached, assuming that all players will choose actions corresponding to history of h. The value of $C(h)$ is a product of probabilities of actions taken by element of nature in every state in history of h needed to reach h.
- I is a tuple $(I_1, I_2, ..., I_n)$ set of n sets of states used to model imperfect information in the game. Set I_p represents information that player p has in the following way: States of the game H are divided into sets I_{p_i}, such that in each set I_{p_i}, there is at least one element of H and every $h \in H$ belongs to exactly 1 set I_{p_i}. For each state in set I_{p_i}, player is not able to distinguish this state from any other state in I_{p_i}. In other words, set I_{p_i} includes all states, whose history the player p cannot distinguish due to imperfect observation.

A pure strategy of player p in an extensive-form game corresponds to assigning one action $a \in A(h)$ to every state $h \in H$, in which player p takes actions, such that a will be played every time h is reached. The same action is assigned to every state of the same information set.

We will restrict only on games with *perfect recall*, which means that every player remembers his previous actions throughout the game and never forgets any information. Formally [4], player p has perfect recall, if for every two states $h_p, h'_p \in H$, that are in the same information set I_{p_i} of player p, for any path (from the root state of the game h_0 to h_p) $h_0, a_0, h_1, a_1, ..., h_m, a_m, h_p$ and any path (from h_0 to h'_p) $h_0, a'_0, h'_1, a'_1, ..., h'_{m'}, a'_{m'}, h'_p$ (where h_j are states of the game and a_j actions available in h_j), it must be the case that:

- $m = m'$
- for all $0 \leq j \leq m$, if player p takes action in h_j, then h_j, h'_j are in the same information set I_{p_k} of the player p and $a_j = a'_j$ (player p takes the same action in h_j and h'_j).

Example of game in the extensive form are card games like poker or dices games like Backgammon.

2.5.3 Sequence form

Sequence form of the game [2] is a compact representation of mixed strategies in extensive-form games. Let us denote σ_p a sequence of actions taken in history of a state h by player p and Σ_p the set of all sequences of player p, empty sequence will be denoted as $\emptyset$. The history of h can be represented as $(\sigma_1, \sigma_2, ..., \sigma_n, \sigma_r)$ - a tuple of sequences played by each player and the Nature. If sequence σ_p doesn't lead to a terminal state, it can be expanded by action a_p (available to player p in state h, corresponding to σ_p, assuming that p takes an action in h) to sequence $\sigma'_p = \sigma_p a_p$. All states in one information set I_{p_i} of player p share

the same sequence of his actions. Every sequence σ_p uniquely defines an information set for player p (because all states in information set of player p share the same sequence, all the states that contain σ_p will form this information set). A mixed strategy in sequence form game are usually represented as probability distribution over sequences of the player. We will call this probability distribution a realization plan of the player.

Algorithm that finds SSE in EFG *c5* exploits representation of the game as Sequence form. The solution consists of solving a MILP, which has a linear size to the size of the game tree and the only integer variables are binary (0 or 1). Number of these binary variables is equal to the number of follower's possible sequences of actions.

Chapter 3

Computing Strong Stackelberg equilibrium in NFGSS

We will focus on Stackelberg normal-form games with sequential strategies for the rest of this work. In the first part, we will briefly introduce this class of games.
In the second part, we will present two algorithms for computing SSE for normal-form games with sequential strategies.
In the final part of this Chapter, we investigate the possibility for applying iterative algorithms to compute SSE in this class of games. As described in existing iterative algorithm for normal-form games [12], one of the most important problems, which is yet to be solved, is quick computation of effectively tight upper bound. We will present two algorithms to compute upper bound for Stackelberg NFGSS.

3.1 Normal-form games with sequential strategies

Despite their title, *normal-form games with sequential strategies* [6] are used to model much more complicated games than normal-form games. Actually, they are more similar to extensive-form games than normal-form games. This representation is used as a simplification of the extensive-form games, because computing optimal Stackelberg strategies in extensive form game is an NP-hard problem [11]. Normal-form games with sequential strategies (NFGSS) have terminal and non-terminal states and element of nature. NFGSS is often used to model such games, in which players have no information about the actions of the opponents until the terminal state is reached. To model such games, we use an acyclic finite Markov decision process (MDP) for every player instead of one big game tree describing all players together. Every state in MDP of player p represents one of his information sets. We will use the following notation:

- H_p is set of all states in player p's MDP (his information sets).
- A_p is set of actions in p's MDP. However, we will often use A_p for a function, which for every state $h_p \in H_p$ refers to actions available to p in state h_p ($A_p(h_p) \subseteq A_p$).

- $C : H_p \times A_p \times H_p \mapsto [0, 1]$ is function used to represent stochastic actions. For states $h_{p_1}, h_{p_2} \in H_p$ and action $a \in A_p(h_{p_1})$ is $C(h_{p_1}, a, h_{p_2})$ equal to the probability, that h_{p_2} will be reached by playing a in h_{p_1}.

- u_p is a utility function of player p. It can be assigned in some state $h_p \in H_p$ when action $a_p \in A_p(h_p)$ is played and other player $(-p)$ plays action a_{-p} in state h_{-p} (given that $a_{-p} \in A_{-p}(h_{-p})$) and we will denote it $u_p((h_p, a_p), (h_{-p}, a_{-p}))$. For a mixed strategy profile (s_p, s_{-p}) (s_p is a strategy of player p and s_{-p} is a strategy of players $-p$) we define $u_p(s_p, s_{-p})$ as

$$\sum_{H_p \times A_p} \sum_{H_{-p} \times A_{-p}} s_p(h_p, a_p)\ s_{-p}(h_{-p}, a_{-p})\ u_p((h_p, a_p), (h_{-p}, a_{-p}))$$

  where $s_p(h_p, a_p)$ is a probability that state h_p will be reached and then a_p will be played by p by following s_p and $s_{-p}(h_{-p}, a_{-p})$ is probability that other players will reach their respective states from h_{-p} and then play actions from a_{-p} following strategies from s_{-p}.

Pure strategy of player p in NFGSS is defined as choosing one action for every state of p's MDP, such that this action will be played every time this state is reached. Mixed strategy assigns a probability distribution to every state of p's MDP over actions available in that state.

There are two major differences between previous game representations and NFGSS. Most importantly, utility value for each player is defined for every combination of players actions instead of combination of terminal states and therefore, each player can obtain some utility value every time he plays an action.

The other major difference is, that in MDP, one state can be reached by different action sequences (MDP is a graph, generally not a tree). This, however, prevents us from using functions like $C(h)$ from extensive-form game (probability, that state h will be reached due to Nature, assuming that all actions needed to reach h were played), because the assumption of perfect recall does not hold in NFGSS.

This is useful to model such games, in which reaching the same state can yield different utility values for different sequences leading to this state. MDPs are usually significantly smaller than any game-tree representation would be, allowing us to model larger games. Also, NFGSS allows simpler strategy representation than extensive-form games. Terminal states are used to ensure that the game is finite, as there is no action available for any player in a terminal state.

In NFGSS, there is one leader indexed with 1, his MDP containing set of his states H_1, set of strategies S_1, set of his actions A_1 (and $A_1(h_1)$ actions available in h_1), one follower indexed with 2, his set of states H_2 in his MDP, set of his strategies S_2 and set of his actions A_2 (and $A_2(h_2)$ actions available in h_2).

LP for NFGSS [6] finds a Nash equilibrium for zero-sum games in NFGSS. This algorithm exploits the fact, that both players have the same expected utility values in the root states of their MDPs in equilibrium strategies and that the minimizing player (the follower in terms of Stackelberg games) plays the action with minimal expected utility in every state of his MDP. Using those facts, the LP is formed and by solving it, we find a Nash equilibrium.

Nash and Stackelberg equilibria coincides in zero-sum games, therefore, this algorithm can be used to find SSE in zero-sum NFGSS.

3.2 Strong Stackelberg equilibrium in normal-form games with sequential strategies

In the first part of this Section, we will focus on finding SSE in NFGSS. There are two standard approaches for computing SSE that we will apply for NFGSS: The first is to solve the game by computing multiple linear programs (one for each pure strategy of the follower [1]), the other is to formulate and compute a MILP [3] [5].

3.2.1 Multiple LP's representation

The main idea of this approach is finding a mixed leader's strategy s_1 for every pure strategy s_2 of the follower, so that s_2 is the best response to s_1 and the expected utility of the leader is maximal. That means solving a LP for every pure strategy of the follower. Since the number of pure strategies is exponential based on the strategy length (number of actions in sequences leading to terminal states), we will need to solve exponentially many LPs consisting of exponentially many constraints.

Assuming a fixed strategy of the follower, let us denote:

- $P_1(h_1)$ the probability that state $h_1 \in H_1$ will be reached and $P_1(h_1, a_1)$ the probability that h_1 will be reached and action $a_1 \in A_1$ will be then played (assuming $a_1 \in A_1(h_1)$, otherwise it is always equal to 0). These probabilities form the mixed strategy we are computing.
- $P_2(h_2), P_2(h_2, a_2)$ are similar probabilities for the follower and since the follower's strategy is fixed, these probabilities are constants. Follower strategy is best response to the leader's strategy we are computing.
- $S'_2 = \{(h_2, a_2) | a_2 \in A_2, h_2 \in H_2, P_2(h_2) = P_2(h_2, a_2)\}$. That is set of states and corresponding actions used in fixed pure strategy of the follower (best response to leader's strategy).
- v_{h_2} is the expected utility of the follower, given that he will reach state $h_2 \in H_2$.
- ${h_1}^r$ is the leader's root state (state in which he will start the game). Similarly, ${h_2}^r$ is follower's root state.
- $\xi_{(h_2,a_2)}$ is a slack variable for state $h_2 \in H_2$ and action $a_2 \in A_2$. It ensures, that follower's fixed strategy is the best response to leader's strategy.
- $Z_p \subset H_p$ is set of terminal states of player p.
- $C(h_p, a_p, h'_p)$ the probability that player p reaches state h'_p by playing action a_p in state h_p.

Network-flow constraints is set of constraints ensuring that probability of reaching a state is equal to sum of probabilities of actions available in that state and that root state of a MDP is reached with probability of 1.

Stackelberg strategies are calculated by LPs in this form:

$$max \sum_{(h_1,a_1)\in H_1\times A_1} \sum_{(h_2,a_2)\in S'_2} P_1(h_1,a_1)\ P_2(h_2,a_2)\ u_1((h_1,a_1),(h_2,a_2)) \tag{1}$$

s.t.

$$P_1(h_1{}^r) = 1 \tag{2}$$

$$P_1(h_1) = \sum_{h'_1\in H_1} \sum_{a'_1\in A_1(h'_1)} P_1(h'_1,a'_1)\ C(h'_1,a'_1,h_1) \quad \forall h_1 \in H_1 \setminus \{h_1{}^r\} \tag{3}$$

$$P_1(h_1) = \sum_{a_1\in A_1(h_1)} P_1(h_1,a_1) \quad \forall h_1 \in H_1 \setminus Z_1 \tag{4}$$

$$v_{h_2} = \xi_{(h_2,a_2)} + \sum_{(h'_2,a'_2)\in H_2\times A_2} v_{h'_2}\ C(h_2,a_2,h'_2) + \sum_{(h_1,a_1)\in H_1\times A_1} P_1(h_1,a_1)\ u_2((h_1,a_1),(h_2,a_2))$$
$$\forall (h_2,a_2) \in (H_2 \setminus Z_2)\times A_2 \tag{5}$$

$$P_1(h_1,a_1) \geq 0 \quad \forall (h_1,a_1) \in H_1 \times A_1 \tag{6}$$

$$\xi(h_2,a_2) \geq 0 \quad \forall (h_2,a_2) \in H_2 \times A_2 \tag{7}$$

$$\xi(h_2,a_2) = 0 \quad \forall (h_2,a_2) \in S'_2 \tag{8}$$

$$v_{(h_2)} = 0 \quad \forall h_2 \in Z_2 \tag{9}$$

The mathematical program works as follows: Leader is trying to maximize his utility, which is described by a function (1). Constraint (2) means that the leader reaches his root state with probability of 1. Constraints (3) and (4) are the network-flow constraints. Constraint (3) ensures, that the probability of reaching state h for the leader is equal to the sum of probabilities of playing an action in some state leads to h, over all states and actions. Constraint (4) forces the probability of reaching h for the leader to be equal to the sum of probabilities of playing an action over all actions available in h. Constraint (5) describes expected utility of the follower for every follower's state. Non-negative slack variable (constraint (7)) here serves to transform inequation into an equation and by being equal to 0 (constraint (8)) for actions of follower's fixed pure strategy (S'_2), it restricts leader's searched strategy, such that S'_2 is best response to this strategy. Constraint (6) ensures (along with (2) and network-flow conditions), that all probabilities have their values between 0 and 1. Constraint (9) ensures, that follower's expected utility of his terminal states is equal to 0.

Example: Let us solve the following simple example (see Figure 4.1): leader's MDP has 6 states
$H_1 = \{h_1{}^r, h_{1_1}, h_{1_2}, h_{1_3}, h_{1_4}, h_{1_5}\}$ and he can play actions $A_1 = \{a_{1_1}, a_{1_2}, a_{1_3}, a_{1_4}, a_{1_5}\}$, where actions a_{1_1}, a_{1_2} can be played in $h_1{}^r, C(h_1{}^r, a_{1_1}, h_{1_1}) = 0.85, C(h_1{}^r, a_{1_1}, h_{1_2}) = 0.15$, $C(h_1{}^r, a_{1_2}, h_{1_2}) = 1$.

Follower's MDP is deterministic (no stochastic actions) and consists of 7 states $\{h_2{}^r, h_{2_1}, h_{2_2}, h_{2_3}, h_{2_6}, h_{2_5}, h_{2_6}\}$, he can play actions $A_2 = \{a_{2_1}, a_{2_2}, a_{2_3}, a_{2_4}, a_{2_5}, a_{2_6}\}$.

Any combination (x_1, x_2), $x_1 \in \{(h_{1_1}, a_{1_3}), (h_{1_1}, a_{1_4}), (h_{1_2}, a_{1_5})\}$, $x_2 \in \{(h_{2_1}, a_{2_4}), (h_{2_2}, a_{2_5}), (h_{2_3}, a_{2_6})\}$ yields utility for both players described by a matrix bellow, any unlisted combination yields utility equal to 0 for both players:

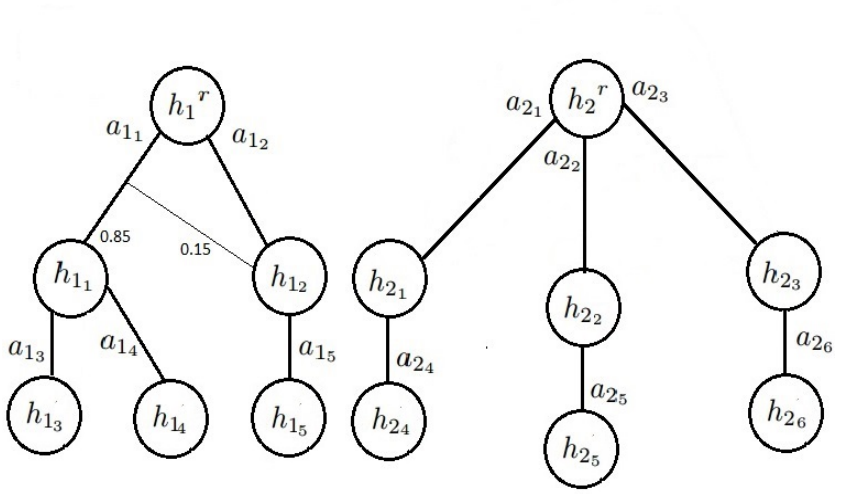


Figure 3.1: Example: MDPs of both players

Payoff matrix:

	(h_{2_1}, a_{2_4})	(h_{2_2}, a_{2_5})	(h_{2_3}, a_{2_6})
(h_{1_1}, a_{1_3})	0;-1	2;1	1;-2
(h_{1_1}, a_{1_4})	-1;3	-3;2	3;0
(h_{1_2}, a_{1_5})	1;0	0;-1	2;2

Before we will create LPs solving this problem, let us denote a set of several constraints as B. Since many of these constraints will be the same for all LPs, B will be set of constraints used in every LP. In those constraints, set $H_1 \times A_1$ has elements $(h_1{}^r, a_{1_1}), (h_1{}^r, a_{1_2}), (h_{1_1}, a_{1_3})$, $(h_{1_1}, a_{1_4}), (h_{1_2}, a_{1_5})$, set $H_2 \times A_2$ consists of $(h_2{}^r, a_{2_1}), (h_2{}^r, a_{2_2}), (h_2{}^r, a_{2_3}), (h_{2_1}, a_{2_4}), (h_{2_2}, a_{2_5})$, (h_{2_3}, a_{2_6}), set Z_1 has elements $h_{1_3}, h_{1_4}, h_{1_5}$ and similarly set Z_2 consists of $h_{2_4}, h_{2_5}, h_{2_6}$:

$B = \{$

$$P_1(h_1{}^r) = 1$$

$$P_1(h_{1_1}) = 0.85\ P_1(h_1{}^r, a_{1_1})$$

$$P_1(h_{1_2}) = 0.15\ P_1(h_1{}^r, a_{1_1}) + P_1(h_1{}^r, a_{1_2})$$

$$P_1(h_1{}^r) = P_1(h_1{}^r, a_{1_1}) + P_1(h_1{}^r, a_{1_2})$$

$$P_1(h_{1_1}) = P_1(h_{1_1}, a_{1_3}) + P_1(h_{1_1}, a_{1_4})$$

$$P_1(h_{1_2}) = P_1(h_{1_2}, a_{1_5})$$
$$P_1(h_{1_3}) = P_1(h_{1_1}, a_{1_3})$$
$$P_1(h_{1_4}) = P_1(h_{1_1}, a_{1_4})$$
$$P_1(h_{1_5}) = P_1(h_{1_2}, a_{1_5})$$
$$P_1(h_1, a_1) \geq 0 \quad \forall (h_1, s_1) \in H_1 \times A_1$$
$$v_{h_2{}^r} = \xi_{(h_2{}^r, a_{2_1})} + v_{h_{2_1}}$$
$$v_{h_2{}^r} = \xi_{(h_2{}^r, a_{2_2})} + v_{h_{2_2}}$$
$$v_{h_2{}^r} = \xi_{(h_2{}^r, a_{2_3})} + v_{h_{2_3}}$$
$$v_{h_{2_1}} = \xi_{(h_{2_1}, a_{2_4})} - P_1(h_{1_1}, a_{1_3}) + 3\ P_1(h_{1_1}, a_{1_4}) + v_{h_{2_4}}$$
$$v_{h_{2_2}} = \xi_{(h_{2_2}, a_{2_5})} + P_1(h_{1_1}, a_{1_3}) + 2\ P_1(h_{1_1}, a_{1_4}) - P_1(h_{1_2}, a_{1_5}) + v_{h_{2_5}}$$
$$v_{h_{2_3}} = \xi_{(h_{2_3}, a_{2_6})} - 2\ P_1(h_{1_1}, a_{1_3}) + 2\ P_1(h_{1_2}, a_{1_5}) + v_{h_{2_6}}$$
$$\xi_{h_2} \geq 0 \quad \forall (h_2, a_2) \in H_2 \times A_2$$
$$v_{z_2} = 0 \quad \forall z_2 \in Z_2 \ \}$$

There are 3 pure strategies of the follower, since for $h_{2_1}, h_{2_2}, h_{2_3}$, there is always only 1 possible action, so pure strategies differ only in action taken in $h_2{}^r$. Let us denote S_{2p_1} a strategy with a_{2_1} taken in $h_2{}^r$, S_{2p_2} a strategy with a_{2_2} taken in $h_2{}^r$, S_{2p_3} a strategy with a_{2_3} taken in $h_2{}^r$.We will use following constants for these strategies:

	S_{2p_1}	S_{2p_2}	S_{2p_3}
$P_2(h_2{}^r)$	1	1	1
$P_2(h_2{}^r, a_{2_1})$	1	0	0
$P_2(h_2{}^r, a_{2_2})$	0	1	0
$P_2(h_2{}^r, a_{2_3})$	0	0	1
$P_2(h_{2_1}) = P_2(h_{2_1}, a_{2_4})$	1	0	0
$P_2(h_{2_2}) = P_2(h_{2_2}, a_{2_5})$	0	1	0
$P_2(h_{2_3}) = P_2(h_{2_3}, a_{2_6})$	0	0	1
$P_2(h_{2_4})$	1	0	0
$P_2(h_{2_5})$	0	1	0
$P_2(h_{2_6})$	0	0	1

We will use these constants to calculate the objective function for each LP. The objective function has following general form:

$$P_1(h_{1_1}, a_{1_3})\ (0\ P_2(h_{2_1}, a_{2_4}) + 2\ P_2(h_{2_2}, a_{2_5}) + 1\ P_2(h_{2_3}, a_{2_6})) +$$
$$P_1(h_{1_1}, a_{1_4})\ (-1\ P_2(h_{2_1}, a_{2_4}) - 3\ P_2(h_{2_2}, a_{2_5}) + 3\ P_2(h_{2_3}, a_{2_6})) +$$
$$P_1(h_{1_2}, a_{1_5})\ (1\ P_2(h_{2_1}, a_{2_4}) + 0\ P_2(h_{2_2}, a_{2_5}) + 2\ P_2(h_{2_3}, a_{2_6}))$$

We will now present a LP for every pure strategy of the follower. See Table 3.1. for results of those LPs.

LP for S_{2p_1} will have following form:

$$max \ -1 \ P_1(h_{1_1}, a_{1_4}) + 1 \ P_1(h_{1_2}, a_{1_5})$$

s.t.

$$B$$

$$\xi(h_2{}^r, a_{2_1}) = 0$$

$$\xi(h_{2_1}, a_{2_4}) = 0$$

LP for S_{2p_2} will have following form:

$$max \ 2 \ P_1(h_{1_1}, a_{1_3}) - 3 \ P_1(h_{1_1}, a_{1_4})$$

s.t.

$$B$$

$$\xi(h_2{}^r, a_{2_2}) = 0$$

$$\xi(h_{2_2}, a_{2_5}) = 0$$

LP for S_{2p_3} will have following form:

$$max \ 1 \ P_1(h_{1_1}, a_{1_3}) + 3 \ P_1(h_{1_1}, a_{1_4}) + 2 \ P_1(h_{1_2}, a_{1_5})$$

s.t.

$$B$$

$$\xi(h_2{}^r, a_{2_3}) = 0$$

$$\xi(h_{2_3}, a_{2_6}) = 0$$

	S_{2p_1}	S_{2p_2}	S_{2p_3}
$P_1(h_{1_1})$	0.5333	0.85	0.4
$P_1(h_{1_2})$	0.4667	0.15	0.6
$P_1(h_1{}^r, a_{1_1})$	0.6275	1	0.4706
$P_1(h_1{}^r, a_{1_2})$	0.3725	0	0.5294
$P_1(h_{1_1}, a_{1_3}) = P_1(h_{1_3})$	0.3333	0.85	0
$P_1(h_{1_1}, a_{1_4}) = P_1(h_{1_4})$	0.2	0	0.4
$P_1(h_{1_2}, a_{1_5}) = P_1(h_{1_5})$	0.4667	0.15	0.6
Expected leader's utility	0.2667	1.7	**2.4**

Table 3.1: Results of the example

This mixed strategy yields expected utility equal to 2.4.

Strategy for S_{2p_3} yields maximal expected utility for the leader, so optimal strategy, which will leader commit to, is playing a_{1_1} in $h_1{}^r$ with probability 0.4706, a_{1_2} in $h_1{}^r$ with

probability 0.5294, a_{1_4} in h_{1_1} and a_{1_5} in h_{1_2}.

As stated above, those LPs will solve the problem, but there will be exponentially many LPs (one for every pure strategy of the follower) and therefore this solution will be applicable only on small games. To improve this solution, we can reformulate the problem by using MILP.

3.2.2 MILP representation

The main idea for reformulation the multiple LPs algorithm as a MILP is introducing binary variables $P_2 : H_2 \times A_2 \mapsto \{0, 1\}$, which for state $h_2 \in H_2$ and action $a_2 \in A_2$ represents the probability that h_2 will be reached and a_2 played. We will again use the same network-flow constraints and slack variables. Leader is again forced to play such mixed strategy, to which the follower's strategy is the best response. The difference is that we do not have a fixed follower's strategy and we need to represent it in the program. We cannot use constants (as before) forcing slack of one strategy to be equal to 0 directly. We exploit the binary variables $P_2(h_2, a_2)$, which are equal to 1 for every (h_2, a_2) used in follower's computed strategy and we force slack variables of these states and actions to be equal to 0. For this representation, let us assume that MDP of the follower is deterministic, otherwise (even for pure strategies) the probabilities of actions in this strategy could be a real number between 0 and 1, making it impossible to use binary variables and the solution would be more complicated (it would result in a mixed integer quadratic program - MIPQ).

Since the strategy of the follower isn't fixed, we will also have to use different objective function. The utility of the leader depends on the probabilities of states and actions of both players, therefore we introduce a new variable $P : H_1 \times A_1 \times H_2 \times A_2 \mapsto [0, 1]$, which will represent a joint probability, that both leader reaches state h_1, plays action a_1 and follower reaches state h_2 and there he plays action a_2. Similarly, we will use also variable $P : H_1 \times H_2 \mapsto [0, 1]$, representing probability that leader reaches state h_1 and follower state h_2. We will create new network-flow constraints for these variables as well. It is obvious, that $P(h_1, a_1, h_2, a_2) = P_1(h_1, a_1)\ P_2(h_2, a_2)$, but that interpretation would result in MIQP, which is much more difficult to compute. Therefore, we will use variable $P(h_1, a_1, h_2, a_2)$ and force it to be equal to $P_1(h_1, a_1)\ P_2(h_2, a_2)$ by using network-flow linear constraints and the assumption, that follower actions are deterministic. With all those new variables, we can formulate MILP solving the problem of finding SSE for NFGSS:

$$max \sum_{(h_1,a_1,h_2,a_2)\in Z} P(h_1, a_1, h_2, a_2)\ u_1((h_1, a_1), (h_2, a_2)) \tag{10}$$

s.t.

$$v_{h_2} = \xi_{(h_2,a_2)} + \sum_{(h'_2,a'_2)\in H_2\times A_2} v_{(h'_2,a'_2)}\ C(h_2, a_2, h'_2) + \sum_{(h_1,a_1)\in H_1\times A_1} P_1(h_1, a_1)\ u_2((h_1, a_1), (h_2, a_2))$$

$$\forall (h_2, a_2) \in (H_2 \setminus Z_2) \times A_2 \tag{11}$$

$$P_p({h_p}^r) = 1 \qquad \forall p \in \{1,2\} \tag{12}$$

$$P_p(h_p) = \sum_{h'_p \in H_p} \sum_{a'_p \in A_p(h'_p)} P_p(h'_p, a'_p)\, C(h'_p, a'_p, h_p) \;\; \forall p \in \{1,2\}, \forall h_p \in H_p \setminus \{{h_p}^r\} \tag{13}$$

$$P_p(h_p) = \sum_{a_p \in A_p(h_p)} P_p(h_p, a_p) \quad \forall p \in \{1,2\}, \forall h_p \in H_p \setminus Z_p \tag{14}$$

$$P_2(h_2, a_2) \in \{0,1\} \;\; \forall (h_2, a_2) \in H_2 \times A_2 \tag{15}$$

$$0 \leq \xi_{(h_2,a_2)} \leq (1 - P_2(h_2, a_2))\, M \quad \forall (h_2, a_2) \in H_2 \times A_2, M \gg 1 \tag{16}$$

$$0 \leq P(h_1, h_2) \leq P_1(h_1) \;\; \forall h_1 \in H_1, h_2 \in H_2 \tag{17}$$

$$P(h_1, h_2) \leq P_2(h_2) \;\; \forall h_1 \in H_1, h_2 \in H_2 \tag{18}$$

$$0 \leq P(h_1, a_1, h_2, a_2) \leq P_1(h_1, a_1) \;\; \forall (h_1, a_1, h_2, a_2) \in H_1 \times A_1 \times H_2 \times A_2 \tag{19}$$

$$P(h_1, a_1, h_2, a_2) \leq P_2(h_2, a_2) \;\; \forall (h_1, a_1, h_2, a_2) \in H_1 \times A_1 \times H_2 \times A_2 \tag{20}$$

$$P({h_1}^r, {h_2}^r) = 1 \tag{21}$$

$$P(h_1, h_2) = \sum_{a_1 \in A_1(h_1)} \sum_{a_2 \in A_2(h_2)} P(h_1, a_1, h_2, a_2) \;\; \forall h_1 \in H_1 \setminus Z_1, h_2 \in H_2 \setminus Z_2 \tag{22}$$

$$P(h_{1_1}, h_{2_1}) = \sum_{(h_1, a_1, h_2, a_2) \in H_1 \times A_1 \times H_2 \times A_2} P(h_1, a_1, h_2, a_2) \;\; C(h_1, a_1, h_{1_1}) \;\; C(h_2, a_2, h_{2_1})$$
$$\forall h_{1_1} \in H_1 \setminus {h_1}^r, h_{2_1} \in H_2 \setminus {h_2}^r \tag{23}$$

$$v_{h_2} = 0 \quad \forall h_2 \in Z_2 \tag{24}$$

Mathematical program works as follows: Function (10) is the objective function we are trying to maximize and it is equal to the expected value of utility for the leader. Constraint (11) is similar to (5) in the formulation for multiple LPs and again it is the follower's expected value, given that state h_2 is reached. Maximal expected value of utility is reached when the slack variable is equal to 0, which means that the follower's strategy, for which every state and action played in this strategy has slack variable equal to 0, is best response to leader's strategy. Constraints (12), (13) and (14) are network-flow constraints. Constraint (12) ensures that both players will definitely start the game in their root state. Constraint (13) forces the probability that state h will be reached to be equal to sum of probabilities that by playing an action in other state will lead to s, over all states and actions (13). Constraint (14) describes, that some action needs to be played in the state h if it is reached. Constraint (15) defines variable P_2 as stated above. Constraint (16) modifies (7) and (8) from multiple LPs solution, stating that for those states actions, which yield value of P_2 equal to 1, is their slack variable equal to 0 and this pure strategy is therefore (as was described in explanation of (11)) best response to leader's computed strategy. It is advisable to use reasonably low constant M, such that it does not limit slack variables. Too high value of M leads to

longer computation and numeric errors. We used M equal to the product of maximal utility value of follower's actions and the length of strategies (number of actions in the longest sequence). Constraints (17), (18), (19), (20) restrict variables for state (and action) tuples with an upper bound, while lower bound is provided by network-flow Constraints for those tuples (21), (22), (23). $C(h_2, a_2, h_{2_1})$ in Constraint (23) is always equal to 0 or 1, because the follower is deterministic. Constraints (17) and (19) also force all variables representing probabilities to be non-negative, which, along with network-flow, is ensuring, that they really are probabilities. Constraint (24) is similar to (9).

Note that while we define utility as well as create variables $P(h_1, a_1, h_2, a_2)$ and $P(h_1, h_2)$, for every tuple $(h_1, a_1, h_2, a_2) \in H_1 \times A_1 \times H_2 \times A_2$, there is need to do that while computing the MILP. We can restrict the MILP only on those tuples, for which there exists a player p, such that $u_p(h_1, a_1, h_2, a_2) \neq 0$, plus every tuple needed for network-flow constraints to be closed on successors and predecessors. That means, that for every state tuple $(h_1, a_1, h_2, a_2) \in H_1 \times A_1 \times H_2 \times A_2$, every $h'_1 \in H_1, h'_2 \in H_2$, such that h'_1 is a possible successor of (h_1, a_1) and h'_2 is possible successor of (h_2, a_2), there must exist a variable for the tuple (h'_1, h'_2) (and similarly also variables for the predecessors must also exist). Otherwise, network-flow constraints might contain non-existing variables. If these rules hold, we can save a significant amount of memory-space and time, as there would quadratically many variables in the game and even smaller games would be challenging to compute.

3.3 Computing upper bound for a sub-game

Both previous algorithms solve the game exactly, but they does not scale well for large games. There is, however, another way to compute SSE in NFGSS. Double-Oracle algorithm [6] solves this problem for zero-sum games by iterative computation of strategies for both players, while it prunes strategies with low expected utility and expands ones with high expected utility. For general-sum games, it needs a tight upper bound, that would be reliable in all domains, calculated in short time for effective pruning (as e.g. in Bayesian normal-form games in [12]). We present two ways to compute an upper bound (UB) for Stackelberg NFGSS.

3.3.1 First variant

We want an LP for creating the upper bound, because while LP can be solved in polynomial time, solving a MILP is a NP-complete problem. First idea of creating a LP computing the upper bound is to use the MILP presented above and relax set of the constraints of the game. There is no general rule to choosing these constraints, we are trying to find a reasonable compromise between tightness of the upper bound and complexity of the computation.

Since we want to create a LP instead of MILP, the first approach is to linearize the MILP, leaving out the binary variables $P_2(h_2, a_2)$ and relaxing all constraints containing them. The only exception is Constraint (16), which was modified so that it doesn't contain the binary variable anymore. The upper bound is computed by following mathematical program:

$$max \sum_{(h_1,a_1,h_2,a_2)\in Z} P(h_1, a_1, h_2, a_2)\ u_1((h_1, a_1), (h_2, a_2)) \tag{25}$$

s.t.

$$(11), (17), (19), (21), (22), (23), (24)$$

$$P_1(h_1{}^r) = 1 \tag{26}$$

$$P_1(h_1) = \sum_{h'_1 \in H_1} \sum_{a'_1 \in A_1(h'_1)} P_1(h'_1, a'_1)\, C(h'_1, a'_1, h_1) \;\; \forall h_1 \in H_1 \setminus \{h_1{}^r\} \tag{27}$$

$$P_1(h_1) = \sum_{a_1 \in A_1(h_1)} P_1(h_1, a_1) \;\; \forall h_1 \in H_1 \setminus Z_1 \tag{28}$$

$$0 \leq \xi_(h_2, a_2) \leq \sum_{(h_1, a_1) \in H_1 \times A_1} (P_1(h_1, a_1) - P(h_1, a_1, h_2, a_2))\, M$$

$$\forall (h_2, a_2) \in H_2 \times A_2, M \gg 1 \tag{29}$$

$$P_1(h_1) = \sum_{h_2 \in H_2} P(h_1, h_2) \;\; \forall h_1 \in H_1 \tag{30}$$

Variables $P_2(h_2)$ and $P_2(h_2, a_2)$ and all constraints containing them were removed, everything else remained unchanged, except for network-flow constraints, which are applied only on the leader strategy (Constraints (26), (27), (28)). The only major change was between constraints (16) and it's counterpart (29). Assuming that (h_2, a_2) is played in follower's pure strategy (so no other action can be played in that state), following equation stands: $P_1(h_1, a_1) = P(h_1, a_1, h_2, a_2) \forall (h_1, a_1) \in H_1 \times A_1$. Therefore slack variables will still be forced to be zero for that strategy of the follower. Constraint (30) ensures, that probability of reaching leader's state h_1 is equal to sum of probabilities, that tuple of states (h_1, h_2) will be reached over all corresponding follower states.

Note that space of values for each variable satisfying all constraints in this LP definitely contains optimal solution of MILP representation and the objective function is the same for both MILP and LP. Therefore, LP can't yield lower optimal value than the MILP and because of that, this LP indeed computes an upper bound for the game. Also, note that by absence of binary variables, we no longer need the assumption of deterministic actions for the follower.

Experiments showed, that this form of computing of the upper bound is not perspective, as linearization (transformation removing the binary variables) of constraint (16) does not force the follower to play his best response to leader's strategy. Constraint (16) paired with binary variables forced probabilities of all actions with non-zero slack variable to be equal to 0. By transformation of this constraint to constraint (30), this assumption was lost, as follower is no longer forced to play pure strategy. Therefore, even actions with non-zero probability can have a non-zero slack and the follower can play any strategy. By maximizing leader's utility, this leads to both players cooperating to the leader's benefit, resulting in the upper bound being often equal to the highest possible leader's utility. Therefore, the upper bound is not tight at all. With that experience, we formulate the second form of computing the upper bound.

3.3.2 Second variant

In the second variant of computing of the upper bound, we again use the MILP described above, but we will use another idea of forcing the follower to play his best response to leader's strategy. This idea is variant of approach used in [7] for normal-form games. We force only the action with highest expected utility of the follower to be played in every follower's state. Because pure strategy in NFGSS corresponds to choosing one action in every state of the MDP, playing the one with highest expected utility is best response to leader's strategy. Given a leader's mixed strategy s_1, let us assume follower's non-terminal state h_2, actions $A_2(h_2)$ available in h_2 and action $a_2 \in A_2(h_2)$, such that expected utility of playing a_2 with leader playing s_1 is greater or equal than expected utility of any other $a'_2 \in A_2(h_2)$. We want only a_2 to be played in h_2 with non-zero probability. We formulate one constraint for every tuple $(a'_2, a''_2) \in A_2(h_2) \times A_2(h_2)$ as follows: expected utility of a'_2 weighted by probability $P_2(h_2, a'_2)$ is greater or equal than expected utility of a''_2 weighted by $P_2(h_2, a'_2)$. For $a_2 = a''_2$, expected utility of a''_2 is greater or equal than expected utility of any a'_2 when both are weighted by the same probability. Therefore, $P_2(h_2, a'_2)$ must be equal to 0 whenever $a'_2 \neq a_2$, forcing both expected utilities to be equal to 0. The only action, that can be played with non-zero probability, is a_2. This fact forces the follower to play his best response to s_1 at the cost of quadratic number of constraints. In order to formulate the LP computing the upper bound, we introduce variables $v_{(h_2,a_2)}$, representing follower's expected utility, assuming that state h_2 is reached and then the action a_2 is played, for every state and action of the follower. The upper bound is computed by LP in following form:

$$max \sum_{(h_1,a_1,h_2,a_2) \in Z} P(h_1, a_1, h_2, a_2)\ u_1((h_1, a_1), (h_2, a_2)) \tag{31}$$

s.t.

$$(17), (19), (21), (22), (23), (24), (26), (27), (28), (30)$$

$$v_{h_2} = \sum_{a_2 \in A_2(h_2)} v_{(h_2,a_2)} \quad \forall h_2 \in H_2 \setminus Z_2 \tag{32}$$

$$v_{(h_2,a_2)} = \sum_{(h_1,a_1) \in H_1 \times A_1} P(h_1, a_1, h_2, a_2) U_2(h_1, a_1, h_2, a_2) + \sum_{h'_2 \in H_2} v_{h'_2} C(h_2, a_2, h'_2)$$

$$\forall (h_2, a_2) \in H_2 \times A_2 \tag{33}$$

$$v_{(h_2,a_2)} \geq \sum_{(h_1,a_1) \in H_1 \times A_1} P(h_1, a_1, h_2, a_2) U_2(h_1, a_1, h_2, a'_2) + \sum_{h'_2 \in H_2} v_{h'_2} C(h_2, a'_2, h'_2)$$

$$\forall (h_2, a_2) \in H_2 \times A_2\ \forall a'_2 \in A_2(h_2) \tag{34}$$

This program works similarly to the previous one, but constraint (11) and (29) were replaced by (32), (33), (34). Constraint (32) forces the expected utility of every follower's state to be equal to the sum of expected utilities of actions available in that state. Constraint

(33) describes expected utility of follower's actions. Constraint (34) ensures, that follower plays his best response to leader's strategy as was described above.

Similarly to the first variant of upper bound, the space of admissible values of variables used in the objective function (which has the same form as objective function used in the MILP) contains all elements of the space of admissible values from the MILP. Therefore every feasible solution (including the optimal one) from the MILP is also feasible solution in this program. This program computes an upper bound of the value of the game.

Chapter 4

Experiments

In this section, we will first describe the game used for testing of described algorithms, present results of experiments and then compare those results to examine scalability of algorithms. We will examine two settings. In the first setting, we want to test correctness of results provided by the MILP, therefore we will compare it's results to results of LP for NFGSS using a zero-sum game. We will also compare both algorithms in terms of time needed to compute a Strong Stackelberg equilibrium to see the practical difference between LP and MILP.

The second setting consists of comparing both variants of upper bound algorithms to MILP representation in terms of computation time and result value (tightness of the upper bound) using a general-sum game.

All algorithms are deterministic, therefore every experiment was run just once, the difference in computation time between two runs of the same algorithm is marginal.

4.1 Transit Game

Transit game [6] (see Figure 4.1) is a game of two players (attacker and defender, in this case the attacker is the follower and the defender is the leader) moving in a square grid (possible actions in every node are: stay in the same field, move to neighboring field - it is possible to move diagonally). The game ends after specified amount of actions have been played by each player. The defender starts in his base - one node in the grid with coordinates given by the game (in Figure 4.1 leader starts at the node marked as "base of the defender"), while the attacker can start in any node of the first column that he chooses. The attacker's goal is to cross the grid (reach any field in the last column, in Figure 4.1 marked as "attacker's goal") without encountering the defender in the possibly lowest amount of steps. The attacker receives a small negative utility for each action played. They encounter if both players play actions leading to the same field, in which case the attacker receives a negative utility while defender a positive utility. The defender's goal is to encounter the attacker before he crosses the grid and then return to his base (if he fails to return to his base by the end of the game, he will receive large negative utility).
In the following experiments, we used two different grid sizes. In the first case, the grid with size $3 \times n$ was used and the length of strategies (number of steps) was set to $n + 1$. In the

Outcome	Utility value of the leader	Utility value of the follower
Follower caught	1	-1
Follower reached last column	-2	2
A step	0.02	-0.02
Leader failed to return to his base	-1000	1000

Table 4.1: Utility values: zero-sum game used in experiments

second case, we used a $2 \times n$ grid and n steps. As for the utility values, we used different values to model two games, one being a zero-sum game and the other being a general-sum game. Utility values for both games are described in Tables 4.1 and 4.2. The experiments were run on a standard PC (dual-core Intel Core i5 460M 2.53 GHz, 4GB RAM), Java (build 1.8.0_31-b13) API of IBM Cplex Optimization Studio 12.4.0

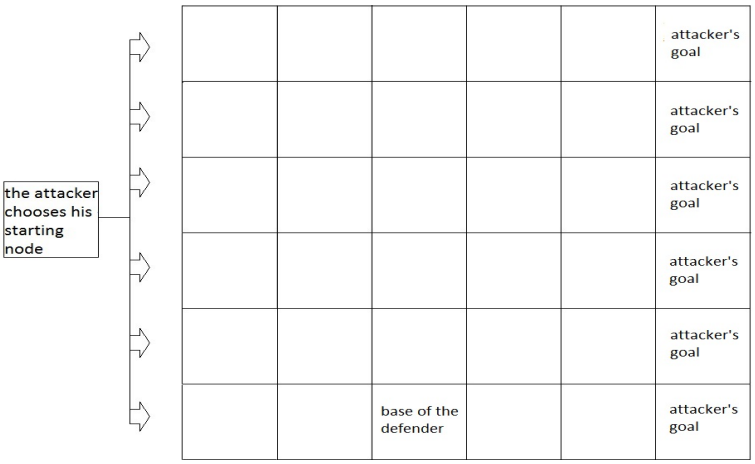


Figure 4.1: Example of transit game

4.2 Zero-sum experiments

We use zero-sum game for this setting, because LP for NFGSS finds Nash equilibrium in zero-sum games (and therefore SSE in zero-sum games), and both players have deterministic actions. The purpose of this experiment is to prove correctness of the results provided by the MILP representation. The result of the MILP should be the same as the one provide by LP for NFGSS. Table 4.3 shows dependence of computing time on n and results of both algorithms on a $3 \times n$ grid and Table 4.4 shows similar outcomes for $2 \times n$ grid. The item "$>$ 24 hours" means, that the computation was terminated after 24 hours.
The summarized dependence of computation time on amount of game states of experiments

Outcome	**Utility value of the leader**	**Utility value of the follower**
Follower caught	1	-10
Follower reached last column	-10	2
A step	0	-0.02
Leader failed to return to his base	-1000	0

Table 4.2: Utility values: general-sum game used in experiments

n	**computation time of LP for NFGSS**	**computation time of MILP**	**result of LP for NFGSS**	result **of MILP**
2	0.265 s	2.168 s	-2	-2
3	0.256 s	742.720 s	-1.4856	-1.4856
4	0.358 s	$>$ 24 hours	-1.1237	N/A
5	0.436 s	$>$ 24 hours	-1.0629	N/A

Table 4.3: Comparison of MILP and LP for NFGSS on a $3 \times n$ grid

on both grids is shown in Figure 4.2. The number of game states is equal to the product of width of the grid, n and number of states (e.g. $3 \times n \times (n + 1)$ for the first grid). Both experiments proved, that MILP provides correct results, but for zero-sum games, LP for NFGSS is significantly faster than the MILP. For grid with 8 nodes and maximum of 4 actions played in each sequence (32 states in each MDP) it took LP 0.374 seconds to find the SSE, while MILP it took MILP more than 28 minutes to find the same solution.

4.3 General-sum experiments

We now turn to the general-sum game in the following experiments. While both upper bound algorithms compute the upper bound even for non-deterministic follower, both players have deterministic actions in this setting. That way, we can compare the tightness of the upper bound of both algorithms on the game, for which the MILP solves the game exactly.

n	**computation time of LP for NFGSS**	**computation time of MILP**	**result of LP for NFGSS**	result **of MILP**
2	0.265 s	0.514 s	-0.8379	-0.8379
3	0.343 s	4.914 s	0.0105	0.0105
4	0.374 s	1708.039 s	0.3374	0.3374
5	0.390 s	$>$ 24 hours	0.3837	N/A

Table 4.4: Comparison of MILP and LP for NFGSS on a $2 \times n$ grid

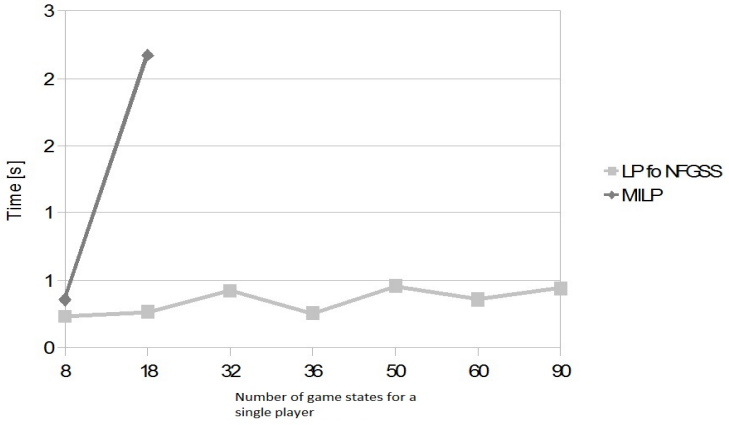


Figure 4.2: Zero-sum experiment: Dependence of computation time on amount of game states

4.3.1 Comparing MILP representation with first variant of the upper bound

In this experiment, we compare MILP with the first variant of upper bound algorithm (UB1). Table 4.5 shows dependence of computing time and calculated value of the game (Expected leader's utility) on n on a 3 x n grid and Table 4.6 shows similar dependence for 2 x n grid. Both experiments showed that while the first variant of the upper bound computation is significantly faster than the MILP, it does not provide any additional information about the game, as the result is always the highest possible leader's utility of all outcomes. For every game, this result is equal to the number of steps, as in every step the highest possible utility for the leader is 1, which corresponds to the encounter of both players. So the strategies of both players lead to their encounter in every action played in this game. This algorithm is the fastest of the presented general-sum algorithms (see Figure 4.3).

4.3.2 Comparing MILP representation with second variant of the upper bound

We now compare MILP with second variant of upper bound algorithm (UB2). Table 4.7 shows dependence of computing time and calculated value of the game (Expected leader's utility) on n on a 3 x n grid and Table 4.8 shows similar dependence for 2 x n grid.
Both experiments showed, that the upper bound computed by this variant is quite tight and is faster than the MILP (it provides a result for $n = 4$ for both grids in less then 14 minutes, while the MILP didn't provide any result in 24 hours), it is significantly slower than the first variant of the upper bound algorithm (see Figure 4.3) due to quadratic amount of constraints based on number of strategies and does not scale well for large games.

n	computation time of UB1	computation time of MILP	Value of the game in UB1	Value of the game in MILP
2	0.405 s	1.248 s	3	-10
3	1.263 s	249.507 s	4	0.206
4	5.132 s	$>$ 24 hours	5	N/A
5	15.459 s	$>$ 24 hours	6	N/A

Table 4.5: Comparison of MILP and the first variant of upper bound calculation on a $3 \times n$ grid

n	computation time of UB1	computation time of MILP	Value of the game in UB1	Value of the game in MILP
2	0.249 s	0.343 s	2	-0.3497
3	0.374 s	2.745 s	3	0.7659
4	0.702 s	775.215	4	0.9905
5	0.967 s	$>$ 24 hours	5	N/A

Table 4.6: Comparison of MILP and the first variant of upper bound calculation on a $2 \times n$ grid

n	computation time of UB2	computation time of MILP	Value of the game in UB2	Value of the game in MILP
2	0.405 s	1.248 s	0	-10
3	2.823 s	249.507 s	0.286	0.206
4	822.655 s	$>$ 24 hours	0.438	N/A

Table 4.7: Comparison of MILP and second variant of upper bound calculation on a $3 \times n$ grid

n	computation time of UB2	computation time of MILP	Value of the game in UB2	Value of the game in MILP
2	0.327 s	0.343 s	0	-0.3497
3	0.530 s	2.745 s	1	0.7659
4	11.606 s	775.215	1.1891	0.9905
5	22.292 s	$>$ 24 hours	1.2893	N/A

Table 4.8: Comparison of MILP and second variant of upper bound calculation on a $2 \times n$ grid

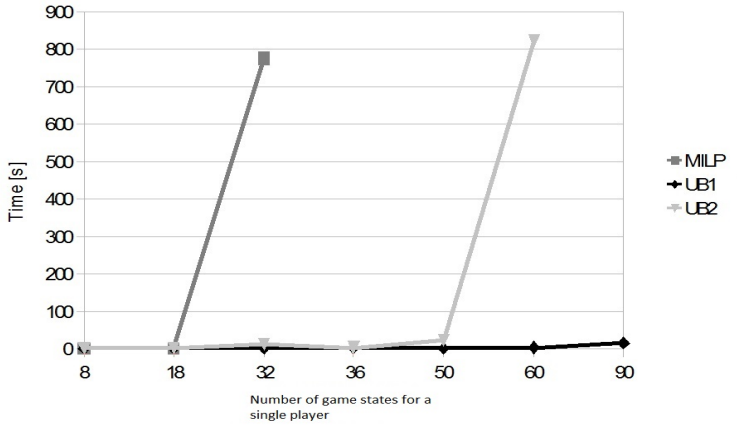


Figure 4.3: General-sum experiment: Dependence of computation time on amount of game states

The experiments show that while MILP solves more general games than LP for NFGSS (general-sum games instead of zero-sum), it does not scale well into large games.

The first variant of upper bound computation solves the problem much faster than the MILP, but the computed result is not tight and therefore it cannot be used as a basis for a double-oracle iterative algorithm, as it does not provide any useful information about the sub-game.

The second variant of the upper bound algorithm might be tight enough, but it is significantly slower than the first variant and is not much faster than the MILP itself. Double-oracle iterative algorithm compute upper bounds for different sub-games repeatedly and needs dramatically faster computation of the upper bound. Therefore, neither this variant can be used to improve the iterative algorithm, because it would not improve the scalability of the algorithm.

Chapter 5

Conclusion

Strong Stackelberg equilibrium (SSE) is a solution concept, in which one player commits himself to play an optimal strategy, such that the other player observes this committed strategy and plays his best response to it. This solution concept is used in many security applications and therefore it has been given lot of focus in the past. While there exists an efficient algorithm for finding SSE in normal-form games, finding SSE in extensive-form games proves to be challenging (there exists an algorithm for solving this problem, it does not, however, scale well into large games).

We focused on finding SSE in normal-form games with sequential strategies (NFGSS), which is a simplified extensive form game, in which the game is formed as separated Markov decision processes (MDPs) of each player rather than one big game tree common to all players. This allows to model games with sequences of actions and stochastic actions, but without player's ability to observe actions of other players throughout the game. This allows simpler representation of strategies of players. Also, MDPs are usually significantly smaller than the game tree. For this class of games, there exists an algorithm finding Nash equilibrium, which generally coincides with SSE in zero-sum games only.

This work presents an algorithm for computing Strong Stackelberg equilibrium in general-sum normal-form games with sequential strategies. We present two formulations of the solution: first solves the problem by computing multiple linear programs (one for each pure strategy of the follower) and chooses the one with best result, the second solves single mixed integer linear program (MILP). Both representations solve the problem exactly, but do not scale well into large games.

In order to use the iterative algorithms for solving the problem, we formulated two algorithms for computing an upper bound of the expected utility value of the leader. The iterative algorithm needs a quickly computed and tight upper bound to effectively prune as many strategies as possible. The experimental results show, that while the first variant of the upper bound algorithm is quick enough, it is not tight and the second variant of the upper bound algorithm is tight, but it does not scale well for large games.

In future work, we will focus on improving the second variant of the upper bound algorithm. Problem with this variant is the size of the LP solving the task, therefore we will investigate the methods for solving also the computation of the upper bound iteratively, which would give us the option of choosing between tight and quickly computed upper bound or finding the compromise between the two.

Bibliography

[1] Conitzer, V., and Sandholm, T. 2006. Computing the optimal strategy to commit to. In EC '06 Proceedings of the 7th ACM conference on Electronic commerce, 82-90.

[2] Koller, D.; Megiddo, N.; and von Stengel, B. June 1996. Efficient computation of equilibria for extensive two-person games. In Games and Economic Behavior Volume 14, Issue 2, 247–259.

[3] Paruchuri, P.; Pearce, J.; Marecki, J.;Tambe, M.; Ordonez, F.; and Kraus, S. 2008. Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games. In AAMAS '08 Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2, 895-902.

[4] Shoham, Y., and Leyton-Brown, K. 2009. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*

[5] Bosansky, B.; and Cermak, J. 2015. Sequence-form algorithm for computing stackelberg equilibria in extensive-form games. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 805-811.

[6] Bosansky, B.; Xin Jiang, A.; Tambe, M.; and Kiekintveld, C. 2015. Combining compact representation and incremental generation in large games with sequential strategies. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 812-818.

[7] Conitzer, V., and Korzhyk, D. 2011. Commitment to Correlated Strategies. In Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, 632-637.

[8] M. Jain, J. Tsai, J. Pita, C. Kiekintveld, S. Rathi, M. Tambe, and F. Ordóñez. 2010. Software Assistants for Randomized Patrol Planning for the LAX Airport Police and the Federal Air Marshals Service. Interfaces, 40:267–290.

[9] J. Pita, M. Tambe, C. Kiekintveld, S. Cullen, and E. Steigerwald. 2011. Guards - game theoretic security allocation on a national scale. In AAMAS '11 The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1, 267 - 290.

[10] von Stengel, Bernhard and Zamir, Shmuel. July 2010. Leadership games with convex strategy sets. In Games and Economic Behavior, Volume 69, Issue 2, 446–457.

[11] Letchford, J., Conitzer, V. 2010. Computing optimal strategies to commit to in extensive-form games. In EC '10 Proceedings of the 11th ACM conference on Electronic commerce, 83-92.

[12] Yin, Z., Tambe, M. 2012. A unified method for handling discrete and continuous uncertainty in Bayesian Stackelberg games. In AAMAS '12 Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, 855-862.

Appendix A

Contents of CD

File gtlibrary.zip contains a Java project containing classes used for computing SSE in NFGSS. The project needs adding a path to CPLEX library to VM arguments to function properly.

Classes related to this work:

- cz.agents.gtlibrary.nfg.MDP.Stackelberg.MDPStackelbergMILP.java - class computing SSE in NFGSS using MILP (function solve()). Run FullCostPairedMDP* with argument "milp".
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.MDPStackelbergUB.java - class computing upper bound in NFGSS using the first variant of upper bound computation (function solve()). Run FullCostPairedMDP* with argument "ub1".
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.MDPStackelbergUB2.java - class computing upper bound in NFGSS using the second variant of upper bound computation (function solve()). Run FullCostPairedMDP* with argument "ub2".
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.MDPMultipleLPs.java - class computing value of the game in NFGSS with fixed pure strategy of the follower (function solve()).
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.MDPExample.java - class used for computing example presented in the project.
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.MDPSubStrategy.java- class used to model strategy in sub-MDP started by a state and an action played.
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.MDPPureStrategy.java - class used to model strategy with several added public functions altering actions probabilities (functions putActionToPureStrategy, removeActionFromPureStrategy, isInPureStrategy).
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.StateTuple.java - class used to model pair of states (one for the leader and one for the follower).

- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.StateActionTuple.java - class used to model pair of states and actions (one for the leader and one for the follower).
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.SlackVariable.java - class used for better addressing of slack variables in cplex.
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.ExpectedUtilityStateVariable.java - class used for better addressing of variables representing expected utility of follower's state in cplex.
- cz.agents.gtlibrary.nfg.MDP.Stackelberg.implementations.ExpectedUtilityActionVariable.java - class used for better addressing of variables representing expected utility of follower's state and action in cplex.

* FullCostPairedMDP reffers to cz.agents.gtlibrary.nfg.MDP.FullCostPairedMDP.java

It is possible to change the setting of the game (grid size, number of steps, utility values) in cz.agents.gtlibrary.nfg.MDP.domain.transitgame.TGConfig.java – grid size and number of steps are constants, utility values can be changed in function getGeneralSumUtility()

In the file export.zip is a directory export, which contains jar file for solving an example (grid 2×3 with 3 steps and general-sum settings). Run it from command line using command "start-SSE option" (option is "milp", "ub1" or "ub2" depending on which algorithm is to be tested). The script needs variables JAVA_HOME (directory to java 8 directory e.g. C:\Program Files\Java\jre1.8.0_31) and CPLEX_BIN (directory to cplex_version.dll e.g. C:\Program Files\IBM\ILOG\CPLEX_Studio124\cplex\bin\x64_win64) to be defined (set JAVA_HOME=...).