ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Rindt** Jméno: **Eduard** Osobní číslo: **406324**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Dynamické programování pro výpočet Stackelbergových strategií v sekvenčních hrách

Název diplomové práce anglicky:

Dynamic Programming for Computing Stackelberg equilibrium in Sequential Games

Pokyny pro vypracování:

Stackelberg Equilibrium (SE) is a solution concept where the leader commits to a strategy that is observed by the follower that plays a best response. There are several known theoretic and algorithmic results for computing Stackelberg equilibria in sequential games. However, the existing algorithms always consider the whole game since SE prescribes to find a global optimum. This is one of the reasons for poor scalability in games with very long (or infinite horizon) since the size of the game grows exponentially with the horizon. The goal of the student is thus to (1) design optimal dynamic programming algorithm for computing SE in finite concurrent-move games, (2) investigate possibilities for computing approximate SE, (3) implement the approximate algorithm, (4) compare the new algorithm with the existing state-of-the-art exact solutions and analyze the trade-off between the scalability and the quality of found strategies.

Seznam doporučené literatury:

[1] B. Bosansky, J. Cermak; Sequence-Form Algorithm for Computing Stackelberg Equilibria in Extensive-Form Games, AAAI 2015
[2] B. Bosansky, S. Branzei, K. A. Hansen, P. B. Miltersen, T. B. Lund; Computation of Stackelberg Equilibria of Finite Sequential Games, ACM TEAC 2017
[3] J. Cermak, B. Bosansky, K. Durkota, V. Lisy, C. Kiekintveld; Using Correlated Strategies for Computing Stackelberg Equilibria in Extensive-Form Games, AAAI 2016
Computing Optimal Strategies with Memory to Commit to in Sequential Games

Jméno a pracoviště vedoucí(ho) diplomové práce:

Mgr. Branislav Bošanský, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Mgr. Branislav Bošanský, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

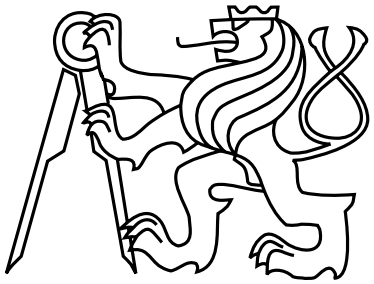
III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

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



Master's Thesis

Dynamic Programming for computing Stackelberg equilibrium in Sequential Games

Eduard Rindt

Supervisor: Mgr. Branislav Bošanský Ph.D.

Study Programme: Otevřená Informatika

Field of Study: Umělá Inteligence

January 8, 2019

Aknowledgements

I would like to give many thanks to my supervisor, Mgr. Branislav Bošanský, PhD., for his guidance and advice he provided me with and for his patience and for his critiques, when I was heading in wrong direction.

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. 2018 .....

Abstract

In Stackelberg games, one agent must commit to a strategy before the other agent compute his, allowing him to always play the best response to first player's strategy. The standard models of computing optimal strategy of the first player are difficult to compute, because they solve a linear program for the whole Game Tree. In this work, we will focus on Stackelberg games and their solution in sequential games using dynamic approach, which can solve potentially infinite games.

Keywords: Game Theory, Stackelberg games, Sequential games, Dynamic programming

Abstrakt

Ve Stackelbergových hrách se musí jeden z agentů přihlásit ke své strategii ještě před tím, než druhý hráč spočítá svou. To mu dává možnost hrát svou nejlepší strategii proti prvnímu hráči. Standartní modely výpočtu optimální strategie prvního hráče v sekvenčních hrách jsou výpočetně náročné, jelikož spočívají ve vyřešení lieárního programu pro celý herní strom. V této práci se zaměříme na Stackelbergovy hry a jejich řešení ve hrách v sekvenční formě s využitím principu dynamického programování, což umožní řešit i potenciálně nekonečné hry.

Klíčová slova: Teorie her, Stackelbergovy hry, hry v sekvenční formě, dynamické programování

Contents

1 Introduction **1**
1.1 Outline . . . 1

2 Game theory **3**
2.1 Utility and utility function . . . 3
2.2 Actions and strategies . . . 4
2.3 Stackelberg games . . . 5
2.3.1 Strong Stackelberg equilibrium . . . 5
2.4 Game classification based on utility dependence . . . 6
2.4.1 Constant-sum games . . . 6
2.4.2 General-sum games . . . 7
2.5 Game classification based on representation . . . 7
2.5.1 Normal-form games . . . 7
2.5.2 Extensive form games . . . 7
2.5.3 Extensive-form games with perfect information and concurrent moves 8

3 Computing Strong Stackelberg equilibrium in Sequential games **11**
3.1 MILP representation . . . 11
3.2 Dynamic approach . . . 13
3.2.1 Sub-game representation . . . 14
3.2.2 Using the representation of sub-games to form a solution . . . 14
3.2.3 Extreme points pruning . . . 15
3.3 Computing the lower bound on the value of Stackelberg equilibrium . . . 22

4 Experiments **25**
4.1 Pursuit-evasion Game . . . 25
4.2 Deterministic utility . . . 26
4.3 Randomly generated utility . . . 26

5 Results **31**

6 Conclusion **33**

A Contents of CD **37**

List of Figures

3.1 Example of extreme points pruning 16
3.2 Example: visualization of pruning phase for follower's action c 22
3.3 Example: visualization of pruning phase for follower's action d 23

4.1 Dependence of the number of generated extreme points on the size of the game-tree in the logarithmic scale 28
4.2 Dependence of the number of pruned extreme points on the size of the game-tree in the logarithmic scale 29
4.3 Dependence of the average error of the approximative algorithms based on the game size in the random setting 30

List of Tables

3.1 Example: utility values of extreme points along with their membership to facets and sub-trees . . . 18
3.2 Example: facets and extreme points generated by the LPs . . . 21
3.3 Example: Generated facets and extreme points after the pruning phase . . . 21

4.1 Dependence of the computation time on the game size for all algorithms . . . 26
4.2 Dependence of value of the utility value of the leader provided by the algorithms on the game size . . . 26
4.3 Dependence of value of the utility value of the leader provided by the algorithms on the game size . . . 27
4.4 Dependence of the number of generated extreme points on the size of the game-tree . . . 27
4.5 Dependence of the number of pruned extreme points on the size of the game-tree 27

Chapter 1

Introduction

In game-theoretical problems, the roles of agents are often not symmetric. Stackelberg game, in which one player called the leader must commit himself to a strategy before the other player (the follower) computes his. That allows the follower to play the best response to the leader's committed strategy. These games are often used to model real-world problems, such as finding the optimal security protocol, which can be observed by possible attackers. Optimal strategy, to which the leader should commit to, is described by the concept of Strong Stackelberg Equilibrium (SSE) [6] and the application of existing algorithms for finding it has seen a lot of success in infrastructure, property protection [1] or economy fields [12].

However, most of the existing algorithms are simplified and do not apply to sequential games [2]. Reason for this is that computation of similar problems in sequential games is often an NP-hard problem [5], even in many one-shot games.

Algorithms based on mathematical programs constructed for the whole game-tree suffer from poor scalability because of the exponential size of the game-tree with the horizon of the game. While there is an algorithm finding SSE by finding Stackelberg Extensive-Form Correlated Equilibrium (SEFCE) [8], which can be found in polynomial time, this algorithm still cannot be used for games with infinite horizon.

The main goal of this thesis is to explore the possibility to find SSE in sequential games with possibly infinite horizon. That can be done by using the dynamic approach and representing the solution of the solved sub-problem in a compact and accurate way, allowing us to use these sub-solutions to construct the solution of the whole game. We will further use this algorithm to develop an approximative algorithm, that would scale well into large problems.

1.1 Outline

In the second chapter of this work, we will introduce essential parts of the game theory. It is necessary for understanding the main goal of this work. We will describe the outcomes of the game, utility function, strategies of players and the concepts of Nash and Stackelberg equilibria using those strategies. Then we will describe several game classes and existing algorithms for solving different types of Stackelberg games, on which will we then focus.

In Chapter 3, we will focus on the problem of finding strong Stackelberg equilibrium in general-sum sequential games. We will modify the existing program solving the correlated version of this problem into a MILP that solves the task exactly, though being an NP-hard problem. We will then focus on the dynamic approach to find SSE in sequential games. As the reachable outcomes in Stackelberg games correspond to strategies of both players, such that follower's strategy is the best response to the leader's strategy, and SSE corresponds to the best of such outcomes (best for the leader), we will organize outcomes of the sub-games into facets corresponding to strategies of the follower (in other words, a point in some facet represents strategy of both leader and best response of the follower, and two different points in the same facets represent two different strategies of the leader, which share the same best response of the follower). We will call boundaries of these facets (points, where the best response of the follower changes) extreme points. These facets from sub-games bear all the necessary information to be transformed into a set of constraints, that we will transform into a polytope form. Extreme points of this polytope form a new facet, which can be used again at higher levels of the game-tree. In the final part of this chapter, we will present the approximative algorithm, that provides a lower bound on the solution, while reducing the memory complexity of the problem.

In the experiment part, we will focus on the precision of this approximative algorithm based on the memory complexity and the randomness rate in utility function values for both players.

Chapter 2

Game theory

Game theory is a mathematical discipline describing the interaction of two or more agents solving a given problem (for the rest of the work referred to as the game) [4]. We will focus on non-cooperative game theory, which means that every agent (often referred to as a 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, the 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 the rest of this work, we will use only the 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 pursuit 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.

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

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

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 the 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 lead to a probability distribution over several possible states (often referred to as stochastic actions, or having the element of nature).

A *strategy* [4] 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* [4].

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* [4].

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 a 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.

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, the best response to a given set of strategies of other players is not unique.

Definition 2.2.1. Nash equilibrium *Nash equilibrium* is a strategy profile $R = (s_1, ..., s_n)$ 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 the main difference between Stackelberg and other games. From now on, we will consider the game of only two players (it can be extended to an 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 the 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. Therefore, the leader needs to commit himself to such strategy, which will with it's best response from the follower yield the maximal utility for the leader. Such strategy along with the corresponding best response of the follower is called *Stackelberg equilibrium.*

2.3.1 Strong Stackelberg equilibrium

We now define *Strong Stackelberg equilibrium* (SSE). We assume, that the follower prefers outcome with the 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.

Definition 2.3.1. Strong Stackelberg equilibrium Strong Stackelberg equilibrium is any strategy profile s, such that $s = \underset{s \in S^*}{\operatorname{argmax}}\, U_1(s)$

In other words, we find SSE as the strategy of the leader, 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 - player 1 plays a or b with probability 0.5 and 2 plays d. Note, that by committing himself to the strategy corresponding to Nash equilibrium, the leader can force follower to reach the Nash equilibrium. Therefore, in SSE, the leader will always get the same or greater utility value than in Nash equilibrium.

	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 [9]), 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.

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*.

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.

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.

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 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.

Given a game in form of the game-tree with root in node n, we call sub-tree with root in node ch, where ch is the child of n in the game-tree, a sub-game of n given by ch. We will also call the set of all sub-games given by children of n sub-games of n.

Example of Stackelberg game in the extensive form is the Transition game, where the follower must travel from one side of the game-plan to the other while avoiding the leader, which tries to catch the follower.

2.5.3 Extensive-form games with perfect information and concurrent moves

We will restrict only on games with *perfect information*, so players will have perfect observation. Therefore, every information set contains exactly one node and we may omit the information sets completely. We will also restrict to such games, in which both players make their actions simultaneously, which means, that in the game tree, child of n is given by pair of actions (one for every player)

We will shortly describe the existing algorithm for finding SSE in finite sequential games with perfect information and concurrent moves [11] as we aim to extend this algorithm. Let us denote N the set of all nodes in the game tree, Z the set of leafs of this tree, $q(n)$ the set of all children of node $n \in N \setminus Z$,$U_l(n)$ and $U_f(n)$ the values of utility function for the leader and the follower in leaf n from Z, $A_l(n)$ and $A_f(n)$ the action sets of leader and follower in node n from $N \setminus Z$, $q(n, a_l, a_f)$ the child of node n reachable by action profile (a_l, a_f), $U_m(n)$ the minmax value of the follower of the sub-game given by node n. Further, considering chance nodes in the game-tree, we denote $C(n, a)$ the probability that action a is played in chance node n, $h(n)$ the set of players making a move in the n (it will always be $\{c\}$ for chance nodes and $\{1, 2\}$ for all other nodes). The existing algorithm consists of linear program in a following form:

$$\max_{p,v} \sum_{n \in Z} p(n)\, U_l(n) \tag{1}$$

s.t.

$$p(r) = 1 \tag{2}$$

$$0 \leq p(n) \leq 1 \quad \forall n \in N \tag{3}$$

$$p(n) = \sum_{n' \in q(n)} p(n') \quad \forall n \in N \setminus Z,\ h(n) = \{1, 2\} \tag{4}$$

$$p(q(n, a)) = p(n)C(n, a) \quad \forall n \in N \setminus Z\ \forall a \in A_c(n)\ , h(n) = \{c\} \tag{5}$$

$$v(n) = p(n)U_f(n) \quad \forall n \in Z \tag{6}$$

$$v(n) = \sum_{n' \in q(n)} v(n') \quad \forall n \in N \setminus Z \tag{7}$$

$$\sum_{a_l \in A_l(n)} v(q(n, a_l, a_{f_1})) \geq \sum_{a_l \in A_l(n)} p(q(n, a_l, a_{f_1}))U_m(q(n, a_l, a_{f_2}))$$
$$\forall n \in N \setminus Z, a_{f_1}, a_{f_2} \in A_2(n) \tag{8}$$

Variables $p(n)$ represent the probability that node n is reached, while variables $v(n)$ represent expected follower's utility in node n.

The objective function (1) we are maximizing corresponds to sum of utilities in the leafs multiplied by the probability, that that leaf is reached. Constraints (3) bound the probability variables to be non-negative, (2) ensures that the root is always reached, (4) and (5) are the network-flow constraints for nodes, in which both players take the action, and for chance node respectively. Constraint (6) defines the expected utility variables of the follower in leafs, while (7) sets these variables in other nodes to be equal to sum of values in it's children. Constraint (9) forces the follower to play his best response to leader's strategy. By formulating that constraint for every pair of follower's actions in every node, it is ensured that the expected utility after playing that action is highest among possible actions, or the probability of that action must be zero (because assuming that it will be played with positive probability p would mean that there is another child with higher expected utility, which played with probability p would yield higher value).

Chapter 3

Computing Strong Stackelberg equilibrium in Sequential games

We will focus on Stackelberg general-sum extensive form games with perfect information and concurrent moves for the rest of this work. In the first part of this chapter, we will describe computing of SSE for sequential games using a Mixed integer linear program (MILP) for the whole game. In the second part, we will do the same using a dynamic approach
In the third and final part of this Chapter, we describe a heuristic algorithm, that finds a lower bound for the utility value of the leader, by introducing pruning of some solutions.

3.1 MILP representation

As the baseline algorithm to compare the results to, we use the modified LP described in the previous chapter. This LP utilizes chance nodes, which do not need, as we focus on games without them, therefore we can omit them as well as functions $C(n, a)$ and $h(n)$ (as in each state of the game both players make a move). This removes constraint (5) completely. However, the previous LP allows the follower to use mixed strategies, which is not desired in our class of games. Therefore we will transform this LP into a MILP (Mixed integer linear program) by adding the constraints using binary variables to ensure that follower always chooses a pure strategy as his best response. Similarly to previous LP, we introduce following variables: $p(n)$ representing a probability that node n from N will be reached, $v(n)$ representing expected follower's utility in node n, and we further introduce binary variables $b(n, a_f)$, which represent, if the action a_f from $A_f(n)$ is the best response of the follower in node n. The MILP has the following form.

$$\max_{p,v} \sum_{n \in Z} p(n)\, U_l(n) \tag{10}$$

s.t.

$$0 \leq p(n) \leq 1 \quad \forall n \in N \tag{11}$$

$$p(r) = 1 \tag{12}$$

$$v(n) = p(n) U_f(n) \quad \forall n \in Z \tag{13}$$

$$p(n) = \sum_{n' \in q(n)} p(n') \quad \forall n \in N \setminus Z \tag{14}$$

$$v(n) = \sum_{n' \in q(n)} v(n') \quad \forall n \in N \setminus Z \tag{15}$$

$$\sum_{a_l \in A_l(n)} v(q(n, a_l, a_{f_1})) \geq \sum_{a_l \in A_l(n)} p(q(n, a_l, a_{f_1}))U_m(q(n, a_l, a_{f_2}))$$

$$\forall n \in N \setminus Z, a_{f_1}, a_{f_2} \in A_2(n) \tag{16}$$

$$b(n, a_f) \in \{0, 1\} \qquad \forall n \in N \setminus Z, a_f \in A_f(n) \tag{17}$$

$$p(q(n, a_l, a_f)) \leq b(n, a_f) \qquad \forall n \in N \setminus Z, a_l \in A_l(n), a_f \in A_f(n) \tag{18}$$

$$\sum_{a_f \in A_f(n)} b(n, a_f) = 1 \qquad \forall n \in N \setminus Z \tag{19}$$

We are maximizing objective function (10), which corresponds to leader's utility values in the outcomes of the game, and constraints (11), (12) and (14) will force values of $p(n)$ to satisfy the constraints of probability distribution, (13), (15) restrict values of variables v to correspond to the utility of the follower while (16) forces the follower to play the best response to the strategy of the leader, similarly to the algorithm described above. Constraints (17), (18), (19) utilize binary variables for follower's actions, forcing him to play only one action in each node by ensuring that if there are two child nodes reached with non-zero probability, they are reached through the same combination of actions.

Note that while in the MILP it is stated that the binary constraints are used for every node, in the implementation, it is used only in those nodes where it is really needed (See algorithm 1). This is done by solving the algorithm iteratively and adding the constraints to every node where the follower randomizes between actions, until there are no such nodes. Only the time of the last run of the MILP is taken into account, therefore we know that only the minimal needed number of these constraints will be used. This extension of the existing algorithm using correlated strategies to find SSE in this class of games [8], which is current

State of the Art. That is why we chose it as baseline algorithm.

Algorithm 1: Baseline algorithm for Solving the Extensive form Stackelberg games with perfect information and concurrent moves

```
1  function ComputeLP (root);
   Input  : Node root
   Output: solved LP
2  LP = formLPwithouBinaryVariables();
3  queue.add(root);
4  while queue is not empty do
5  |  forall Node n in queue do
6  |  |  checkPureStrategy(n, queue, LP);
7  |  end
8  end
9
10
11 function checkPureStrategy (n, queue, LP);
   Input  : Node n – node for which the strategy should be checked
   Input  : queue – queue of nodes yet to be checked
   Input  : LP
12 if isLeaf(n) then
13 |  queue.remove(n);
14 else
15 |  A ← n.getFollowerActions();
16 |  A.filterOutZeroProbbabilities(LP);
17 |  if |A| > 1 then
18 |  |  forall followerAction fa ∈ A do
19 |  |  |  LP.addBinaryVariable(fa);
20 |  |  |  LP.addBinaryVariableToNodeConstraint(fa, n);
21 |  |  end
22 |  else
23 |  |  queue.remove(n);
24 |  |  S ← n.getSuccessors();
25 |  |  queue.addAll(S);
26 |  |  forall Node succ ∈ S do
27 |  |  |  checkPureStrategy(succ, queue, LP);
28 |  |  end
29 |  end
30 end
```

3.2 Dynamic approach

The main idea of this work is to use results computed in sub-games to find the Stackelberg equilibrium. Nash equilibrium of the game given by some node in game-tree can always be computed from the set of Nash equilibria of sub-games given by children of this node (so no

strategy profiles except the Nash equilibrium will ever be needed in the node's predecessor). No such assumption holds for Stackelberg equilibrium, therefore we must represent the result computed in sub-games in a compact way, in which no information about that sub-game is lost.

3.2.1 Sub-game representation

To hold information about sub-games necessary to compute SSE, we represent every node in the game-tree as the set of facets, where each facet represents a strategy of the follower in the sub-game given by this node and all the strategies of the leader, to which this follower's strategy is the best response. Borders of this facets correspond to probability distribution over the leader's actions (which we will denote as leader's simplex), at which the follower's best response changes. We will call those borders extreme points of that facet. In every extreme point, the important information, which can be used to construct solution of the parent node in the game tree, are utility values of that extreme point for both players, leader's simplex in the first step of the strategy represented by that extreme point (which is not needed for the computation of the value, just for reconstructing of the optimal strategy, to which the leader should commit to) and which extreme points belong to the same facet (in other words, which extreme points belong to the strategies, to which the best response of the follower is the same). The leafs of the tree are represented as the facet containing a single point, which is assigned the same utilities as the leaf.

3.2.2 Using the representation of sub-games to form a solution

Now that we have the facets and extreme points for the leafs, we will focus on creating the set of facets in the node of the tree, for which the facets of all it's children are known. Because facets need to reflect strategy of the follower throughout the tree, for every action a_f of the follower and set of child nodes Q_{a_f}, which can be reached by action a_f, new facet will be constructed for every combination of facets such that there is exactly one facet from each child node from the set Q_{a_f}. Selecting one facet in a sub-game corresponds to the leader committing himself to strategy that leads the follower to play strategy represented by this facet in this sub-game. The leader's simplex in that sub-game can be seen as a linear combination of strategies in the extreme points of this facet, and the utility values of both players in such case can be computed by multiplying the utility values of those extreme points by the coefficients given by this linear combination. By taking into account every combination of facets in the sub-games corresponding to the same action of the follower (we need to fix his strategy as the best response to create new facet) and all possible actions of the leader, we can form a new facet representing follower's strategy (the fixed action in this node and the facet combination bears all the information about sub-games) without loss of any information. Formally, given node $n \in N \setminus Z$, for which the set of facets is already computed in all of his children, let us denote $A_f(n)$ the set of follower's actions available in n, similarly $A_l(n)$ set of leader's actions in n, $q(a_l, a_f)$ child node of n reachable by action profile (a_l, a_f) and $F(a_l, a_f)$ set of facets of this child node. Given action a_f from A_f, let us consider a combination of facets s, such that for every action a_l from A_l there is exactly 1 facet from $F(a_l, a_f)$ in s denoted $f(a_l, a_f)$. We will denote the set of all such possible combinations $S(a_f)$. Furthermore, let us denote $E(f)$ the set of extreme points of facet

f and $M_f(a_l, a_f)$ the minimum utility that the follower will surely get in $q(a_l, a_f)$. Given following constraints, we form an LP for every combination of facets s from $S(a_f)$. For this LP, we introduce variables $p(a_l)$, which represent a probability, that the leader plays action a_l from A_l, and variables $c(e)$, which represent the contribution of extreme point e of some facet f to the utility of both players (given a point at the facet $f(a_l)$ from $S(a_f)$, values of these variables of it's extreme points represent a point at this facet as a linear combination of it's extreme points, multiplied by probability $p(a_l)$). These variables are used to form a new solution based on extreme points from sub-games. Extreme points of the polytope of this LP correspond to extreme points of the desired facet and the LP has following form:

$$0 \leq p(a_l) \leq 1 \quad \forall a_l \in A_l(n) \tag{20}$$

$$\sum_{a_l \in A_l(n)} p(a_l) = 1 \tag{21}$$

$$p(a_l) = \sum_{e \in E(f(a_l, a_f))} c(e) \qquad \forall a_l \in A_l \tag{22}$$

$$0 \leq c(e) \leq 1 \qquad \forall a_l \in A_l(n), e \in E(f(a_l, a_f)) \tag{23}$$

$$\sum_{a_l \in A_l(n)} p(a_l) M_f(a_l, a_{f_1})) \leq \sum_{a_l \in A_l(n)} \sum_{e \in E(f(a_l, a_f))} c(e) U_f(e))$$

$$\forall a_{f_1} \in A_f(n) \setminus \{a_f\} \tag{24}$$

Constraints (20) and (21) ensure that variables $p(a_l)$ satisfy constraints of probability distribution. Constraints (22) and (23) represent a possible strategies in subgames. Given that and constraint (24) ensures that the follower is playing his best response. There is no objective function, as we need all extreme points of this polytope, not just one.

We used lrslib [10] to transform an LP in form of those constraints into polytope form and reading the extreme points from that form.

3.2.3 Extreme points pruning

Number of facets and extreme points grows exponentially due to the depth of the tree. Not all the extreme points (and possibly facets) are however needed, some strategies are just dominated and can be pruned. In terms of those extreme points, let us consider the set of all extreme points of all facets, for which the best response of the follower in the computed node (just one action, not considering children) is the same. We will denote this set D. In terms of simplex, any linear combination of those points is reachable for the leader (because the best response of the follower will not change) as long as it doesn't lower the follower's utility, therefore if we find an extreme point $E_1 \in D$ such that there is a linear combination of two other points from D that offers the same or better utility to both players, E_1 can be pruned. If we use follower's and leader's utility of extreme points from D as the coordinates and plot those points into a graph along with their linear combinations, only points from the upper envelope can be preserved without loss of any possible solution to the whole problem (See Figure 3.1.). If all extreme points of an facet are pruned, that facet is pruned as well.

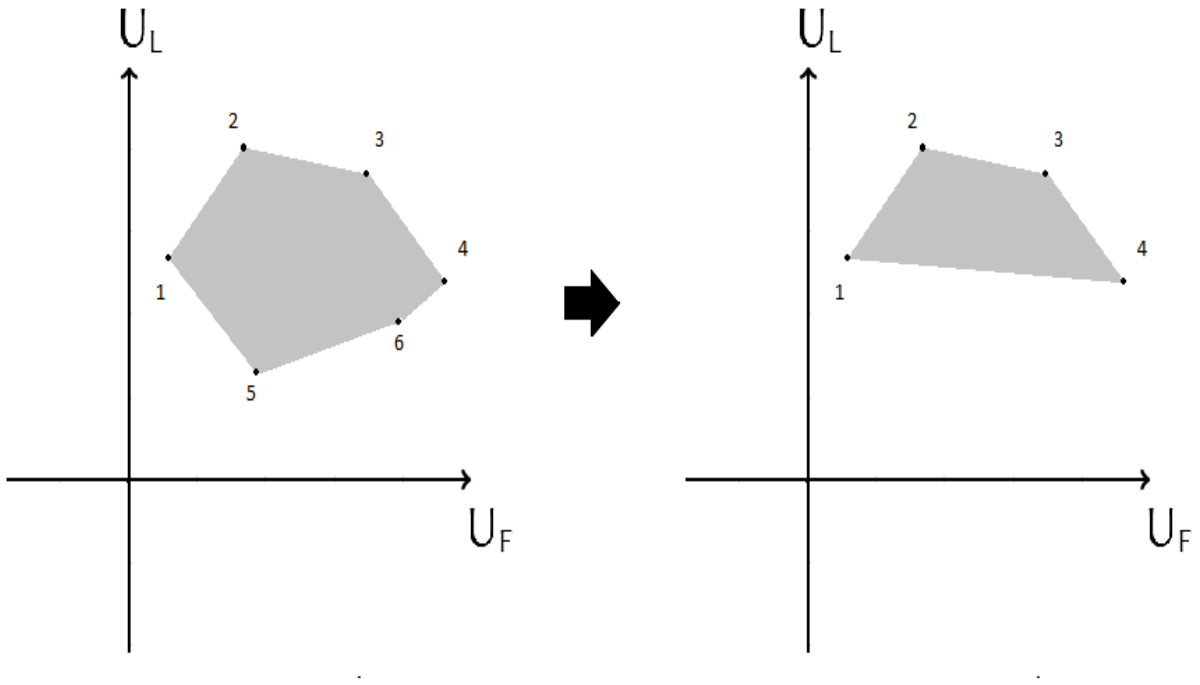


Figure 3.1: Example of extreme points pruning

Algorithm 2: Baseline algorithm for Solving the Extensive form Stackelberg games with perfect information and concurrent moves

```
1  function DynamicApproach (maxDepth, NodesByLevel);
   Input  : int maxDepth - maximal depth of the Tree
   Input  : NodesByLevel[][] - all nodes in the game-tree grouped by the their depth in
            the tree
   Output: Value of the game
2  i ← maxDepth;
3  while i ≥ 0 do
4      NodesAtLevel ← NodesByLevel[i];
5      forall Node n ∈ NodesAtLevel do
6          if isLeaf(n) then
7              E ← createExtremePoint(n.getLeadersUtility(), n.getFollowersUtility());
8              n.createFacet({E});
9          else
10             A ← n.getFollowerActions();
11             forall followerAction fa ∈ A do
12                 FacetSetList F;
13                 S ← n.getAllSuccessorsOfAction(fa);
14                 forall Node n ∈ S do
15                     F.addFacetSet(S.getFacets()).;
16                 end
17                 C ← F.getOneFacetFromEachSet();
18                 forall FacetCombination c ∈ C do
19                     LP ← createLPfromFacets(c);
20                     Points ← LP.transformToPolytope().getExtremePoints();
21                     Points.fillUtilityValues();
22                     n.createFacet(Points);
23                 end
24                 n.prunePoints();
25                 n.removeEmptyFacets();
26             end
27         end
28     end
29 end
30 return root.getBestExtremePoint().getLeaderutility();
```

Example: Now we will show the simple example of how the LPs are created. Let us consider a game in node n, in which $A_l(n) = \{a, b\}$, $A_f(n) = \{c, d\}$. We will denote $q(n, a, c) = s_1$, $q(n, b, c) = s_2$, $q(n, a, d) = s_3$ and $q(n, b, d) = s_4$. There will be 7 facets and 10 extreme points in this example, as depicted in 3.1.

As U_f values of a sub-tree correspond to the minimal utility the follower can get in that sub-tree, $U_f(s_1) = -1$, $U_f(s_2) = 0$, $U_f(s_3) = -2$ and $U_f(s_4) = -4$. Because the follower can play 2 actions, there will be just one best response constraint in every LP.

Now, we will consider action c to be follower's best response and create a linear program

sub-tree	**Facet**	**Extreme point**	**leader's utility value**	follower's utility value
s_1	f_1	e_1	0	1
s_1	f_1	e_2	-1	2
s_1	f_2	e_3	1	-1
s_2	f_3	e_4	1	1
s_2	f_4	e_5	2	1
s_2	f_4	e_6	3	0
s_3	f_5	e_7	2	-2
s_4	f_6	e_8	-1	-3
s_4	f_7	e_9	-2	-4
s_4	f_7	e_{10}	-1	2

Table 3.1: Example: utility values of extreme points along with their membership to facets and sub-trees

for each of 4 facet combinations from sub-trees s_1 and s_2. Starting with facets f_1 and f_3, their LP will have following form:

$$0 \leq p(a) \leq 1$$
$$0 \leq p(b) \leq 1$$
$$0 \leq c(e_1) \leq 1$$
$$0 \leq c(e_2) \leq 1$$
$$0 \leq c(e_4) \leq 1$$
$$p(a) + p(b) = 1$$
$$p(a) = c(e_1) + c(e_2)$$
$$p(b) = c(e_4)$$
$$1\ c(e_1) + 2\ c(e_2) + 1\ c(e_4) \geq -2\ p(a) - 4\ p(b)$$

This will produce polytope with 3 extreme points, first for $p(a) = c(e_1) = 1$, second for $p(a) = c(e_2) = 1$ and the third for $p(b) = c(e_4) = 1$, (which correspond to leader playing one of his pure strategies). Therefore, newly generated facet $f(f_1, f_3)$ will have 3 extreme points with utility values of both players equal to utility values of points e_1, e_2 and e_4 (for the summary of which extreme points and facets were created, see Table 3.2).

Now we move to LP for combination of facets f_1 and f_4. The LP will have following form:

$$0 \leq p(a) \leq 1$$
$$0 \leq p(b) \leq 1$$
$$0 \leq c(e_1) \leq 1$$
$$0 \leq c(e_2) \leq 1$$

$$0 \leq c(e_5) \leq 1$$

$$0 \leq c(e_6) \leq 1$$

$$p(a) + p(b) = 1$$

$$p(a) = c(e_1) + c(e_2)$$

$$p(b) = c(e_5) + c(e_6)$$

$$1\ c(e_1) + 2\ c(e_2) + 1\ c(e_5) + 0\ c(e_6) \geq -2\ p(a) - 4\ p(b)$$

This will produce polytope with 4 extreme points, $p(a) = c(e_1) = 1$, $p(a) = c(e_2) = 1$, $p(b) = c(e_5) = 1$ and $p(b) = c(e_6) = 1$, so this LP will also just copy utility values of all four extreme points e_1, e_2, e_5 and e_6 to the new facet $f(f_1, f_4)$.

We move combination of facets f_2 and f_3.

$$0 \leq p(a) \leq 1$$

$$0 \leq p(b) \leq 1$$

$$0 \leq c(e_3) \leq 1$$

$$0 \leq c(e_4) \leq 1$$

$$p(a) + p(b) = 1$$

$$p(a) = c(e_3)$$

$$p(b) = c(e_4)$$

$$-1\ c(e_3) + 1\ c(e_4) \geq -2\ p(a) - 4\ p(b)$$

The Polytope of this LP will have just 2 extreme points, again corresponding to pure strategies of the leader. So new facet $f(f_2, f_3)$ will contain 2 extreme points.

The final combination of facets for action c as best response is the combination of facets f_2 and f_4.

$$0 \leq p(a) \leq 1$$

$$0 \leq p(b) \leq 1$$

$$0 \leq c(e_3) \leq 1$$

$$0 \leq c(e_5) \leq 1$$

$$0 \leq c(e_6) \leq 1$$

$$p(a) + p(b) = 1$$

$$p(a) = c(e_3)$$

$$p(b) = c(e_5) + c(e_6)$$

$$-1\ c(e_3) + 1\ c(e_5) + 0\ c(e_6) \geq -2\ p(a) - 4\ p(b)$$

This will produce polytope with 3 extreme points, once again just copying utility values of extreme points e_3, e_5 and e_6 to new extreme points in facet $f(f_2, f_4)$.

That is all for the action c of the follower, so now we will move to LPs for action d being the best response of the follower. We start by combining facets f_5 and f_6, which will create a following LP:

$$\begin{gathered}
0 \leq p(a) \leq 1 \\
0 \leq p(b) \leq 1 \\
0 \leq c(e_7) \leq 1 \\
0 \leq c(e_8) \leq 1 \\
p(a) + p(b) = 1 \\
p(a) = c(e_7) \\
p(b) = c(e_8) \\
-2\ c(e_7) - 3\ c(e_8) \geq -1\ p(a) + 1\ p(b)
\end{gathered}$$

This LP is clearly infeasible, which means that this combination of facets will never correspond to any best response of the follower. No facet or extreme points will be generated by this LP, so we move to the last combination of facets, f_5 and f_7:

$$\begin{gathered}
0 \leq p(a) \leq 1 \\
0 \leq p(b) \leq 1 \\
0 \leq c(e_7) \leq 1 \\
0 \leq c(e_9) \leq 1 \\
0 \leq c(e_{10}) \leq 1 \\
p(a) + p(b) = 1 \\
p(a) = c(e_7) \\
p(b) = c(e_9) + c(e_{10}) \\
-2\ c(e_7) - 4\ c(e_9) + 2\ c(e_{10}) \geq -1\ p(a) + 1\ p(b)
\end{gathered}$$

This LP will yield 3 extreme points for the new facet $f(f_5, f_7)$, one of which just copies utilities values of extreme point, e_{10}, but second one corresponds to values of $p(b) = 1, c(e_9) = 1/3, c(e_{10}) = 2/3$ which will create new extreme point e_{11} with $U_l(e_{11}) = 1/3\ (-2) + 2/3\ (-1) = -4/3$ and $U_f(e_{11}) = 1/3\ (-4) + 2/3\ (2) = 0$. The last extreme point e_{12} corresponds to $p(a) = 2/3, p(b) = 1/3, c(e_8) = 2/3, c(e_{10}) = 1/3$, therefore $U_l(e_{12}) = 2/3\ (2) + 1/3\ (-1) = 1$ and $U_f(e_{12}) = 2/3\ (-2) + 1/3\ (2) = -2/3$.

Now that all extreme points are calculated, begins the pruning phase. For action c, all the extreme points were just copied twice into new facets, therefore one copy of each will surely get pruned. Also, since extreme point e_5 fares better (it has better leader's utility value for the sam follower's utility value) than points e_1 and e_4, all copies of those two points will be pruned as well. Facet $f(f_2, f_4)$ will be empty after the pruning and will be removed. For action d, linear combination of copy of point e_{10} and e_{12} (with coefficients $3/4$ and $1/4$) fares better than the point e_{11}, therefore e_{11} will be pruned from the facet $f(f_5, f_7)$. The solution (4 facets, 6 extreme points) after the pruning is written in Table 3.3.

follower's action	**Facet**	**leader's utility value**	follower's utility value
c	$f(f_1, f_3)$	0	1
c	$f(f_1, f_3)$	-1	2
c	$f(f_1, f_3)$	1	1
c	$f(f_1, f_4)$	0	1
c	$f(f_1, f_4)$	-1	2
c	$f(f_1, f_4)$	2	1
c	$f(f_1, f_4)$	3	0
c	$f(f_2, f_3)$	1	-1
c	$f(f_2, f_3)$	1	1
c	$f(f_2, f_4)$	1	-1
c	$f(f_2, f_4)$	2	1
c	$f(f_2, f_4)$	3	0
d	$f(f_5, f_7)$	1	-2/3
d	$f(f_5, f_7)$	-4/3	0
d	$f(f_5, f_7)$	-1	2

Table 3.2: Example: facets and extreme points generated by the LPs

follower's action	**Facet**	**leader's utility value**	follower's utility value
c	$f(f_1, f_3)$	-1	2
c	$f(f_1, f_4)$	2	1
c	$f(f_1, f_4)$	3	0
c	$f(f_2, f_3)$	1	-1
d	$f(f_5, f_7)$	1	-2/3
d	$f(f_5, f_7)$	-1	2

Table 3.3: Example: Generated facets and extreme points after the pruning phase

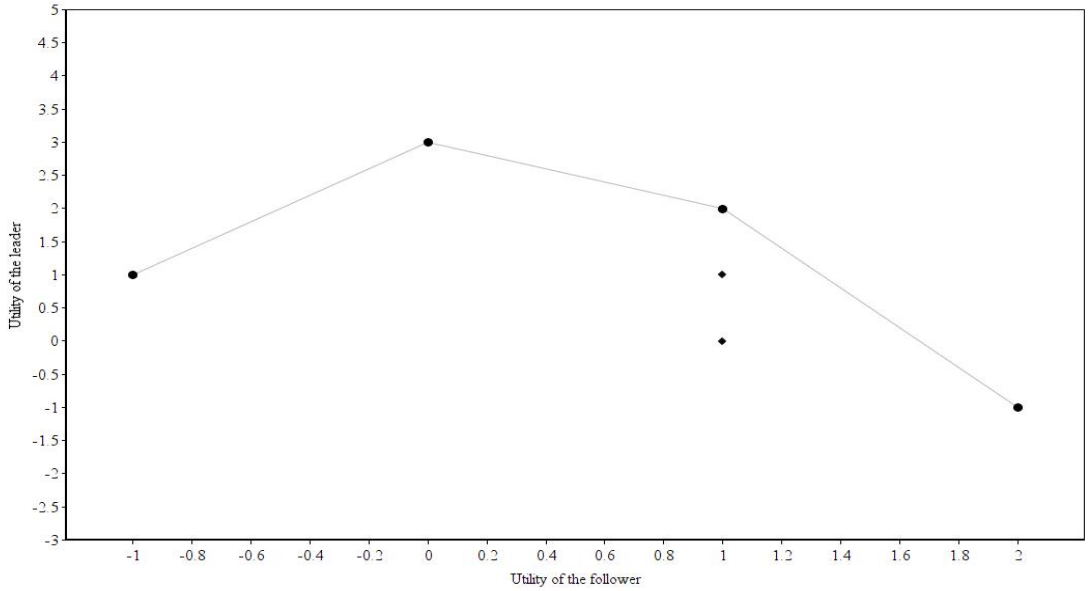


Figure 3.2: Example: visualization of pruning phase for follower's action c

3.3 Computing the lower bound on the value of Stackelberg equilibrium

In this part, we present an heuristic algorithm, which provides a lower bound on the value of the Stackelberg game. The main idea is to use the basic dynamic algorithm as described above, but make the pruning phase more aggressive. Pruning more extreme points of course brings the risk of pruning some points that could be used in the parent nodes to force the follower to play some strategy, that would benefit the leader.

The pruning in this algorithm will only preserve fixed number k (given to the algorithm as parameter) points with the best utility of the leader for each action a_f of the follower. In the exact dynamic algorithm, number of facets and therefore also extreme points can grow exponentially in the higher levels of the game-tree (as the facet represents a strategy of the follower throughout the game, and number of those grows exponentially). The main goal of this approach is to prevent that exponential growth of the number of facets, as the memory complexity might be problematic to deal with for the exact dynamic algorithm.

Note that when we restrict the number of extreme points to k, we restrict also number of facets in each sub-game given by node n to $k|A_f(n)|$, as every facet must contain at least one extreme point, otherwise it's pruned. This removes the exponential size of the problem. Therefore, even for larger k, the memory complexity should much lower for the approximative algorithm than for the exact one in large games.

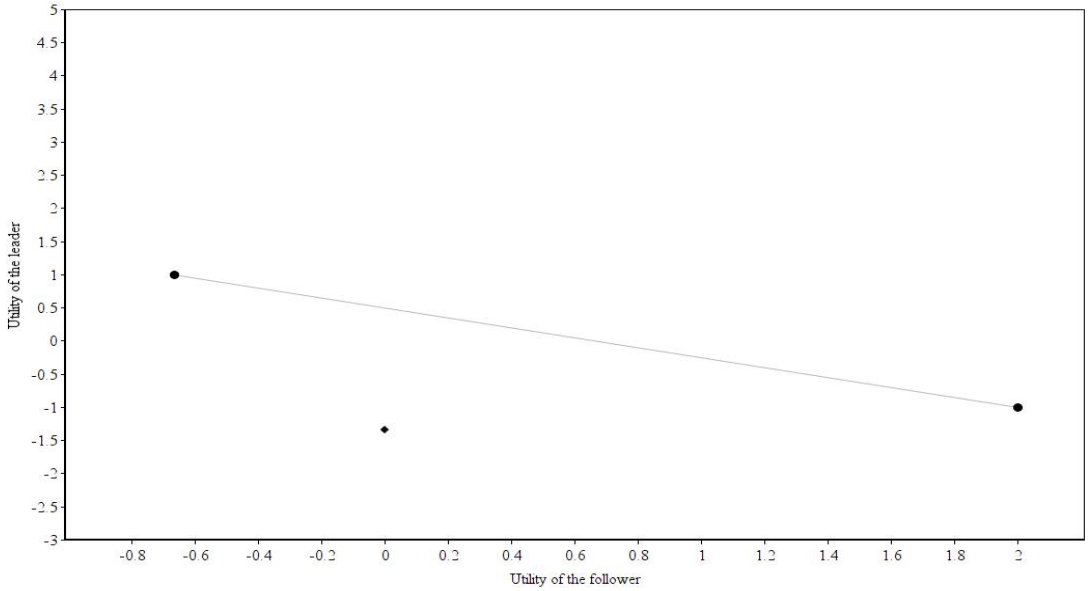


Figure 3.3: Example: visualization of pruning phase for follower's action d

Also, note that the approximative algorithm prunes only the strategy space of the leader, as there are some points left for every action of the follower (and therefore, follower's strategy space is not affected by the pruning). That means, that the algorithm will always be pessimistic. Hence, we have the algorithm that provides the lower bound on the value of the leader in extensive form games with perfect information and concurrent moves, that should address the biggest problem with the exact algorithm.

While the memory complexity of this algorithm is bound to be much better than that of it's exact form, there is no reasonable estimate on how precise lower bound will the algorithm provide or how much faster will it be compared to the exact algorithm. However, the ability to select k should provide some trade-off options between those criteria.

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 and precision of algorithms. We will first focus on testing of the correctness of results provided by the exact algorithm as well as the performance of both exact and approximative algorithms, therefore we will compare it's results to results of the baseline algorithm. In this setting, both utility values and 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. We will then focus on the games with randomly generated utility to better estimate the tightness of lower-bound provided by the approximative algorithm and it's comparison to the exact form in terms of memory complexity. The experiments were run on standard PC, CPU: 4x Intel(R) Core(TM) i7-4710MQ 2.50Ghz, 16GB RAM, C++ compiled with GCC on 64-bit Windows 8.1.

4.1 Pursuit-evasion Game

Pursuit-evasion game [6] (see Figure 4.1) is a game of two players (pursuer and evader, in this case the evader is the follower and the pursuer is the leader) moving in a graph, in our case represented as a square grid (possible actions in every node are: move 1 field in one of 4 directions, if possible). The game ends after specified amount of actions have been played by each player or if both players are at the same position (the evader is caught). Starting positions of players are known to both of them and they get complete information about current whereabouts after each tuple of steps (concurrent move) is played. The pursuer's goal is to catch the evader, in which case he is awarded positive utility, though he gets a large penalty if he fails to catch the pursuer in the specified amount of steps. The pursuer's goal is to escape capture. If the follower is caught, the leader is awarded a positive utility, while the follower gets negative utility. If the follower escapes the leader and is not caught in the specified amount of steps, he is awarded positive utility while the leader gets negative utility. The leader also gets small penalty to the utility for every step he makes, therefore he prefers to catch the follower in least amount of steps possible.

Algorithm	$2 \times 2 \times 5$	$2 \times 2 \times 6$	$2 \times 3 \times 5$	$2 \times 3 \times 6$	$2 \times 3 \times 7$	$3 \times 3 \times 5$	$3 \times 3 \times 6$
Baseline	0.004 s	0.005 s	2.153 s	4.886 s	N/A	N/A	N/A
Dynamic	0.208 s	0.343 s	2.881 s	7.406 s	89.42 s	15.35 s	96.15 s
Dynamic 2	0.123 s	0.243 s	2.455 s	6.971 s	62.73 s	13.26 s	71.13 s
Dynamic 3	0.157 s	0.274 s	2.617 s	7.000 s	67.51 s	14.19 s	78.84 s
Dynamic 4	0.194 s	0.309 s	2.755 s	7.049 s	72.33 s	14.84 s	81.36 s

Table 4.1: Dependence of the computation time on the game size for all algorithms

Algorithm	$2 \times 2 \times 5$	$2 \times 2 \times 6$	$2 \times 3 \times 5$	$2 \times 3 \times 6$	$2 \times 3 \times 7$	$3 \times 3 \times 5$	$3 \times 3 \times 6$
Baseline	8.72	9.242	6.756	7.978	N/A	N/A	N/A
Dynamic	8.72	9.242	6.756	7.978	8.725	4	3.75
Dynamic 2	8.516	8.777	5.904	6.887	7.421	4	3.247
Dynamic 3	8.533	8.868	6.756	7.487	8.348	4	3.75
Dynamic 4	8.72	8.899	6.756	7.978	8.725	4	3.75

Table 4.2: Dependence of value of the utility value of the leader provided by the algorithms on the game size

4.2 Deterministic utility

In the first setup, we tested the game on the grid of different sizes with the leader starting in the left bottom corner (coordinates [0,0]) and the follower starting one-step upwards and to the right of the leader (coordinates [1,1]). If the follower is caught, he gets utility of -8, while the leader is awarded utility of 10. If the follower escapes, leader gets utility of -10 and the follower utility of 8. The penalty for each step is 1 for the leader. Results of these tests are shown in tables 4.1 and 4.2, where in the rows are the computation time and the value of the game provided by algorithms (name *Dynamic* k is used for the approximative algorithm using k best Extreme points for every action of the follower), while in columns is the size of the game (in form $x \times y \times s$, where x, y represent size of the grid and s the maximum number of steps. The approximative algorithm was tested for $k \in \{2, 3, 4\}$. N/A value means that the algorithm was terminated after 2 hours.

4.3 Randomly generated utility

In one of the experimental settings, both players were awarded small positive or negative amount of utility in each step. It was mainly used to determine how precise values the lower bound algorithm provides in the setup with many different extreme points and facets, as the previous test cases contained many extreme points with the same utility values in different sub-games.

In our setup, the random utility in each leaf was generated uniformly in range from -3 to 3 for the leader and from -2 to 2 for the follower. As the values of the game differed in consecutive runs of the algorithm, the measured criteria was the average percentage difference

Algorithm	$2 \times 2 \times 2$	$2 \times 2 \times 3$	$2 \times 2 \times 4$	$2 \times 2 \times 5$	$2 \times 2 \times 6$	$2 \times 2 \times 7$
Dynamic 2	0.76 %	1.31 %	2.13 %	3.47 %	4.67 %	7.89 %
Dynamic 3	0.71 %	1.14 %	1.29 %	2.73 %	3.69 %	4.85 %
Dynamic 4	0 %	0.45 %	0.72 %	1.76 %	3.52 %	3.70 %

Table 4.3: Dependence of value of the utility value of the leader provided by the algorithms on the game size

Algorithm	$2 \times 2 \times 5$	$2 \times 2 \times 6$	$2 \times 3 \times 5$	$2 \times 3 \times 6$	$2 \times 3 \times 7$	$3 \times 3 \times 5$
Dynamic	527	1 092	9 322	36 229	126 404	57 130
Dynamic 2	390	800	5 273	17 304	47 113	30 184
Dynamic 3	440	920	6 129	20 394	47 934	47 934
Dynamic 4	475	996	6 134	22 046	58 946	58 946

Table 4.4: Dependence of the number of generated extreme points on the size of the game-tree

between value of the game (utility value of the leader in SSE) and the solution found by the approximative algorithm over 30 runs. The results are depicted in Table 4.3 and Figure 4.3.

The other criterion which we focused on in this setup was the memory complexity of both the exact dynamic algorithm and it's approximative variants based on how many extreme points they generate and prune during it's run. We used the average values over only 10 runs, and the variance of results was much smaller than in the case of measuring of utility values. the results are rounded up to whole numbers, and can be seen in Table 4.4 and Figure 4.1. As for pruning of the points, results are depicted in Table 4.5 and Figure 4.2.

Algorithm	$2 \times 2 \times 5$	$2 \times 2 \times 6$	$2 \times 3 \times 5$	$2 \times 3 \times 6$	$2 \times 3 \times 7$	$3 \times 3 \times 5$
Dynamic	143	208	2 224	5 919	27 315	7 301
Dynamic 2	96	176	728	2 818	4 978	4 471
Dynamic 3	107	227	915	3 323	5 989	5 180
Dynamic 4	120	258	1 063	3 575	7 806	5 581

Table 4.5: Dependence of the number of pruned extreme points on the size of the game-tree

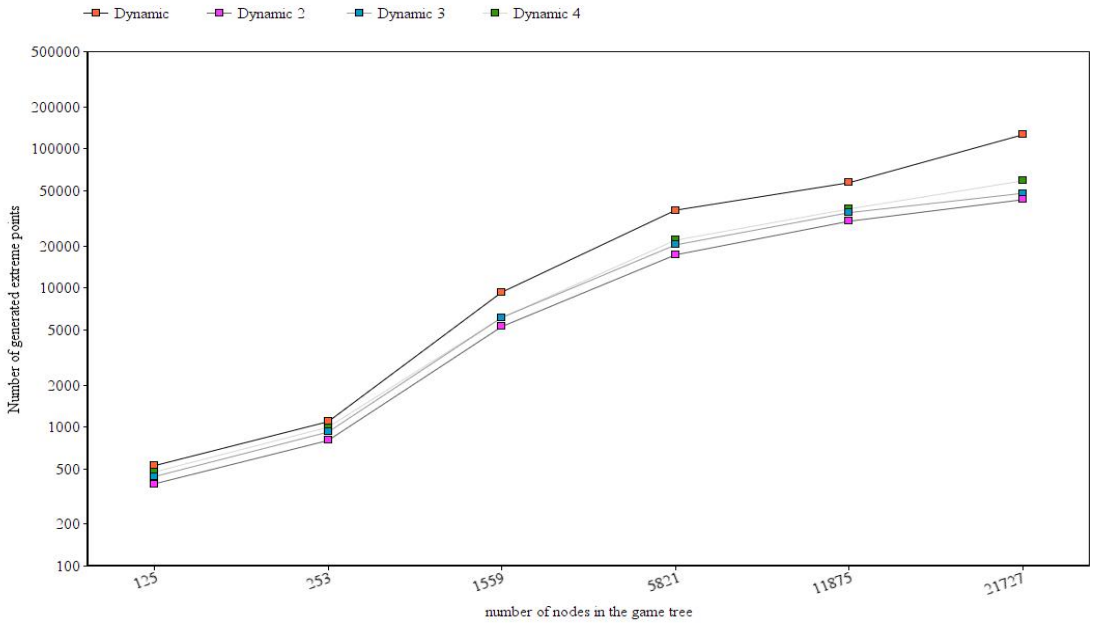


Figure 4.1: Dependence of the number of generated extreme points on the size of the game-tree in the logarithmic scale

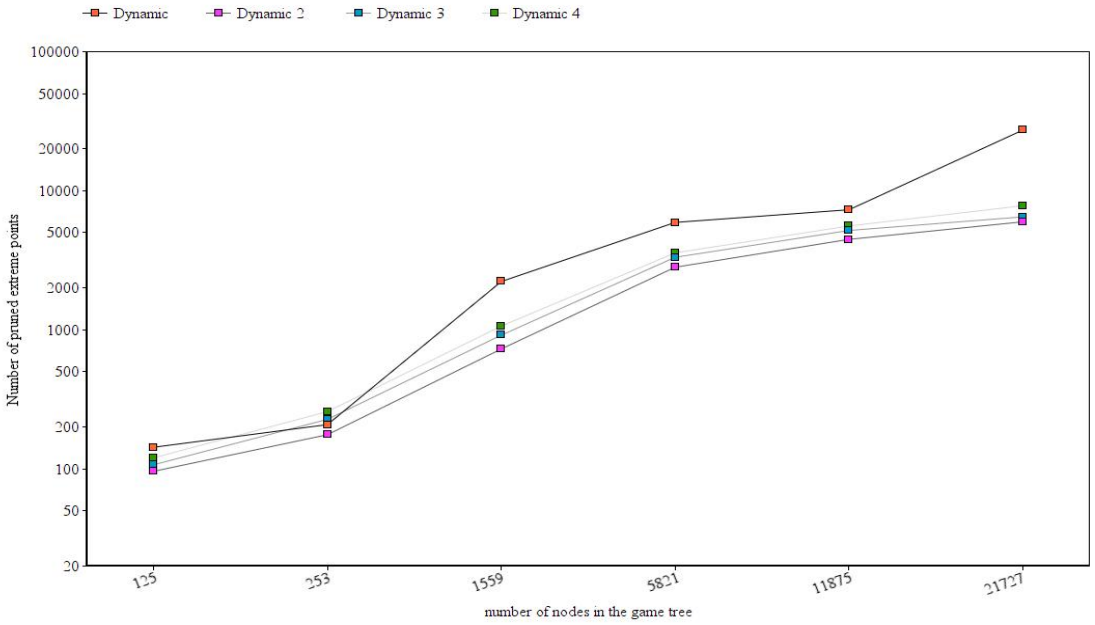


Figure 4.2: Dependence of the number of pruned extreme points on the size of the game-tree in the logarithmic scale

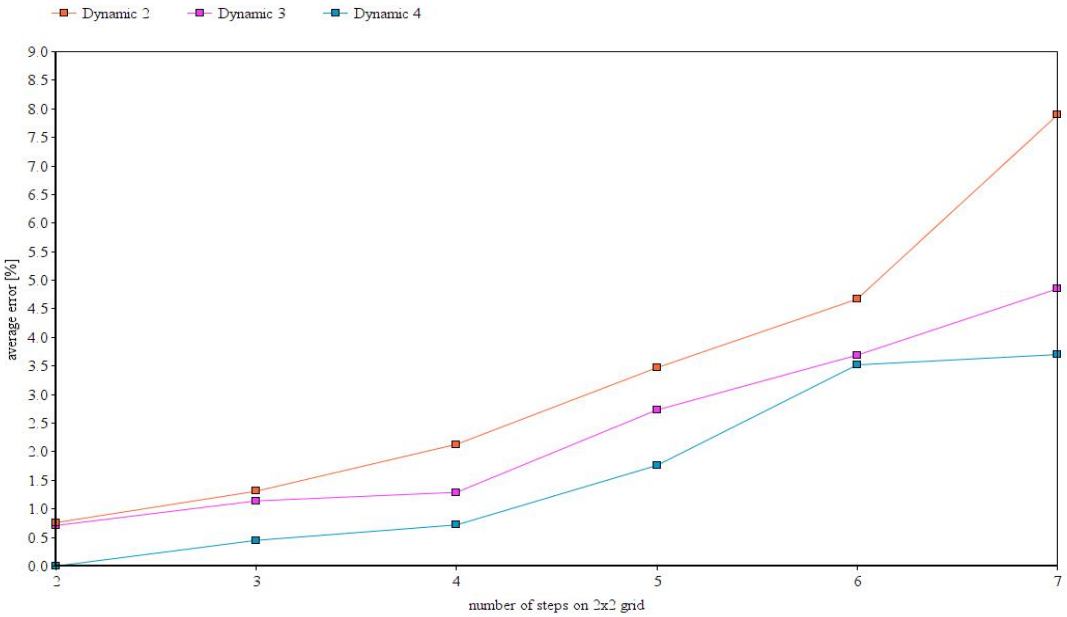


Figure 4.3: Dependence of the average error of the approximative algorithms based on the game size in the random setting

Chapter 5

Results

The experiments show that the baseline algorithm does not scale well into large problems. The exact dynamic algorithm works accurately and can solve larger games than the baseline algorithm.

Approximative algorithm provides trade-off between the computation time and the tightness of the lower-bound on the value of the game. While for $k = 2$ the algorithm was significantly faster than the exact algorithm, it provided very imprecise bound on the solution of larger games, especially in the setting with the randomized utility values. On the other hand, for $k = 4$, the algorithm didn't differ from the exact one much in terms of both computational time and values provided.

Larger problems were not tested, as we represented the game as the whole game-tree (to be able to compare the dynamic algorithm with the baseline) and we encountered memory problems during the larger instances. Also, most of the computation time of the dynamic algorithms on the larger problems were memory operations, especially the approximative algorithm with $k = 2$ spent only about 10% of the computation time by creating the facets and pruning extreme points. If more fitting representation of the game were used, the algorithms would be probably much faster and it would be possible to run the tests on even bigger instances of the game.

Even with the memory problems, the dynamic algorithm computed value of the game for the game-tree containing more than 75 000 nodes in about 96 seconds.

In terms of memory complexity based the number of extreme points created, the exact algorithm produced more than twice the number of extreme points compared to the approximative variant, which was expected, yet it didn't affect the computation time that much, as it wasn't significantly slower than it's heuristic counterparts. It might still be problem in even larger games.

On the other hand, the approximative algorithm with $k = 4$ had an error of about 4% even on larger problems, while computing significantly less extreme points. It seems that the possibility of trade-off between memory complexity of this algorithm and the tightness of it's lower-bound on the solution is working quite well in problems consisting of tens of thousands game-tree nodes.

What stands out is that differences between approximative algorithms for different k in terms of generated and pruned points are rather low. Of course, the difference is still visible,

but not as much as in terms of computational error. Based on this observation, it seems that choosing higher k might significantly improve the tightness of the solution found at the cost of moderately increased memory complexity.

Chapter 6

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 finite sequential games with perfect information and concurrent moves, it cannot be applied to games with infinite horizon.

We focused on finding SSE in such games using the dynamic approach, as that approach can also solve the games with infinite horizon. That was accomplished by representing the solution of the sub-games with set of facets corresponding to strategy profiles sharing the same strategy of the follower (best response to the strategy that the leader has committed to).

We present the exact dynamic algorithm, that finds the Strong Stackelberg Equilibrium and scales into large problems better than the state-of-the-art algorithms for this class of games. We also present approximative version of this algorithm, which provides trade-off between computation complexity and tightness of lower bound on the value of the game.

We encountered memory issues while running tests on larger games, which is caused by comparing baseline and dynamic algorithms on the whole game-tree.

In future work, we will focus on representation of the game in more fitting way for the dynamic algorithm, better utilizing it's advantages, and allowing to solve games with potentially infinite horizon.

Bibliography

[1] Tambe, M. 2011. Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned. Cambridge University Press.

[2] Pita, J.; Jain, M.; Marecki, J.; Ordonez, F.; Portway, C.; Tambe, M.; Western, C.; Paruchuri, P.; and Kraus, S. 2008. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport. In 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 125–132.

[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] Letchford, J., and Conitzer, V. 2010. Computing Optimal Strategies to Commit to in Extensive-Form Games. In 11th ACM conference on Electronic commerce, 83–92.

[6] Leitmann, G. 1978. On generalized Stackelberg strategies. Journal of Optimization Theory and Applications 26(4):637–643.

[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] Jiří Čermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. 2016. Using correlated strategies for computing Stackelberg equilibria in extensive-form games. In Proceedings of AAAI Conference on Artificial Intelligence. AAAI, 439–445.

[9] 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.

[10] http://cgm.cs.mcgill.ca/ avis/C/lrs.html

[11] Branislav Bošanský, Simina Brânzei, Kristoffer Arnsfelt Hansen, Troels Bjerre Lund, and Peter Bro Miltersen. 2017. Computation of Stackelberg Equilibria of Finite Sequential Games. ACM Transactions on Economics and Computation 0, 0, Article 0 (2017), 24 pages

[12] Hamilton, Jonathan, and Steven M. Slutsky. Endogenous timing in duopolygames: Stackelberg or cournot equilibria. Games and Economic Behavior. 1990,Vol. 2, No. 1, pp. 29-46.

Appendix A

Contents of CD

File Stackelberg.zip contains a C++ project containing classes used for computing SSE in Extensive form games with perfect information and concurrent moves (folder ConsoleApplication1). It also contains folder with lrslib and Gurobi libraries, which need to be linked into the project to function properly.

Classes related to this work:

- ConsoleApplication1.ConsoleApplication1.ConsoleApplication1.cpp - main class used for creation of the game and calling the algorithms.
- ConsoleApplication1.ConsoleApplication1.Game.cpp - class containing both algorithms.
- ConsoleApplication1.ConsoleApplication1.Game.h - class containing game specification, such as grid size, number of steps, or if the utility values should be randomized.
- ConsoleApplication1.ConsoleApplication1.Facet.cpp - class representing the facet, containing set of extreme points.
- ConsoleApplication1.ConsoleApplication1.ExtremePoint.cpp - class representing the extreme point.
- ConsoleApplication1.ConsoleApplication1.State.cpp - class representing the state of the game-plan (coordinates of both players and their possible actions).
- ConsoleApplication1.ConsoleApplication1.SubTree.cpp - class representing the solution of the sub-game. It contains set of facets and the pruning algorithm.
- ConsoleApplication1.ConsoleApplication1.TreeNode.cpp - class representing a node in the game-tree.
- ConsoleApplication1.ConsoleApplication1.GridTreeBuilder.cpp - class used to create the game-tree using the specification in Game.h.
- ConsoleApplication1.ConsoleApplication1.BinaryConstraintVars.cpp - class used by the baseline algorithm to store information about binary variables, that have been added to the MILP.

- ConsoleApplication1.ConsoleApplication1.BinaryConstraintMap.cpp - class used by the baseline algorithm to find binary variables for given game-tree node.

In the Folder debug, there is ConsoleApplication1.exe, runnable binary that runs the baseline algorithm, dynamic algorithm, and dynamic approximative algorithm with $k = 2$ on the deterministic setup on the grid of the size 2×2 with maximum of 4 steps.