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

Dynamic Programming for Computing a Stackelberg Equilibrium in Finite Sequential Games with Partial Imperfect Information

Bachelor thesis

David Kraus

Study programme: Open Informatics
Specialization: Artificial Intelligence and Computer Science
Supervisor: Mgr. Branislav Bošanský Ph.D.

Prague, May 2021

Declaration

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

Prague, May 21, 2021 .....

David Kraus

Abstract

There are two types of agents in Stackelberg games called leader and follower, which makes them asymmetrical. The leader has to commit to a strategy before the game begins. The follower observes the leader's strategy before he commits to his own. This work aims to re-implement the existing algorithm for finding Stackelberg equilibria in a programming language Julia, and extend it for computing Stackelberg equilibria in games with partially imperfect information.

Keywords: Game theory, Stackelberg games, Stackelberg equilibrium, Dynamic programming, Imperfect information.

Abstrakt

Stackelbergovy hry jsou asymetrické, protože v nich existují dva typy agentů - leader a follower. Leader se musí před začátkem hry zavázat k určité strategii. Follower pak může jeho strategii pozorovat předtím, než určí svou vlastní. Tato práce má za cíl reimplementovat existující algoritmus pro hledání equilibrií ve Stackelbergových hrách v programovacím jazyce Julia, a následně ho rozšířit pro hry s neúplnou informací.

Keywords: Teorie her, Stackelbergovy hry, Stackelbergovo equilibrium, Dynamické programování, Neúplná informace.

Acknowledgements

I would like to thank my supervisor, Mgr. Branislav Bošanský, Ph.D., for his guidance and patience.

Contents

Chapter 1

Introduction

Game theory is one of the key topics in computer science. There is a lot of real-life applications, from simple games like Rock paper scissors to various scientific fields, including economy [1], biology [2] or security [3].

The game theory studies situations where multiple subjects (called agents) interact with each other. We are often interested in finding states where all of the agents are content with the status quo. These states are called equilibria.

According to the number of agents or whether they have the same role in the game, the games can be divided into various categories. In this work, we will restrict ourselves to Stackelberg games. These games are asymmetrical, meaning that different agents have different options on how to act. More specifically, in Stackelberg games, one agent (called leader) commits himself to some strategy. The rest of the agents (called followers) will observe that strategy before committing themselves. We will be interested in finding Stackelberg equilibria (SE) [4] in these games.

The games can be further divided into games with perfect and imperfect information. In games with perfect information, each player always knows which state he is located in. On the contrary, there can be some degree of uncertainty in games with imperfect information, represented with information sets. Intuitively, the player can not distinguish between game nodes located in one information set. In this work, we will restrict to games where follower always has perfect information (each follower's information set contains only one node). Furthermore, we will restrict to games with perfect recall, which means that the leader can remember which actions he has already taken in the game. This stetting may, for example, apply in security tasks [5], where it is reasonable to assume

that the attacker has a perfect information. The perfect recall is also a logical assumption in real-world applications.

This work will first describe an existing algorithm for finding Stackelberg equilibria, which will be used as a baseline. This algorithm finds the equilibria by constructing and solving a single mixed-integer linear program (MILP). However, this algorithm does not apply to large games. The size of the game tree grows exponentially with its depth, which results in huge demands on memory.

This work aims to devise and implement a more scalable algorithm for computing Stackelberg equilibria in games with imperfect information. The new algorithm will be using dynamic programming (DP). The dynamic programming is based on solving only small segments of the game one by one, which reduces the demands on the memory, and thus the MILP-based algorithms can never be successfully applied to games with very long (or even infinite) horizon.

1.1 Outline

In the second chapter, we will focus on some of the main concepts from the game theory. We will introduce the normal form representation as a basic game representation and the extensive form representation, which will be mostly used in this work. We will further introduce the Stackelberg games and Stackelberg equilibria, which is the main topic of this work.

In chapter three, we will introduce two algorithms for finding Stackelberg equilibria in extensive form games. The first one will be based on solving one mixed-integer linear program (MILP). The second one will be using dynamic programming (DP).

In the fourth chapter, we will extend the existing DP algorithm to be able to find Stackelberg equilibria in games with imperfect information. We will also introduce another baseline MILP algorithm.

In the fifth chapter, we will benchmark the two algorithms for finding Stackelberg equilibria in games with imperfect information.

Chapter 2

Game Theory

Game theory is a mathematical study with numerous areas of application, including computer science [6], biology [2] or economy [1]. It concerns the interaction of two or more independent agents. In this work, we will only consider self-interested agents, which means that their only aim is to maximize their own profit. The agent's profit will be quantified with a utility function, which is a mapping from the states of the game to some real values. In the game theory, agents impact each other with their actions, so they have to consider each other's expected behavior in the game. [7]

2.1 Normal Form

The normal form is probably the best known way to represent a game. The formal definition [7] follows:

Definition 2.1.1 (Normal-form games)**.** A normal-form game is a tuple (N, A, u), where:

- N is a finite set of n players;
- A is a Cartesian product of actions available to each agent (player's i actions are denoted as A_i). Each vector $a = (a_1, ..., a_n) \in A$ is called action profile;
- u is a set of utility functions $(u_1, ..., u_n)$, where $u_i : A \mapsto \mathbb{R}$, is the utility function of player i.

The normal form can be represented with n-dimensional matrix, where n is the number of agents. For two player games with players i, j, each row corresponds to player i actions,

and each column corresponds to player j actions. For $a_x \in A_i$ and $a_y \in A_j$, the cell with indices (i, j) corresponds to the outcome in that action profile.

2.1.1 Strategies in Normal Form Games

One way how an agent can play is to select one single action deterministically. This is called a *pure strategy*. If each agent chooses a pure strategy, we call it a *pure strategy profile*. In that case, each agent's payoff is defined by his utility function.

There is also another type of strategy, called *mixed strategy*. In a mixed strategy, the player can randomize, and his strategy is a probability distribution over his actions. Similarly to the pure strategy case, the *mixed strategy profile* is a Cartesian product of each player's mixed strategy. We can denote the probability that action a_i will be played in mixed strategy profile S_i as $s_i(a_i) \in \langle 0, 1 \rangle$. The set of actions $(a_n \mid s_i(a_n) > 0)$ is called *support* of strategy profile s_i.

In a mixed strategy profile, we introduce *expected utility* to express the payoff of an individual player.

Definition 2.1.2 (Expected utility)**.** For player i in a mixed strategy profile $s = (s_1, ..., s_n)$, the expected utility is defined as

$$u_i(s) = \sum_{a \in A} u_i(a) \prod_{i=1}^{n} s_j(a_j)$$

Informally speaking, the expected utility is a sum of utilities of all possible action profiles, multiplied by the probability that this action profile will occur.[7]

2.1.2 Best Response

In a normal form game (N, A, u), we will define $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_n)$ as a strategy profile of all agents except of agent i. If player i could observe s_{-i}, his decision making would be straightforward - he could calculate exactly which strategy s_i would provide him the biggest payoff. That strategy is called *best response*. Formally, we define player's i best response to the strategy profile s_{-i} as a strategy $s_i^* \in S_i$ such that player's i utility $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$. [7]

2.1.3 Minmax strategy

In a two-player game where $N = (i, -i)$, the minmax strategy of player i denoted as s_i is defined as $\arg\min_{s_i} \max_{s_{-i}} u_{-i}(s_i, s_{-i})$. It is such a strategy that minimizes players $-i$ maximum payoff. If player i plays the minmax strategy, he is not interested in maximizing his own payoff. His only aim is to punish the player $-i$. [7]

2.2 Extensive Form games

Even though the normal form is the basic way to represent the game and any game can be represented in a normal form, in this work, we will mostly be using a different representation called *extensive form*, which is an effective way to represent sequential games. Informally speaking, games in the extensive form are trees with utility values stored in their leaves. The nodes of the tree represent choices of agents, and the edges correspond to the possible actions.

2.2.1 Extensive form with perfect information

This section will introduce a special case of two-player games in extensive form with perfect information and concurrent moves, which means that agents know which state they are in, and the players act simultaneously in each node. In the following section, we will extend this definition to games with imperfect information.

Definition 2.2.1 (Extensive-form games with perfect information)**.** The game in extensive form is a tuple $G = (N, A, H, Z, \chi, \rho, \sigma, u)$ [7], where

- N is a set of n players (in this work always two);
- A is a set of actions;
- H is a set of non-terminal nodes;
- Z is a set of terminal nodes;
- $\rho : H \mapsto N$ is a mapping from non-terminal nodes to players. It determines which players can act in given node;
- χ is a mapping from H to the set of possible actions;

- $\sigma : H \times A \mapsto H \cup Z$ is a mapping from non-terminal nodes and action to the successor (non-terminal or terminal) nodes;
- $u = (u_1, ...u_n)$, where $u_i : Z \mapsto \mathbb{R}$ is an utility function of player i.

Extensive form games with concurrent moves

Extensive form is often used to represent games where agents act separately, and each node is mapped to only one of the agents - agent 1 acts in the root (he chooses one of the successor nodes), agent 2 acts in that chosen node, etc. However, in this work, we will consider a slightly different type of extensive form with *concurrent moves*, which means that agents do not act separately. As we mentioned before, in this work, we will restrict to two-player games, and both of them will participate in each move. A successor node reached from node n by an action profile (a_l, a_f) is denoted as $q(n, a_l, a_f)$. [8]

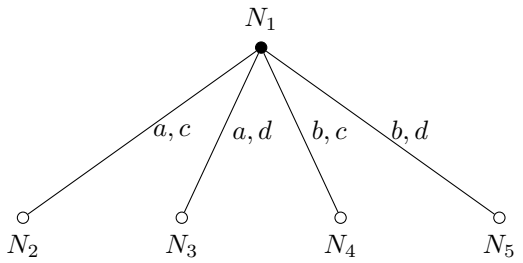

Figure 2.1: Extensive-form game with concurrent moves

	c	d
a	$q(n, a, c) = N_2$	$q(n, a, d) = N_3$
b	$q(n, b, c) = N_4$	$q(n, b, d) = N_5$

Table 2.1: Node of an extensive form game

2.2.2 Extensive form with imperfect information

In the games with concurrent moves, both the leader and the follower know which node they are located in. On the contrary, in games with imperfect information, the players only have the information about which information set they are located in. The game nodes are partitioned into these information sets. If two nodes belong to the same information set, the player can not distinguish between them.

The formal definition follows. [7]

Definition 2.2.2 (Sequential-form games with imperfect information)**.** The extensive-form imperfect-information game is a tuple $(N, A, H, Z, \chi, \rho, \sigma, u, I)$, where:

- $(N, A, H, Z, \chi, \rho, \sigma, u, I)$ is a perfect-information game in extensive form from definition 2.2.1; and
- $I = (I_1, ..., I_n)$, where $I_i = (I_{i,1}, ..., I_{i,k_i})$ is a set of equivalence classes on $h \in H$ (non-terminal nodes), where $\rho(h) = i$ (player i can act in that node), with the property that $\rho(h) = \rho(h')$ (the same players can act in both h and h') and $\chi(h) = \chi(h')$ (the available actions are the same in h and h') whenever there exists a j for which $h \in I_(i, j)$ and $h' \in I_(i, j)$.

Note that each player has its own information sets. It means that one player could be able to distinguish between two given nodes that are indistinguishable for the other player. In this work, we will restrict to the games where the follower can distinguish between any two nodes, i.e., all of his information sets contain only one node.

2.2.3 Strategies in games with imperfect information

In the games with concurrent moves, the player's i strategy in node n, $i \in \rho(n)$, was a probability distribution over all available actions from that node.

In imperfect-information games, player i can not distinguish between nodes from the same information set. By definition, the set of available actions is the same for each game node from one information set. We will define the mixed strategy in a game with imperfect information as a probability distribution in each information set over all actions available in that set.

Perfect recall

The imperfect-information games with perfect recall are a subset of games with imperfect information. In the general imperfect-information games, there are no restrictions about which nodes can belong to which information sets. On the other hand, in games with perfect recall, players can remember which actions they have taken so far.

We will use the notation $seq_i(n)$, where i is a player and n is a game node, to denote the sequence of actions player i has to perform to reach node n. In games with perfect recall, a node n_1 can not be in the same information set as a node n_2 if $seq_i(n_1) \neq seq_i(n_2)$.

It also means that for any two nodes n_1 and n_2 belonging to the same (player's i) information set I_i, $seq_i(n_1) = seq_i(n_2)$. That is why we can overload the notation - $seq_i(I_{i_1})$ denotes the sequence that player i has to take to reach the information set I_{i_1}.

Note that two nodes, n_1, n_2, where $seq_i(n_1) = seq_i(n_2)$, do not necessarily need to belong to the same information set. The perfect recall is rather the minimal knowledge that both players surely have.

In this work, we will restrict to games with perfect recall.

2.3 Stackelberg games

In Stackelberg games, the roles of players are not symmetrical. Generally, there is one leader and $n-1$ followers in *n-player Stackelberg game*, even though we will restrict to two-player games with one leader and one follower in this work. The role of the leader is to commit to some strategy $s_1 \in S_1$. The follower will then observe the leader's strategy before choosing his strategy $s_2 \in S_2$.

2.3.1 Stackelberg equilibrium

Stackleberg equilibrium (SE) is a solution concept in Stackleberg games. Two strategies are in Stackleberg equilibrium if two conditions hold:

- the follower's strategy s_2 is a pure best response to the leaders strategy s_1, and
- the leader's expected utility is the maximum possible, considering that the follower's strategy will be the best response, i.e. $\forall s'_1 \in S_1; \forall s'_2 \in S_2$ such that s'_2 is a best response to s'_1, it holds that $u_1(s_1, s_2) \geq u_1(s'_1, s'_2)$ [9]

In this work, we will restrict to a *strong Stackelberg equilibrium*. It means that if a follower has two or more possible best responses providing him the same payoff, he will choose the one which provides the maximal payoff to the leader. The leader is aware of this follower's behavior.

The follower's role is quite straightforward because he is only aiming to maximize his own payoff. On the other hand, the leader has to consider both players' payoffs. Since he is self-interested, his only aim is to maximize his own payoff, but he has to take into

account that after making a commitment to some strategy, the follower will be free to maximize his own payoff. We will demonstrate this on an example in 2.2.

Leader \ Follower	C	D
A	(2, 0)	(0, 1)
B	(0, 1)	(-1, -1)

Table 2.2: Stackleberg game

In this simple Stackelberg game, the leader's payoff would be maximal with a pure strategy profile (A, C). However, if he committed himself to a pure strategy A, the follower's best response would be D, decreasing the leader's payoff to 0. The leader's expected utility can't be greater than 0 as long as the follower's best response is D. For that reason, the leader has to commit to such a strategy that makes C a follower's best response. This constraint can be expressed with an inequation:

$$s_l(A)u_f(A, C) + s_l(B)u_f(B, C) \geq s_l(A)u_f(A, D) + s_l(B)u_f(B, D)$$

In this case, if $s_l(A) = 2/3$ and $s_l(B) = 1/3$, the left side is equal to the right side and both C and D are follower's best responses. Since the strong Stackleberg equilibrium holds, the follower will play a pure strategy C, providing leader's expected utility $u_l = 4/3$.

2.3.2 Stackleberg equilibrium in imperfect-information games

In concurrent-moves games, the leader was able to commit to a different strategy in each game node. That allowed him to discourage the follower from going into nodes which did not benefit him - he simply committed to the minmax strategy in those nodes.

In games with imperfect information, the leader's strategy has to be the same for all nodes from a given information set. For that reason, the same strategy, which is supposed to bring him the best possible utility in one node, must also work as the punishment strategy in the other nodes in order to prevent the follower from going into that nodes.

Chapter 3

Computing SE in concurrent-moves games

This chapter will introduce two algorithms for finding strong Stackelberg equilibrium in two-player, sequential form games with concurrent moves and perfect information. The first algorithm constructs a single mixed-integer linear program (MILP) [8], while the second one is based on linear programming [10].

3.1 MILP algorithm

First, we will introduce the mixed integer linear program:

Sets

N	set of players (lower index l denoting leader, f denoting follower)
H	set of non-terminal game nodes (r denotes the root node)
Z	set of terminal game nodes
$A_i(n)$	set of actions of player i in node n
$q(n)$	set of successor nodes of node $n \in H$

Variables

$p(n)$	$n \in H \cup Z$	probability of node n being reached during the game
$v_i(n)$	$i \in N; n \in H \cup Z$	player's i expected utility in node n
$U_i(n)$	$i \in N; n \in Z$	player's i utility in terminal node n
$M(n)$	$n \in H \cup Z$	follower's minmax utility in node n
$q(n, a_l, a_f)$	$n \in H \cup Z; a_i \in A_i(n)$	node reached from node n by action profile (a_l, a_f)
$b(n, a_f)$	$n \in H \cup Z; a \in A_f(n)$	binary variable assigned to each follower's action in each node

$$\max_{p,v} \sum_{n \in Z} p(n) U_l(n) \tag{3.1}$$

Subject to:

$$0 \leq p(n) \leq 1 \qquad \forall n \in H \cup Z \tag{3.2}$$

$$p(r) = 1 \tag{3.3}$$

$$v_f(n) = p(n) U_f(n) \qquad \forall n \in Z \tag{3.4}$$

$$p(n) = \sum_{n' \in q(n)} p(n') \qquad \forall n \in H \tag{3.5}$$

$$v_f(n) = \sum_{n' \in q(n)} v_f(n') \qquad \forall n \in H \tag{3.6}$$

$$\sum_{a_l \in A_l(n)} v_f(q(n, a_l, a_{f_1})) \geq \sum_{a_l \in A_l(n)} p(q(n, a_l, a_{f_1})) M(q(n, a_l, a_{f_2})) \qquad \forall n \in H; a_{f_1}, a_{f_2} \in A_f(n) \tag{3.7}$$

$$b(n, a_f) \in 0, 1 \qquad \forall n \in H; a_f \in A_f(n) \tag{3.8}$$

$$p(q(n, a_l, a_f)) \leq b(n, a_f) \qquad \forall n \in H; a_l \in A_l(n); a_f \in A_f(n) \tag{3.9}$$

$$\sum_{a_f \in A_f(n)} b(n, a_f) = 1 \qquad \forall n \in H \tag{3.10}$$

The objective function (3.1) is maximizing the utility of the leader, expressed as a sum of the leader's utility in individual terminal nodes, multiplied by the probability of reaching those nodes in the game. The constraint (3.2) states that the probability of reaching some node can not be less than 0 and greater than 1, and (3.3) assigns the 1 probability of reaching the root node, i.e., the root is certainly reached in every game. Constraint (3.4) defines a follower's expected utility in terminal node n as a product of utility value of n, and the probability of reaching it. The follower always prefers nodes

with higher value v_f. In (3.5), the probability of reaching node $n \in H$ is equal to the sum of probabilities of reaching its successor nodes. It means that the probability of reaching a node n' is not greater than the probability of reaching its predecessor. The next constraint states the same thing about the follower's expected utility. Constraint (3.7) ensures that the follower always plays his best response. The leader can force the follower to play certain action, denoted as a_{f_1}, by playing a minmax strategy in all the nodes except the one he wants to get in. The left side of the inequation is a sum of follower's expected utility after playing a_{f_1} (note that the expected utility is already multiplied by the probability in (3.4)). The right side is the sum of the follower's minmax utilities in node reached by action profile (a_l, a_{f_2}), multiplied by the probability of leader's actions a_l (note that leaders strategy is fixed). The constraint states that the left side is greater or equal to the right side, so a_{f_1} is the best response. The last three constraints introduce the binary variables to ensure that follower's plays pure responses. (3.8) assigns a binary variable to each follower's action in each node. The next constraint (3.9) states that the probability of reaching a successor of node n with action profile (a_l, a_f) $(q(n, a_l, a_f))$ is less or equal to the binary variable $b(n, a_f)$, i.e. if $b(n, a_f) = 0$, the node $q(n, a_l, a_f)$ can't be reached. The last constraint, (3.10), ensures that exactly one binary variable assigned to node n is equal to one.

3.1.1 Algorithm for adding binary variables iteratively

The linear program can be simplified by leaving out the binary variables. [11] It is presumable that the follower's strategy in most of the nodes will be pure even without them. After solving the LP without binary variables, it is necessary to check the follower's strategy and the binary variables to the nodes where the strategy is not pure, then solve

the LP again.

Algorithm 1: Iterative MILP algorithm

Input: $GameTree$

Result: $LeadersMaximalExpectedUtility$

```
1  LP = formLPwithoutBinaryVariables(GameTree)
2  done = False
3  while (!done) do
4      done = True
5      solve(LP)
6      Q = queue
7      Q.push(GameTree.root)
8      while (!Q.isempty) do
9          N = Q.pop
10         if N.isTerminal then
11             Continue
12         end
13         if (followerStrategyInNodeIsMixed(N)) then
14             addBinaryConstraint(N, LP)
15             done = False
16         end
17         Q.push(N.succesors)
18     end
19 end
```

3.2 Dynamic programming algorithm

The main disadvantage of the MILP (mixed-integer linear program) algorithm is that it needs to work with the whole game tree at the same time. In this chapter, we introduce another algorithm based on dynamic programming (DP). This algorithm can traverse the game tree with a depth-first-search (DFS). It starts with the leaves of the game tree and propagates all the necessary information to the predecessor nodes.

3.2.1 Best response facets

As we mentioned before, in a Stackelberg equilibrium, the follower always plays the best response to the leader's (generally mixed) strategy. We will use the game from 2.2 as an example. In figure 3.1, the horizontal axis represents the probability of a leader's actions in his mixed strategy s_l (0 stands for pure strategy A; 1 means pure B). The vertical axis represents the follower's payoff. The red and blue lines stand for follower's actions. The best response is such an action that brings the follower maximum payoff. Informally, one follower's action is the best response in case that it is plotted above the other action, which means that it brings the follower a better payoff for a given leader's strategy. The leader's expected utility after the follower's best response is plotted with the green lines.

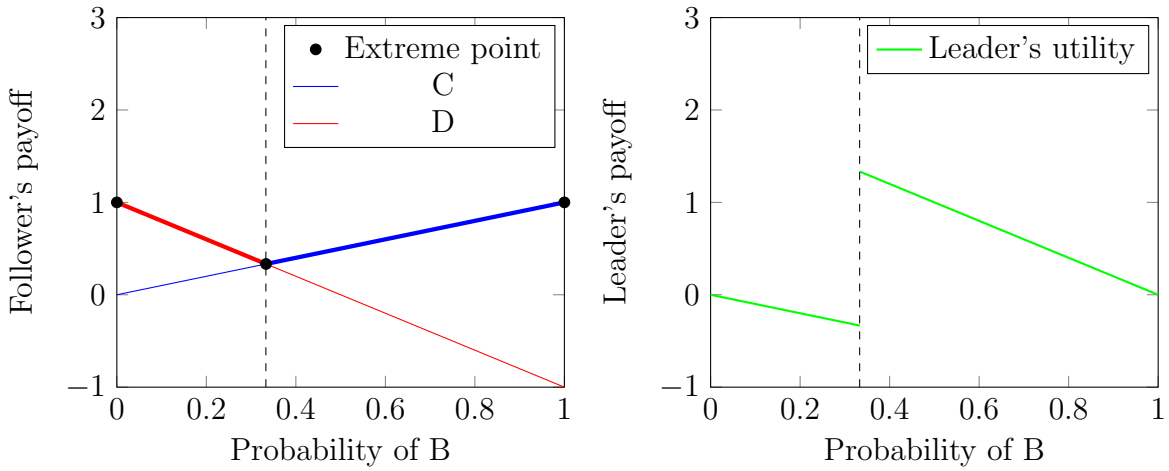

Figure 3.1: Example of creating facets

In the graph, we can see that the segments of the leader's utility function corresponding to the same best response are linear. We will call the points where the best response changes, together with points corresponding to the leader's pure strategy, *extreme points*. The extreme points are marked with black points. Those parts of the follower's utility function corresponding to his best response will be called *facets*. The extreme points mark borders of the facets. In this case, since we are searching for a strong Stackelberg equilibrium, the leader can maximize his expected utility by committing himself to such a strategy that assigns probability $\frac{1}{3}$ to action A and $\frac{2}{3}$ to B, i.e., the strategy corresponding to the middle extreme point. For each facet, we have to maintain the information about both leader's and follower's expected utility in its extreme points, together with the strategies corresponding to those points, in order to be able to reconstruct the strategy throughout the game tree.

This idea can be extended to games with more than two leader's actions by adding more

dimensions. Instead of one single axis, the leader's actions will form a *leader's simplex*. In the sequence-form games, the leaves will be represented with one single extreme point.

However, in the dynamic approach, we can't work directly with the expected utility values in the non-terminal nodes. Instead, we have to consider all the facets from the child nodes (except of those removed by the pruning algorithm, introduced in the following section). For a node $n \in H$, we denote the set of leader's actions $A_l(n)$, and the set of follower's action $A_f(n)$. The set of all facets from child node $q(n, a_l, a_f)$ will be denoted $F(a_l, a_f)$ for $a_l \in A_l(n)$ and $a_f \in A_f(n)$. For a fixed follower's action $a_{f_1} \in A_f$ and all leader's actions $a_l \in A_l$, we introduce a set denoted with $s(a_{f_1})$, containing exactly one facet from each set $F(a_l, a_{f_1})$. One facet from $F(a_l, a_{f_1})$ is denoted with $f(a_l, a_{f_1})$, and $E(f(a_l, a_{f_1}))$ is a set of all extreme points from that facet. A set of all possible combinations of facets will be denoted $S(a_{f_1})$. From each set $s(a_{f_1}) \in S(a_{f_1})$, a new facet will be constructed. We will introduce a linear program to form these new facets. Variables $M(n)$ denotes follower's utility after leader plays a minmax strategy, and $U_f(e)$ denotes follower's utility in the extreme point e. The probability of leader's action a_l being played is denoted with $p(a_l)$. We will also introduce new variables $c(e)$, defined for each extreme point e from each facet $f \in s(a_f)$. They form a probability distribution over all extreme points of a facet, multiplied by the probability that the game node containing that facet will be reached, i.e. they form a probability distribution over all extreme points from each facet $f \in s(a_f)$. The LP for a fixed follower's action a_{f_1} has the following form [10]:

$$0 \leq p(a_l) \leq 1 \qquad \forall a_l \in A_l(n) \tag{3.11}$$

$$\sum_{a_l \in A_l(n)} p(a_l) = 1 \tag{3.12}$$

$$0 \leq c(e) \leq 1 \qquad \forall a_l \in A_l(n); e \in E(f(a_l, a_{f_1})) \tag{3.13}$$

$$p(a_l) = \sum_{e \in E(f(a_l, a_{f_1}))} c(e) \qquad \forall a_l \in A_l \tag{3.14}$$

$$\sum_{a_l \in A_l(n)} \sum_{e \in E(f(a_l, a_{f_1}))} c(e) U_f(e) \geq \sum_{a_l \in A_l(n)} p(a_l) M(q(n, a_l, a_f)) \quad \forall a_f \in A_f(n) \tag{3.15}$$

Note that this LP has no objective function because our aim is to find a polytope defined by the constraints. Constraints (3.11) and (3.12) says that p is a probability

distribution. Constraints (3.13) and (3.14) says that c is a probability distribution over extreme points of a given facet. The meaning of the last constraint (3.15) is similar to the constraint (3.7) from the previous MILP. It ensures that the fixed follower's action a_{f_1} is the best response - follower's utility if the extreme points $e \in E(a_l, a_{f_1})$ must be greater or equal to his minmax utility in the rest of the nodes. This constraint can make the whole LP infeasible. In that case, no new extreme points are generated because a_{f_1} can't be the best response to any leader's mixed strategy profile. In case that it is feasible, it forms a polytope, which we will use as the new facet.

3.2.2 Extreme point pruning

As we mentioned before, we will not need to hold all of the facets. The facets are being used to hold information about leader's and follower's expected utilities. For a node n and a fixed follower's action $a_{f_1} \in A_f(n)$, we denote a set of all extreme points $E(f(a_l, a_{f_1})), \forall a_l \in A_l(n)$ as D.

For this section, we will introduce a new kind of two-dimensional facets f' [10]. For each extreme point $e \in D$, we will only be interested in the leader's and follower's utility in this point, U_l and U_f (we are not interested in leader's strategy). The facet f' is a convex hull of all points $(U_l(e), U_f(e)), \forall e \in D$.

We can prune all the points $(U_l(e), U_f(e))$ from facet f' that can be expressed as a convex combination of some other points from that facet. It means that only the set of generators of the convex hull will remain.

Although we need to keep the information about the whole range of follower's possible outcome, in case of the leader, we only need to keep those points which brings him the best outcome. Let us consider two points $p_1 = (U_{l_1}, U_f), p_2 = (U_{l_2}, U_f)$, where p_1 corresponds to an extreme point, while p_2 may correspond to a convex combination of some extreme points, $U_{l_2} \geq U_{l_1}$ and U_f is fixed. We can prune the extreme point corresponding to p_1 because it wouldn't bring the leader any advantage - he would always prefer p_2.

After this pruning, we will be left with the upper envelope of the initial facet. The pruning is visualized in 3.2.2.

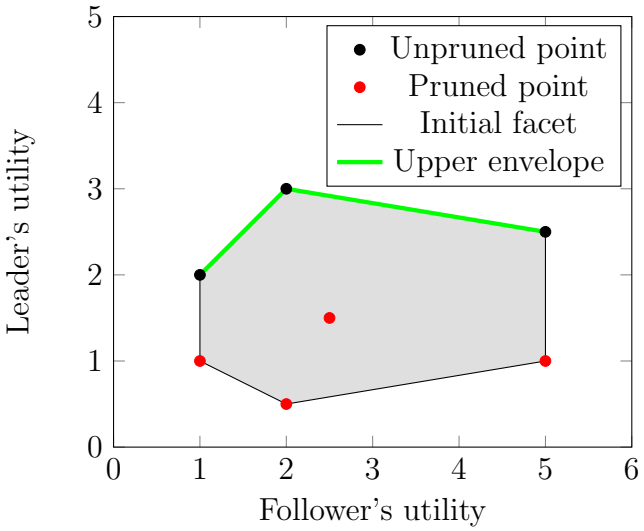

Figure 3.2: Example of pruning

Algorithm 2: Algorithm using dynamic programming

Input: *RootOfGameTree*
Result: *LeadersUtility*

```
processedRoot = DFS(RootOfGameTree)
utility = getLeaderUtility(getBestExtremePoint(processedRoot))
return utility

Function DFS (Node N):
    foreach successor ∈ N.succesors do
        DFS(successor)
    end
    ProcessNode(N)
return N.facets
Function ProcessNode (Node N):
    if N.isTerminal then
        E = createExtremePoint(leaderUtility, followerUtility)
        N.createFacet(E)
    else
        A = N.followerActions
        foreach current_follower_action ∈ A do
            F = []
            S = N.successorsOfAction(current_follower_action)
            foreach successor ∈ S do
                F.addFacetSet(successor.facets)
            end
            C = getCombinationsFromEachSet(F)
            foreach combination ∈ C do
                LP = createLPfromFacets(combination)
                polytope = transformToPolytope(LP)
                vertices = getVertices(polytope)
                E = []
                foreach vertex ∈ vertices do
                    NewExtremePoint = convertVertexToExtremePoint(vertex)
                    E.add(NewExtremePoint)
                end
                points.fillUtilityValues()
                N.createFacet(E)
            end
            prunePoints(N)
            removeEmptyFacets(N)
        end
    end
end
```

The algorithm gets the root of the game as an argument in line 1. It traverses the tree with DFS (line 5), which means that each node is processed (line 9) after all of it's successors are processed (line 6).

If the current node N is terminal (line 12), it will be represented with a single facet (line 14) with one extreme point (line 13). If the node N is not terminal, we will collect all follower's actions in that node into set A (line 16). We will then iterate over all of these actions (line 17). We will introduce a set S (line 19), containing all nodes which can be reached with the current follower's action and some leader's action ($q(N, a_l, current\_follower\_action) \quad \forall a_l \in A_l(N)$). We will also introduce an empty set F (line 19). For each node in S, we will collect all facets from that node into an set, and add that set into F (line 21). Now, the set F contains multiple sets of facets from succeeding nodes.

We have to consider each possible combination of facets from these sets. For example, if the set F contained three sets of facets, (f_1, f_2), (f_3, f_4), (f_5, f_6), the possible combinations would be $[f_1, f_3, f_5]$, $[f_1, f_3, f_6]$, $[f_1, f_4, f_5]$, etc. We will collect all these combinations into set C (line 23).

We will now iterate over these combinations from C. A new facet will be formed for each of these combinations. For each of these combinations, we will construct an LP from 3.2.1 (line 25). The LP can be then transformed into a polytope (line 26). The vertices of the polytope correspond to the extreme points of the new facet - we can convert each vertex to a new extreme point (line 30) because the coordinates of the vertices correspond to the utility values of the extreme points. The facet is then formed from these points (line 34).

After all facets corresponding to some follower's action are formed, the pruning is applied (line 36) and empty facets are removed (37).

3.2.3 Heuristic pruning

The aforementioned pruning technique keeps all the information necessary to determine exactly the leader's best strategy. However, most of the information will most likely be still unused. We can go further with the pruning and keep only a certain number of extreme points with the highest leader's expected utility. Let us consider the example from 3.2.2. If we used the heuristic pruning technique and decided to keep a single point, we would be left with the point with coordinates $(2, 3)$. If we decided to keep two of them,

we would be left with $(2, 3)$ and $(5, 2.5)$.

This approach gives us the lower bound on the leader's expected utility in the whole game [10].

Chapter 4

Computing SE in imperfect-information games

In the Stackelberg games, the leader has to commit to some strategy before the start of the game. The follower observes that commitment and plays his best response.

In perfect-information games, the leader was able to commit to a different strategy in each game node. That allowed him to discourage the follower from going into nodes which did not benefit him - he simply committed to the minmax strategy in those nodes.

In games with imperfect information, the leader's strategy has to be the same for all nodes from a given information set. For that reason, the same strategy, which is supposed to bring him the best possible utility in one node, must also work as the punishment strategy in the other nodes in order to prevent the follower from going into that nodes. We will demonstrate this on an example in section 4.2.1.

In this chapter, we will introduce two algorithms for computing Stackelberg equilibria in games with imperfect information. The first is an existing algorithm, which computes one mixed-integer linear program (MILP) to solve the Stackelberg game. The second one uses dynamic programming. It is based on the same idea as the DP algorithm from the previous chapter. The second algorithm is the main aim of this work.

4.1 MILP algorithm for imperfect-information games

This section will introduce an existing algorithm for finding Stackelberg equilibria in games with imperfect information. We will use it as a baseline algorithm. Unlike the MILP algorithm from the previous section, this one is solved only once.

Sets

N	set of players
Z	set of terminal game nodes
$A_i(I_i)$	set of actions of player i in information set I_i
$\mathcal{I}_i$	set of player's i information sets

Variables

$p(n)$	$n \in H \cup Z$	probability of node n being reached during the game
$u_i(n)$	$i \in N; n \in H \cup Z$	player's i expected utility in node n
$u_i(\sigma_l, \sigma_f)$	$i \in N; \sigma_i \in \Sigma_i$	player's i expected utility in a node reached by given sequences, defined to 0 if the node is not terminal
$r_i(\sigma_i)$	$i \in N; \sigma_i \in \Sigma_i$	probability of player's i sequence
s_{σ_i}	$i \in N; \sigma_i \in \Sigma_i$	slack variable associated with sequence $sigma_i$
$seq_i(I_i)$	$i \in N; I_i \in \mathcal{I}_i$	sequence needed to reach info. set I_i
$inf_i(\sigma_i)$	$i \in N; \sigma_i \in \Sigma_i$	information set in which last action of σ_i was played
v_{I_i}	$i \in N; I_i \in \mathcal{I}_i$	player's i expected utility in info. set I_i

$$\max_{p,r,v,s} \sum_{z \in Z} p(z) u_1(z) \tag{4.1}$$

Subject to:

$$r_i(\emptyset) = 1 \qquad \forall i \in N \tag{4.2}$$

$$r_i(\sigma_i) = \sum_{a \in A_i(I_i)} r_i(\sigma_i a) \qquad \forall i \in N; \forall I_i \in \mathcal{I}_i; \sigma_i = seq_i(I_i) \tag{4.3}$$

$$0 \leq s_{\sigma_2} \leq (1 - r_2(\sigma_2)) \cdot M \qquad \forall \sigma_2 \in \Sigma_2 \tag{4.4}$$

$$0 \leq p(z) \leq r_1(seq_1(z)) \qquad \forall z \in Z \tag{4.5}$$

$$0 \leq p(z) \leq r_2(seq_2(z)) \qquad \forall z \in Z \tag{4.6}$$

$$\sum_{z \in Z} p(z) = 1 \tag{4.7}$$

$$r_2(\sigma_2) \in 0, 1 \qquad \forall \sigma_2 \in \Sigma_2 \tag{4.8}$$

$$0 \leq r_1(\sigma_1) \leq 1 \qquad \forall \sigma_1 \in \Sigma_1 \tag{4.9}$$

$$v_{inf_2(\sigma_2)} = s_{\sigma_2} + \sum_{I' \in \mathcal{I}_2 : seq_2(I') = \sigma_2} V_{I'} + \sum_{\sigma_1 \in \Sigma_1} r_1(\sigma_1) u_2(\sigma_1, \sigma_2) \qquad \forall \sigma_2 \in \Sigma_2 \tag{4.10}$$

The objective function, which we intend to maximize, is the sum of the leader's utility values in leaf nodes, multiplied by the probability of reaching them. The constraint (4.2) states that the empty sequence will be played with probability 1 for each player, and the following constraint (4.3) ensures that the probability of any sequence is equal to the sum of probabilities of all sequences, which extend the original one with one additional action. The constraint (4.4) introduces the slack variables. One slack variable is defined for each follower's possible sequence, and the intuition behind them is that they are equal to 0 in the case that the corresponding sequence is really played in the game. The M in the equation stands for an arbitrary great number. The constraints (4.5) and (4.6) ensures that the probability of reaching some terminal node is not higher than the probability of the sequence leading to that node, both for the leader and the follower, and the following constraint (4.7) ensures that the sum of probabilities of terminal nodes is equal to one, i.e., one of them will be certainly reached. Constraint (4.8) defines the probabilities of the follower's sequences to be binary - the follower can not randomize. The leader is able to randomize, and constraint (4.9) ensures that the probability of any of his possible sequences is greater than 0 and less than 1.

The last constraint, (4.10), ensures that the follower always plays his best response. The left-hand side of the equation is the follower's expected utility in the information set, in which the last action of the current sequence was played. The right-hand side of the constraint can be divided into three parts. The first one is the slack variable, corresponding

to the current follower's sequence. As we mentioned before, if that sequence is played, the variable is equal to 0. The next one is the sum of expected utilities in the follower's information sets, which can be reached with the current sequence (those are the succeeding information sets). The third part is the sum of utilities in all terminal nodes, which can be reached with the current follower's sequence and any leader's sequence. The intuition behind this constraint is that the follower's expected utility of the current information set is constant, and it is equal to the payoff accomplished by playing the best response. For the best response, the slack variable is equal to 0. For the other possible sequences, the slack variable is greater than 0 to fill the gap. The constraint (4.4) ensures that the sequence with a non-zero slack variable can not be played. Thus only the best response will be played.

4.2 Dynamic algorithm for imperfect-information games

In this section, we will introduce an algorithm for finding Stackelberg equilibria in imperfect-information games using dynamic programming. This algorithm is the main aim of this work. We will restrict to two-player games (one leader and one follower) with perfect recall. Furthermore, the follower will always have the perfect information.

This algorithm is based on a similar idea as the aforementioned DP algorithm for perfect-information games. It traverses the game tree with depth-first search, starting with leaves and propagating the necessary information to the predecessor nodes.

4.2.1 Sub-tree representation in games with imperfect information

In the previous part of this work, we used the leader's simplex to represent the leader's possible strategies in a given sub-game. Each vertex of the leader's simplex represented one leader's action. Each point from the convex hull of these vertices represented some leader's mixed strategy in a given game node.

We also used facets to represent follower's actions. There is always at least one follower's best response to each leader's strategy. Each of the follower's facets represented one action in the current sub-game. In the case with perfect information, we were able to determine whether the action is the best response to some leader's strategy. It was also possible to keep only those sections of facets, which represented the best response, and

omit the rest without losing any information important for finding Stackelberg equilibrium.

In games with imperfect information, we can not make this assumption because the leader has to commit to one strategy for the whole information set, not only for one node. We will illustrate this in the following example.

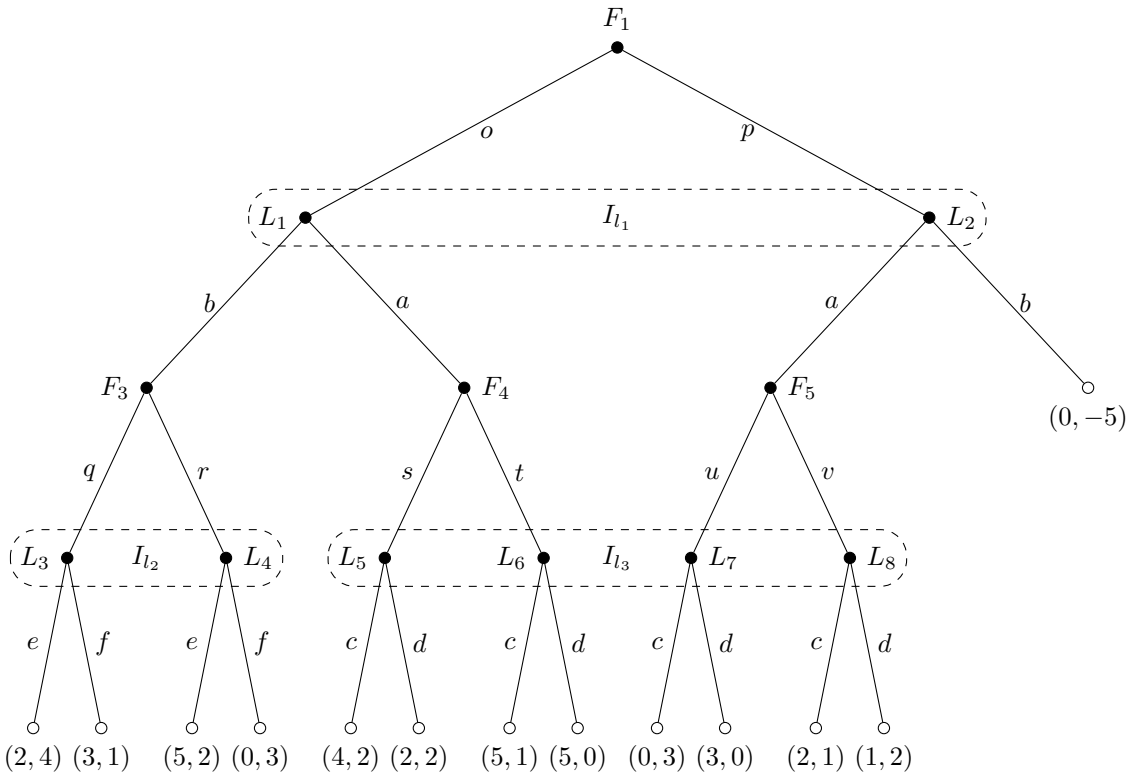

Figure 4.1: Stackelberg game

The sequence-form game from Figure 4.1 represents a simple Stackelberg game with imperfect information. Leader's game nodes are divided into three information sets - I_{l_1}, I_{l_2} and I_{l_3}. Follower always has imperfect information.

First, suppose that leader commits to the action a in the root node, so we will consider only the information set I_{l_3}. First, we will consider the node F_4 and the corresponding sub-tree. In this sub-tree, the follower's action s is the best response to any leader's strategy. For that reason, from the perspective of this sub-tree, the leader's best option is to commit to the pure strategy c.

However, the nodes L_7 and L_8 belong to the same information set as the nodes L_5 and L_6, so the leader has to commit to the same strategy in all of these nodes. Since the follower is able to observe the commitment, he can decide in which sub-tree (F_4 or F_5) he can obtain a better expected payoff and go to that sub-tree by playing o or p in F_1.

If the leader's strategy was pure c, the follower's best response would be the sequence (p, u), which would bring the leader payoff equal to 0. For that reason, the leader has to commit to such a mixed strategy which makes the follower prefer the sub-tree F_4. In this particular case, it would be a strategy in which c is played with probability 2/3 and d with probability 1/3 in the information set I_{l_3}, and follower's best response to this strategy would be the sequence (o, q). The facets from this information set are depicted in Figure 4.2

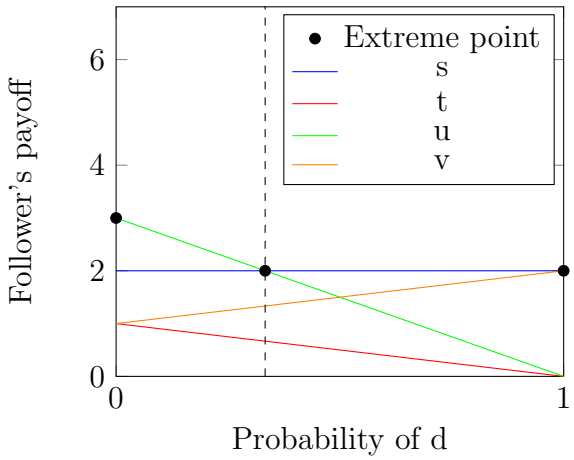

Figure 4.2: Facets in I_{l_3}

Now, we will consider the whole game tree. In the left sub-tree, the leader aimed to discourage the follower from playing action p by committing to an appropriate strategy in the information set I_{l_3}. However, if we consider the whole tree, the leader is also able to influence follower's best response from predecessor nodes. If the aim is to discourage the follower from playing the action p, the leader can assign a non-zero probability to action b in I_{l_1}. Because follower's payoff reached by playing action p in the root is very small, the leader is now able to assign a higher probability to action c in the information set I_{l_3} without changing follower's best response (follower does not want to risk getting to the right-most leaf node). For that reason, we have to keep the whole facet for action s, not only the section which corresponds to the best response from the perspective of the sub-tree.

We have to modify the concept of facets and leader's simplex for the imperfect-information games, which will now be a treeplex - a special kind of polytope [12]. In perfect-information games, each vertex represented one leader's action in a given node. In games with imperfect information, each vertex will represent one leader's possible commitment across all his information sets in a given sub-tree. For example, in the game from Figure 4.1, the polytope in the root node will have 8 vertices because there are three leader's information sets, each with two available actions. There will be one vertex for

commitment a in I_{l_1}, e in I_{l_2} and c in I_{l_3}, another vertex for a in I_{l_1}, e in I_{l_2} and d in I_{l_3}, etc. The polytope has a form of a cube, where one axis represents the commitment in node I_{l_1}, the second axis represents the commitment in I_{l_2}, and the third axis represents the commitment in I_{l_3}. It is depicted in Figure 4.3.

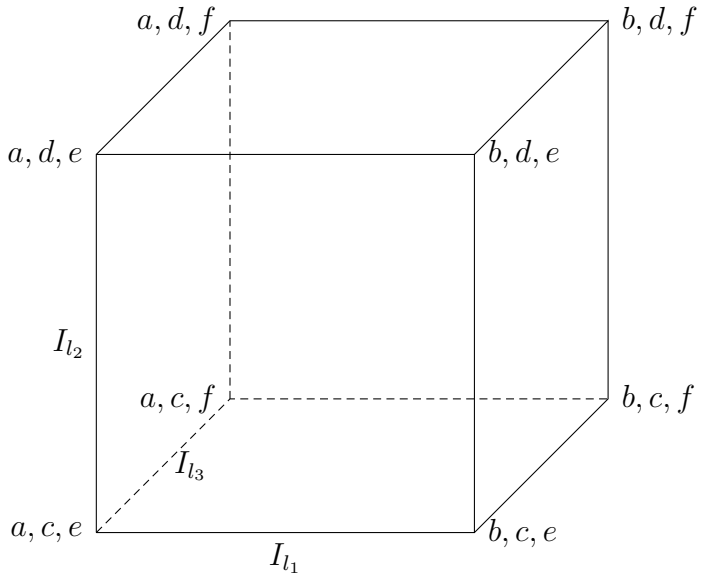

Figure 4.3: Leader's polytope in the root

Facets will now represent possible follower's sequences in a given sub-tree (in perfect-information games, each facet represented only one follower's action from a given node). In the root node, there will be facets for sequences (o, q, u), (p, s), etc. Note that we have to include both q and u into one facet, even though they will never occur at the same time because the leader can commit to such a mixed strategy, which makes both of these actions reachable. On the other hand, we do not have to include, for example, o and u into one facet because once o is played, u is not reachable in the game.

Utility values for these facets are defined on the whole leader's polytope. The whole game can now be represented with a single leader's polytope, containing all leader's possibilities and a facet for each follower's possible response. That is all we need to find the equilibrium in the game. However, the number of facets and extreme points grows exponentially with the number of game nodes. For that reason, we will also introduce a pruning technique, which will enable us to omit some facets.

4.2.2 Finding Stackelberg equilibria using the DP algorithm

The algorithm traverses the game tree with a depth-first search (DFS). The DFS will enable us to construct the final leader's polytope from the bottom of the tree up to the

root. It will also enable us to apply the pruning technique in each sub-tree.

We will construct the polytope in each node we come across and then merge them in the predecessor nodes. Let us consider two sub-trees, each containing one leader's information set, I_1 and I_2. There are two possible leader's action in each of these sets, a and b in I_1, and c and d in I_2. The leader's polytopes in these sub-trees will have a form of a line segment, where each vertex represents one leader's action. By merging these polytopes, we form a new square-shaped polytope. One vertex represnts the commitment to action a in I_1 and b in I_2, another represents a in I_1 and c in I_2, etc.

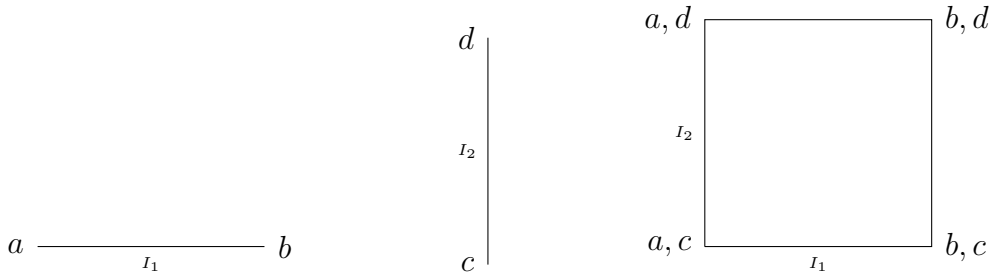

Figure 4.4: Example of merging leader's polytopes

Once we have constructed the polytope for the root, we use an LP to find the Stackelberg equilibrium. The LP is computed for each follower's possible sequence (facet) in the node, and the main idea is that it decides whether the current sequence is the best response to some leader's strategy (some segment of leader's polytope). If so, we can then find a point on the facet, which provides the best payoff to the leader, and save the result as a potential leader's best utility. After computing the LP for each facet, we can choose the highest value from the stored results.

Unlike the perfect-information case, all facets from one node contain the same extreme points. For that reason, we can denote the set of extreme points in node n as $E(n)$, and player's i utility on facet f in extreme point e as $U_i(e, f)$. The LP, computed for facet f_1 from node n, has the following form:

Variables

$c(e)$	probability of the extreme point e from leader's polytope
$U_i(e, f)$	player's i payoff in extreme point e and facet f
$E(n)$	set of extreme points in leader's polytope from node n

$$\max \sum_{e \in E(n)} c(e) U_l(e, f_1) \tag{4.11}$$

$$0 \leq c(e) \leq 1 \qquad \forall e \in E(n) \tag{4.12}$$

$$\sum_{e \in E(n)} c(e) = 1 \tag{4.13}$$

$$\sum_{e \in E(n)} U_f(f_1, e)c(e) \geq \sum_{e \in E(n)} U_f(f, e)c(e) \qquad \forall f \in F(n) \tag{4.14}$$

This algorithm basically transforms the extensive-form game into a normal form. However, the pruning significantly decreases the size of the resulting game.

4.2.3 Facet pruning technique

As we mentioned before, it is possible to omit some facets, which will certainly not be played, because follower always has a better option. For example, let us consider the game from Figure 4.1. In the sub-tree corresponding to the node F_4, follower has two actions available - s and t. However, the action t never provides him better payoff then action s (t never is the best response). Since follower has the perfect information, once he enters this sub-tree, he has no reason to play t. Because of that, we can omit the corresponding facet.

Generally, the facets in imperfect-information games do not represent single actions but rather the whole sequences. Once we omit the facet corresponding to t, we do not have to consider any sequence which contains that action. This significantly reduces the number of facets in the whole game.

As we mentioned before, with the DP algorithm, we traverse the game tree with DFS. In each node, we merge the leader's polytopes and create the new facets. After that, we can run the LP from the previous chapter in that node (once for every facet). For now, we are not interested in the objective value. Our aim is to find out whether the given facet is the best response to some leader's strategy. If so, the LP is feasible, and we have to keep the whole facet. Otherwise, we can omit the whole facet because the follower has no reason to play the corresponding sequence.

Heuristic pruning

If we are interested in the exact result, we have to keep all facets that can be the best response to some segment of the leader's polytope. However, we can also compute the

lower estimation on the leader's expected utility by applying a heuristic pruning.

The heuristic pruning only keeps h facets with the highest leader's expected utility value, and the rest is discarded. Leader's utility is computed with the LP from 4.2.2 for each sub-tree we come across with the depth-first search. This significantly reduces the

number of facets in the game, which decreases the duration of the algorithm.

Algorithm 3: DP algorithm for imperfect-information games

```
Input: Root
utility = DFS(Root)
return utility
Function DFS (Node N):
    foreach successor ∈ N.successors do
        DFS(successor)
    end
    ProcessNode(N)
    retval = Pruning(N)
return retval
Function ProcessNode (Node N):
    if N.isTerminal then
        E = createExtremePoint(N.leaderUtility, N.followerUtility)
        N.addFacet(generateFacet(E))
    else
        polytope = mergePolytopes(N.successors)
        foreach current_follower_action ∈ N.followerActions do
            F = []
            foreach successor ∈ N.successorsOfAction(current_follower_action) do
                F.addFacetSet(successor.facets)
            end
            foreach combination ∈ getCombinationsFromEachSet(F) do
                new_facet = generateFacet()
                follower_sequence = []
                follower_sequence.add(current_follower_action)
                foreach old_facet ∈ combination do
                    follower_sequence.add(old_facet.followerActions)
                    foreach old_point ∈ old_facet.extremePoints do
                        foreach v ∈ polytope.vertices do
                            if old_point.leaderCommitment ⊂ v.leaderCommitment then
                                new_point = createExtremePoint()
                                new_point.utilityValues = old_point.utilityValues
                                new_point.leaderCommitment = v.leaderCommitment
                                new_facet.addPoint(new_point)
                            end
                        end
                    end
                end
                new_facet.followerSequence = follower_sequence
                N.addFacet(new_facet)
            end
        end
    end
end
```

Algorithm 4: Pruning

```
1  Function Pruning (Node N):
2      best_utility = −inf
3      foreach facet ∈ N.facets do
4          LP = CreateLPforFacet(facet, N.facets)
5          if LP.infeasible then
6              N.facets.remove(facet)
7          else
8              if LP.objectiveValue ≥ best_utility then
9                  best_utility = LP.objectiveValue
10             end
11         end
12     end
13 return best_utility
```

4.2.4 Example of the DP algorithm

In this section, we will demonstrate the DP algorithm on the game from Figure 4.1.

As we mentioned before, the algorithm traverses the game tree with DFS (line 3). Processing the leaf nodes is trivial (lines 11), so we start with the node F_4. There is only one leader's information set with actions c and d in this sub-tree, which means that the leader's polytope has a form of a line segment because it is formed by merging two points (line 15). One vertex represents action c, and the other represents d. We will denote these extreme points e_c and e_d. Follower also has two possible sequences (in this case only single actions), s and t, so there will be two facets, each containing both e_c and e_d (line 21). We will denote these facets with f_s and f_t. The utilities of these points are captured in table 4.1.

Facets / E. points	f_s	f_t
e_c	(4, 2)	(5, 1)
e_d	(2, 2)	(5, 0)

Table 4.1: Sub-tree in F_4

After applying the pruning technique (line 8), which utilizes the LP 4.2.2, we can omit the facet f_t, because it never is the best response (line 5).

The LP for facet f_t follows:

$$\max \quad c(e_c) \cdot 5 + c(e_d) \cdot 5 \tag{4.15}$$

$$0 \leq c(e_c) \leq 1 \tag{4.16}$$
$$0 \leq c(e_d) \leq 1 \tag{4.17}$$
$$c(e_c) + c(e_d) = 1 \tag{4.18}$$
$$c(e_c) \cdot 1 + c(e_d) \cdot 0 \geq c(e_c) \cdot 2 + c(e_d) \cdot 2 \tag{4.19}$$

The last constraint is apparently infeasible for non-negative probabilities c.

We will apply the same approach in the nodes F_5 (table 4.2) and F_3 (table 4.3). No facet in these nodes can be pruned, all of them could be a best response (line 7).

Facets / E. points	f_u	f_v
e_c	(0, 3)	(2, 1)
e_d	(3, 0)	(1, 2)

Table 4.2: Sub-tree in F_5

Facets / E. points	f_q	f_r
e_e	(2, 4)	(5, 2)
e_f	(3, 1)	(0, 3)

Table 4.3: Sub-tree in F_3

Now, we can move to the root node. From the perspective of the root, there are three different information sets, each with two actions - one corresponding to the information set I_{l_1}, second one corresponds to I_{l_2}, and the third one to I_{l_3}. Note that leader's polytopes from nodes F_4 and F_5 are the same, so merging them does not increase the dimension of the resulting polytope. The final leader's polytope is depicted in Figure 4.3.

We also have to find all possible follower's sequences (line 23). The two follower's actions in the root are o and p. We will start (line 16) with o: by playing o in the root, follower can get into sub-trees in F_3 (with facets f_q and f_r) and F_4 (with facet f_s, we

have pruned f_t). The follower does not know which of these nodes he will get into, so he has to consider all the possible combinations and form a facet for each of them (line 23). There will be facet for combinations (o, q, s), (o, q, t) (line 25). Now, we will use the same approach to find the facets for those combinations, which contain action p. We will get combinations (p, u) and (p, v).

A new facet will be created for each of these combinations (line 22), and each of these facets will contain all the extreme points from the leader's polytope. The values of the extreme points in different facets are captured in table 4.4.

Facets / E. points	$f_{(o,q,s)}$	$f_{(o,q,t)}$	$f_{(p,u)}$	$f_{(p,v)}$
$e_{a,c,e}$	(4, 2)	(5, 1)	(0, 3)	(2, 1)
$e_{a,c,f}$	(4, 2)	(4, 2)	(0, 3)	(2, 1)
$e_{a,d,e}$	(2, 2)	(5, 0)	(3, 0)	(1, 2)
$e_{a,d,f}$	(2, 2)	(5, 0)	(3, 0)	(1, 2)
$e_{b,c,e}$	(2, 4)	(5, 2)	(0, -5)	(0, -5)
$e_{b,c,f}$	(3, 1)	(0, 3)	(0, -5)	(0, -5)
$e_{b,d,e}$	(2, 4)	(5, 2)	(0, -5)	(0, -5)
$e_{b,d,f}$	(3, 1)	(0, 3)	(0, -5)	(0, -5)

Table 4.4: Facets in the root

Now, we have to apply the LP to each facet (line 3). The LP (line 4) compares the given facet with each other facet to find out whether the given facet is the best response to at least some segment of the leader's polytope.

In this game, leader can obtain the maximum payoff by committing to a mixed strategy, which assigns probability 0.88 to the extreme point $e_{a,c,f}$, and 0.06 to both $e_{b,d,e}$ and $e_{b,d,f}$. Follower's best response to this strategy is to play o in the root, r in F_3 and s in F_4. This provides a payoff of 3.82 to the leader and 2.06 to the follower.

Chapter 5

Experiments

This chapter will benchmark the two algorithms for finding Stackelberg equilibria in games with imperfect information.

5.1 Experiment settings

We will be using randomly generated trees. Follower's utility U_f in each leaf is randomly generated integer in a range from -50 to 50. Leader's utility U_l is then computed as $U_l = (-2 \cdot U_f) + r$ where r is a random number in range from -10 to 10. We are using this method to avoid leaf nodes, which would benefit both the leader and the follower.

The tree will be generated regarding several parameters - depth, number of each player's actions in each node, and the rate of uncertainty. The rate of uncertainty refers to the probability that two nodes with the same leader's sequence belong to the same information set. If it is equal to 0, the leader has perfect information. The game tree parameters will be specified with a triplet x, y, z, where x is the depth, y is the number of actions, and z is the uncertainty.

5.2 Results

Table 5.1 captures the average duration of the algorithms on ten runs for each class of the game tree. The results are presented in a format X, Y, where X stands for the average duration in seconds, and Y stands for the standard deviation. The third, fourth and

fifth column captures the duration of the DP algorithm with heuristic pruning, where h facets are kept in each node. The duration of the DP algorithm grows significantly faster with the size of the game, and the DP algorithm is outperformed by the MILP. However, because the MILP algorithm works with the whole game tree at the same time, it would not cope with large game trees - in this sense, the DP algorithm is more scalable than the MILP algorithm. The heuristic pruning significantly decreases the duration for larger games, but it does not influence the duration for smaller games. For example, the duration of games with depth three is not influenced by the heuristic pruning with $h = 3$ and $h = 5$. The reason is that the number of facets in non-root nodes is never higher than 3.

Algo. / Param.	MILP	DP	DP, $h = 1$	DP, $h = 3$	DP, $h = 5$
$3, 2, 1$	0.008, 0.001	0.054, 0.085	0.032, 0.022	0.040, 0.044	0.03, 0.007
$3, 2, 0.5$	0.008, 0.001	0.047, 0.045	0.027, 0.009	0.042, 0.046	0.033, 0.007
$3, 3, 1$	0.075, 0.009	1.41, 0.873	0.521, 0.284	1.89, 0.913	1.51, 0.361
$3, 3, 0.5$	0.071, 0.009	52.2, 122	4.55, 4.72	66.1, 97.3	54.1, 73.8
$4, 2, 1$	0.073, 0.008	2.98, 2.02	0.334, 0.15	1.91, 1.60	2.34, 1.23

Table 5.1: Average duration on 10 runs in seconds and the standard deviation

However, the heuristic pruning influences the results of the algorithm. Table 5.2 captures the precision after keeping 1, 3 and 5 facets in every node on 4 classes of games in 50 runs. The precision is presented in format X, Y. X stands for the percentage of test cases, for which the heuristic pruning did not influence the result. Y stands for the average difference of the leader's expected utility computed after pruning from the correct result (leader's utility is in a range from -110 to 110). The number of facets we keep clearly increases the precision in Y. For smaller games, the results do not differ much, but the difference grows with the size of the game.

For the game with parameters 3, 3, 0.5, we will inspect what influences the duration of the computation. Figure 5.1 depicts the dependence of the duration of the DP algorithm on the duration of the MILP algorithm for the same game. For 40 test cases, there is no apparent correlation.

Figure 5.2 shows the dependence on the number of extreme points in the leader's polytope in the root node. The dependence here is clear. The duration strongly depends

Parameters \ Algorithm	$h = 1$	$h = 3$	$h = 5$
$3, 2, 1$	0.44, 12.1	0.52, 8.8	0.44, 6.9
$3, 2, 0.5$	0.4, 14.2	0.3, 11.3	0.32, 9.2
$3, 3, 1$	0.06, 27.2	0.06, 16.5	0.06, 15.3
$3, 3, 0.5$	0.05, 35.9	0, 20	0.02, 15.5

Table 5.2: Accuracy of the heuristic pruning

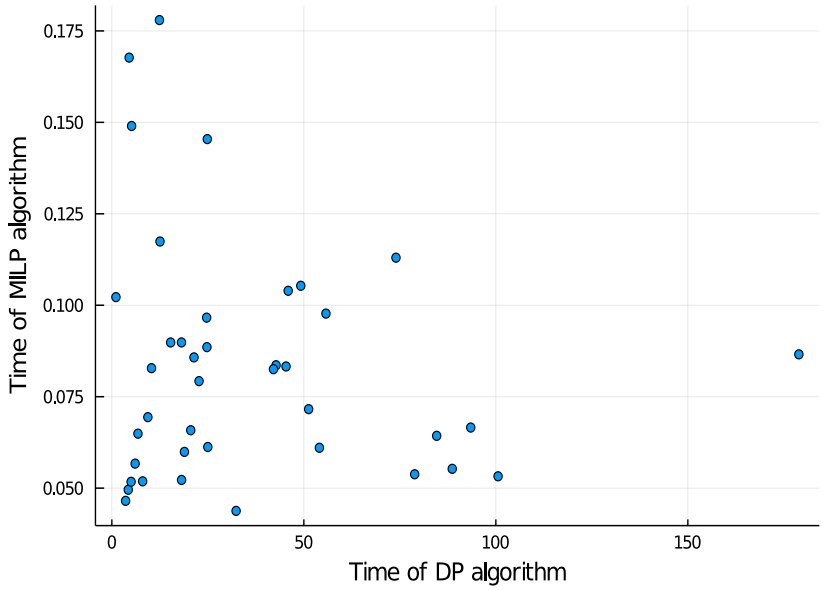

Figure 5.1: Dependence of DP time on MILP time

on the number of extreme points. The number of extreme points grows with the number of information sets in the game. That is why games with higher uncertainty are calculated faster than those with lower uncertainty.

Figure 5.3 shows the dependence on the number of facets in the root node. There is also a correlation. We can see that most of the points are located along three lines. These lines correspond to the dimensions of the leader's polytope. In Figure 5.2, we could see that the most common dimensions are 6, 7, and 8. The left-most line corresponds to dimension 6, etc.

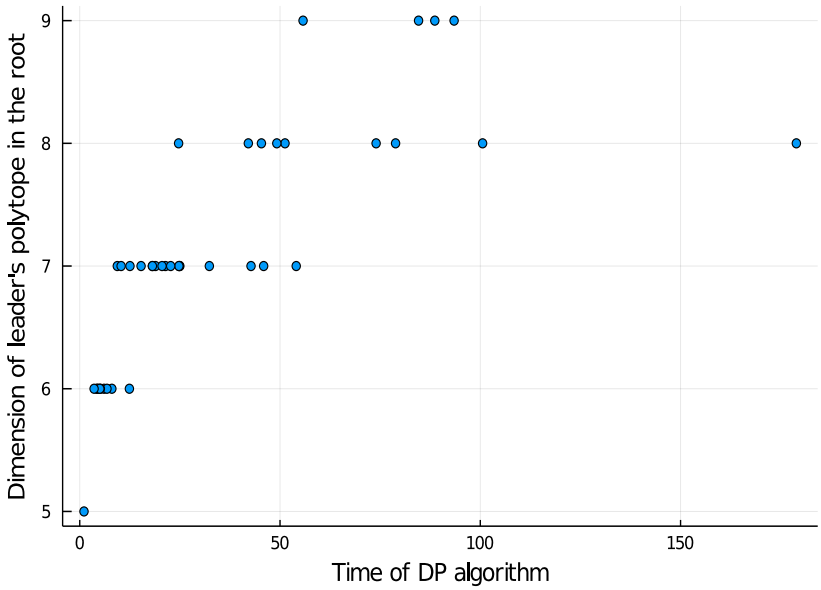

Figure 5.2: Dependence of DP time on the dimension of leader's polytope

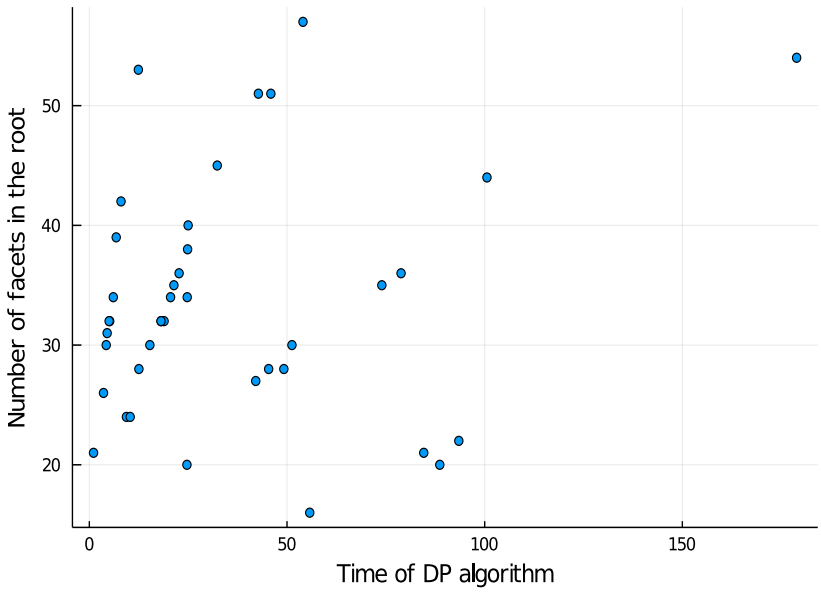

Figure 5.3: Dependence of DP time on number of facets

Chapter 6

Conclusion

Stackelberg equilibria is an important solution concept in the game theory. In Stackelberg games, there are two types of players - leader and follower. The leader's role is to commit to some strategy. The follower then observes the strategy. In Stackelberg equilibria, the leader commits to such a strategy, which provides him the highest possible expected utility, assuming that the follower will play his best response to that strategy.

The goal of this work was to devise and implement an algorithm for finding Stackelberg equilibria in extensive-form games with partially imperfect information. This algorithm extends an existing algorithm for finding Stackelberg equilibria in games with concurrent moves.

The algorithm is based on the idea of dynamic programming (DP), which means that it is not solving the whole game tree all at once. Instead, it is solving individual sub-trees separately, keeping only the necessary information. This opens a possibility for designing a scalable algorithm for games with a long (or even infinite) horizon.

We also have implemented an existing algorithm for finding Stackelberg equilibria, based on solving a single mixed-integer linear program for the whole game. We were using it as a baseline algorithm.

We benchmarked these two algorithms on randomly generated games. Both of the algorithms provided correct results. The baseline algorithm outperformed the new DP-based algorithm in terms of the duration of the computations. However, the advantage of the DP algorithm is in the potential scalability - the baseline algorithm does not apply to large games because of the necessity to work with the whole game tree at once, which results in huge demands on memory. Since it is the first DP algorithm for finding SE

in extensive-form games with imperfect information, the scalability is not optimal yet. Further optimization may be a subject of future work.

We also have introduced a heuristic pruning technique for the DP algorithm. It provides a lower estimation of the leader's expected utility and decreases the duration of the computation.

Bibliography

[1] T.-M. Choi, A. A. Taleizadeh, and X. Yue, "Game theory applications in production research in the sharing and circular economy era", *International Journal of Production Research*, vol. 58, no. 1, pp. 118–127, 2020. DOI: 10.1080/00207543.2019.1681137. eprint: https://doi.org/10.1080/00207543.2019.1681137. [Online]. Available: https://doi.org/10.1080/00207543.2019.1681137.

[2] J. M. Smith, "The Logic of Animal Conflict",, vol. 246, no. 5427, pp. 15–18, Nov. 1973. DOI: 10.1038/246015a0.

[3] D. Korzhyk, Z. Yin, C. Kiekintveld, V. Conitzer, and M. Tambe, "Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness", *Journal of Artificial Intelligence Research*, vol. 41, 297–327, 2011, ISSN: 1076-9757. DOI: 10.1613/jair.3269. [Online]. Available: http://dx.doi.org/10.1613/jair.3269.

[4] S. Kolev, "Heinrich von stackelberg, market structure and equilibrium, translated by damien bazin, lynn urch, and rowland hill (berlin and heidelberg: Springer-verlag, 2011), pp. xiv, 134, 139($hardcover and softcover$).$isbn 978-3-642-12585-0(hardcover); 978-3-642-42390-1(softcover)$.", *Journal of the History of Economic Thought*, vol. 38, no. 4, 557–560, 2016. DOI: 10.1017/S1053837216000894.

[5] K. Horak, B. Bosansky, P. Tomášek, C. Kiekintveld, and C. Kamhoua, "Optimizing honeypot strategies against dynamic lateral movement using partially observable stochastic games", *Computers Security*, vol. 87, p. 101 579, Jul. 2019. DOI: 10.1016/j.cose.2019.101579.

[6] J. Y. Halpern, "Computer science and game theory: A brief survey", *CoRR*, 2007. arXiv: cs/0703148. [Online]. Available: http://arxiv.org/abs/cs/0703148.

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

[8] B. Bosansky, S. Branzei, K. A. Hansen, P. B. Miltersen, and T. B. Sorensen, *Computation of stackelberg equilibria of finite sequential games*, 2016. arXiv: 1507.07677 [cs.GT].

[9] B. Bosansky and J. Cermak, "Sequence-form algorithm for computing stackelberg equilibria in extensive-form games", *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Jan. 2015.

[10] E. Rindt, *Dynamic programming for computing stackelberg equilibrium in sequential games*, 2019.

[11] J. Cermak, B. Bosansky, K. Durkota, V. Lisy, and C. Kiekintveld, "Using correlated strategies for computing stackelberg equilibria in extensive-form games", *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[12] C. Kroer, K. Waugh, F. Kilinc-Karzan, and T. Sandholm, *Theoretical and practical advances on smoothing for extensive-form games*, 2017. arXiv: 1702.04849 [cs.GT].

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kraus David** Personal ID number: **483272**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Dynamic Programming for Computing a Stackelberg Equilibrium in Finite Sequential Games with Partial Imperfect Information

Bachelor's thesis title in Czech:

Dynamické programování pro výpočet Stackelbergových strategií v částečně pozorovatelných sekvenčních hrách

Guidelines:

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 typically consider the whole game since SE prescribes to find a global optimum. Only recently, an algorithm based on dynamic programming has been introduced, however, the algorithm works only for finite sequential games with perfect information and concurrent moves. The goal of the student is to adapt this dynamic-programming algorithm to games with partial imperfect information:
1. Design an adaptation of the existing dynamic-programming algorithm for concurrent-moves games to games with partial imperfect information.
2. Implement the new version of the algorithm.
3. Experimentally compare with a baseline MILP algorithm for computing SE for games with imperfect information.

Bibliography / sources:

[1] 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, 2017. ACM TEAC
[3] E. Rindt; Dynamic Programming for Computing Stackelberg equilibrium in Sequential Games. Master Thesis. 2019

Name and workplace of bachelor's thesis supervisor:

doc. Mgr. Branislav Bošanský, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **09.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

doc. Mgr. Branislav Bošanský, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. _____
Date of assignment receipt

_____
Student's signature