Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

The Computation of intermediate Set

Marek Žuffa

Supervisor: doc. Ing. Tomáš Kroupa, Ph.D.
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Žuffa Marek** Personal ID number: **483799**

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:

The Computation of Intermediate Set

Bachelor's thesis title in Czech:

Výpočet mezilehlé množiny

Guidelines:

The student will learn about the intermediate set, which is a solution concept for cooperative games. Since the intermediate set is typically a non-convex polyhedron, its computation and description can be very demanding problems. The goal of the thesis is to implement the basic algorithm for the intermediate set. It is based on the enumeration of all ordered partitions of the player set and the computation of the underlying convex cells. This algorithm will be implemented for a small number of players and its output will be the list of convex cells (described by their vertices) and, if possible, the visualization of the intermediate set. The implementation will be done in Matlab (using Multi-Parametric Toolbox 3.0) and in Python. The main features of these implementations will be compared (expresssive power, scalability etc.)

Bibliography / sources:

[1] L. Adam and T. Kroupa. The intermediate set and limiting superdifferential for coalitional games: between the core and the Weber set. International Journal of Game Theory, 46(4):891-918, 2017.
[2] G. Chalkiadakis, E. Elkind, and M. Wooldridge. Computational Aspects of Cooperative Game Theory. Morgan & Claypool Publishers, 2011.
[3] M. Herceg, M. Kvasnica, C.N. Jones, and M. Morari. Multi-Parametric Toolbox 3.0. In Proc. of the European Control Conference, pages 502–510, Zurich, Switzerland, July 17–19 2013. http://control.ee.ethz.ch/~mpt .

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Kroupa, Ph.D., Artificial Intelligence Center, FEE

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

Date of bachelor's thesis assignment: **04.02.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

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

Acknowledgements

I would like to thank my family and friends for all the support during the studies. I would like to also thank doc. Ing. Tomáš Kroupa, Ph.D. for valuable information and help during the writing of this thesis.

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, date 21st May 2021

Marek Žuffa

Abstract

This bachelor thesis is devoted to a brief introduction to a coalitional game theory and to present the intermediate set as a solution lying between the core and the Weber set. Firstly we introduce the game theory as a mathematical science, the aim of this thesis, and the motivation for writing it. Next, theoretical basics are explained, as well as some much-needed terms, common solutions, and the intermediate set is introduced. After the theoretical part, we demonstrate the implementation in MATLAB and Python, respectively. Examples are given to clarify what is the intermediate set in real experiments. In conclusion, we compare both implementations and summarize the whole thesis.

Keywords: Game theory, Coalitional game, Core, Weber set, Intermediate set

Supervisor: doc. Ing. Tomáš Kroupa, Ph.D.
Artificial Intelligence Center,
Faculty of Electrical Engineering,
Czech Technical University in Prague,
Karlovo nám. 13,
12135 Praha 2,
Czech republic

Abstrakt

Táto bakalárska práca sa venuje uvedeniu do problematiky koaličných hier a ukazuje medziľahlú množinu ako možné riešenie, ležiace medzi jadrom a Webrovou množinou. Úvod predstavuje teóriu hier ako matematickú vedu, popisuje sa tu cieľ a motivácia k bakalárskej práci. Ďalej je bližšie popísaná potrebná teória a pojmy ku pochopeniu danej problematiky, ako aj vysvetlenie riešení a uvedenie medziľahlej množiny. Po teoretickej časti nasleduje implementácia v MATLABe a Pythone. Po implementačnej časti nasledje časť venovaná príkladom pre lešie predstavenie danej témy. Záver porovnáva implementácie a zhrňuje celú prácu.

Kľúčové slová: Teória hier, Koaličná hra, Jadro, Webrova množina, Medziľahlá množina

Preklad názvu: Výpočet medziľahlej množiny

Contents

Figures

Tables

Chapter 1
Introduction

Game theory is a branch of applied mathematics. It deals with mathematical models of strategic interaction and reward distribution among players, also called agents. Game theory has seen great progress in the last 70 years. The first games defined were two-player games such as prisoner's dilemma. This game is about two prisoners cooperating with the police and testifying against each other. It studies whether it is profitable if they are both silent or not. They are divided and they are not informed about the decision of the other one. It is still used today as a great example and introduction to game theory however, we are not be discussing game theory in general.

This thesis is devoted to an introduction to coalitional game theory and computation of intermediate set. We are studying coalitional games with transferable utilities only. Common solutions, the core, and the Weber set, are shown as they are needed to understand the aim of this thesis. Also, the computation of vertices is briefly explained. Examples are provided to show the differences between the solutions and to give a better general understanding of the topic.

1.1 Aim of the thesis

The main aim of the theoretical part of this thesis is to show and explain a solution to coalitional games introduced by Lukáš Adam and Tomáš Kroupa in International Journal of Game Theory in 2017, the intermediate set [1]. Definition of a convex polytope, some basic terms and common solutions, the core, and the Weber set is provided. The Vertex enumeration problem is discussed. The practical part is devoted to the implementation of a solver to the intermediate set in two programming languages, MATLAB and Python, along with a comparison of those implementations. The experiments, or examples in this thesis, are used to show practicality, functionality, and differences between these programs.

1.2 Motivation

The science of game theory always intrigued me and I have always found it interesting and practical with a wide range of usage. However, I have never studied coalitional game theory in particular so when the occasion showed, I thought it would be great to study a new subject and learn something I can profit from and put to use in the future. It has great potential in economics and many branches of informatics.

Chapter 2

Coalitional game

A coalitional or a strategic game is cooperative if the players can make binding agreements about the distribution of payoffs or the choice of strategies, even if these agreements are not specified or implied by the rules of the game [2]. Coalitional game theory examines and predicts creating coalitions between players in addition to fair, yet still efficient and rational distribution of rewards. However, it does not address finding and making an optimal strategy for the game itself, as it is one of the prerequisites that such strategy is already known. Common solution concepts are the core and the Weber set. This paper addresses these two slightly as we need them to understand the main task given which is an introduction of an intermediate set. The intermediate set is an interpolating solution concept between the core and the Weber set. Both are represented by convex polyhedrons in $n-1$ dimensional space, where n is number of players. The intermediate set is a union of many convex polyhedrons. This chapter is heavily inspired and got most of the information about the intermediate set, the core, and the Weber set from the paper on the intermediate set [1], and materials to multiagent systems from CTU [3] [4].

2.1 Convex polytope

As defined in [5], a convex polytope is defined as a convex hull of finitely many points in a Euclidean space. The convex hull of a set of points X in Euclidean space is the smallest convex set containing X. The convex set Y is such set that all points on the line segment joining any two points $a, b \in Y$ are also in Y. This definition also defines V-representation or vertex representation.

Another definition is using intersection of half spaces, also known as H-representation or half-space representation. Convex polytope is an intersection of a finite number of half-spaces. A closed half-space can be defined using linear inequality:

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq b \tag{2.1}$$

Furthermore, a convex polyhedron can be written as set of solutions to the

system of linear inequalities:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & \leq & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & \leq & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & \leq & b_m \end{array} \tag{2.2}$$

where m is the number of half spaces. Matrix inequality is often used:

$$\mathbf{A}x \leq b \tag{2.3}$$

where $\mathbf{A}$ is a $m \times n$ matrix, x is a $n \times 1$ column vector of variables and b is a $m \times 1$ column vector of scalars [6].

For our needs, we add a subspace defined by a set of linear equations:

$$\begin{array}{ccccccccc} c_{11}x_1 & + & c_{12}x_2 & + & \dots & + & c_{1n}x_n & = & d_1 \\ c_{21}x_1 & + & c_{22}x_2 & + & \dots & + & c_{2n}x_n & = & d_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ c_{m1}x_1 & + & c_{m2}x_2 & + & \dots & + & c_{mn}x_n & = & d_m \end{array} \tag{2.4}$$

Those can also be written in a matrix form:

$$\mathbf{C}x = d \tag{2.5}$$

These further restrict the convex polyhedron and bound it to subspace determined by these equations. This is used in the computation of core and the main task at hand, which is the computation of intermediate set.

2.1.1 The Vertex enumeration problem

The key problem that we are dealing with, is finding vertex representation from half-space representation. Straightforward solution would be:

- Get set $I \subseteq 1, ..., m$
- Choose I rows from $\mathbf{A}$ and solve $\mathbf{A}_I x = b_I$ and $\mathbf{C}x = d$ at the same time
- If only one solution x exists and $\mathbf{A}x \leq b$ still holds, than x is a vertex

The problem of this approach is that the complexity for some polyhedrons grows exponentially in m [7].

Some algorithms solve this problem and are capable to reduce the complexity quite significantly. For example:

- Lexicographic reverse search [8]
- Double Description Method [8]
- A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra [9]

All of the mentioned algorithms are based on the simplex method. However, the simplex method is for solving linear programming problems and not finding the vertices of a polyhedron. The task can be transformed into a linear programming problem. According to [10], the algorithms use more complex methods but one straight forward to show is:

$$\begin{aligned} &\text{minimize } x_0 \\ &\text{subject to } -x_0 + \sum_{j=1}^{n} a_{ij} \leq b_i \quad i \in \{1, 2, ..., m\} \\ &x_0 \geq 0 \end{aligned} \tag{2.6}$$

This problem can be solved using not only the simplex method, but all more sophisticated methods for solving linear programming deriving from the simplex method. Since the algorithms are quite complex and it is not the aim of this thesis to describe these algorithms, they are just mentioned but not looked upon more closely.

2.2 Introduction of a coalitional game

As defined in [3] and [4], the player set is denoted as

$$N := \{1, ..., n\} \tag{2.7}$$

where $n \in \mathbb{N}$. A coalition is a subset $A \subseteq N$. The coalition in which all players are involved is called the grand coalition and is denoted by N. The empty coalition is labeled as $\emptyset$ and if a coalition contains only i-th player, it is marked as $\{i\}$. The power set is the set of all coalitions

$$P(N) := \{A | A \subseteq N\} \tag{2.8}$$

A coalitional game is defined as a pair (N, v) where v is a function

$$v : P(N) \to \mathbb{R} \tag{2.9}$$

such that $v(\emptyset) = 0$. Number $v(A)$ is called the worth of A and it can be interpreted as a utility, or cost, associated with the formation of A. A coalitional structure is a partition

$$S = \{A_1, ..., A_k\} \tag{2.10}$$

of N such that the union of all A_i is N, none of A_i is empty set and intersection of all pairs (A_i, A_j) is empty. The total utility of S is then

$$V(S) = \sum_{i=1}^{k} v(A_i) \tag{2.11}$$

An allocation is a vector $x = (x_1, ..., x_n) \in \mathbb{R}$. If x is allocated to players, coalition A obtains

$$x(A) = \sum_{i \in A} x_i \tag{2.12}$$

The solution of a game v is some set of allocations $x \in \mathbb{R}$. It reflects various aspects of economic rationality, fairness assumptions, or stability. Solutions can be single-valued or multi-valued. Let Γ be the set of all coalitional games with a fixed player set N. Solution is a mapping

$$\sigma : \Gamma \to P(\mathbb{R}^n) \tag{2.13}$$

where $P(\mathbb{R}^n)$ is the family of all subsets of $\mathbb{R}^n$.

2.2.1 Properties

Again, as stated in [3], we say that a coalitional game v is

- Monotone if $v(A) \leq v(B), \forall A \subseteq B$
- Additive if $v(A \cup B) = v(A) + v(B), \forall A \cap B = \emptyset$
- Superadditive if $v(A \cup B) \geq v(A) + v(B), \forall A \cap B = \emptyset$
- Supermodular if $v(A \cup B) + v(A \cap B) \geq v(A) + v(B)$
- Simple if $v(A) \in \{0, 1\}$ and v is monotone and $v(N) = 1$

An allocation x in a coalitional game v is

- Efficient if $x(N) = v(N)$
- Coalitionally rational of $x(A) \geq v(A), \forall A \subseteq N$
- Individually rational if $x_i \geq v(\{i\}), \forall i \in N$

2.3 Common solutions of a coalitional game

There are many solutions and representations of coalitional games, in this section we describe the core and the Weber set since we need them to introduce the main aim of this thesis, the intermediate set.

2.3.1 The Core

A core of a coalitional game is a possible and commonly used solution concept. It is the set of all payoff allocation vectors that are efficient and individually, coalitionally rational. In other words, the core is a set of efficient allocations upon which no coalition can improve [4]. It is defined as:

$$\begin{aligned} C(v) = \{x \in \mathbb{R}^n | x(N) = v(N), \\ x(A) \geq v(A), \forall A \subseteq N\} \end{aligned} \tag{2.14}$$

The core is a convex polyhedron in $\mathbb{R}^n$ of dimension $\leq n-1$ and has a representation

$$C(v) = \text{conv}\{x_1, ..., x_k\}, \tag{2.15}$$

where $x_1, ..., x_k$ are the vertices of $C(v)$. However, there is a possibility that no such vector is found and thus the core is empty. We decide nonemptiness using a linear program [4]:

- Minimize $x_1 + ... + x_n$
- subject to $\sum_{i \in A} x_i \geq v(A), \forall A \subseteq N$

The following statements are equivalent

- The optimal value is $v(N)$
- $C(v) \neq \emptyset$

2.3.2 The Weber set

Let Π_n be the set of all permutations π of the player set N. Let $v \in \Gamma(N)$ and $\pi \in \Pi_n$. A marginal vector of a game v with respect to π is the payoff vector $x^v(\pi)(N) \in \mathbb{R}^n$ with coordinates

$$x_i^v(\pi) = v\left(\bigcup_{j \leq \pi^{-1}(i)} \{\pi(j)\}\right) - v\left(\bigcup_{j < \pi^{-1}(i)} \{\pi(j)\}\right), i \in N. \tag{2.16}$$

The Weber set of v is the convex hull of all the marginal vectors of v,

$$W(v) = \text{conv}\{x^v(\pi) | \pi \in \Pi_n\}. \tag{2.17}$$

Since $x^v(\pi)(N) = v(N)$, the Weber set is a solution on $\Gamma(N)$. Moreover, the inclusion $C(v) \subseteq W(v)$ holds true for every $v \in \Gamma(N)$ [1]. Downside of the Weber set is that it contains too much information, much of it is to no use, since it contains payoff allocations that does not really make any sense what so ever.

2.4 The Intermediate set

Defined in [1], the intermediate set lies in between the core and the Weber set. It is a non-convex solution containing efficient payoff vectors that depend on some chain of coalitions and marginal coalitional contributions with respect to the chain. It is denoted as $M(v)$.
Let $v \in \Gamma(N)$. Then:

- $M(v) \neq \emptyset$
- $C(v) \subseteq M(v) \subseteq W(v)$
- $W(v) = \text{conv}(M(V))$
- v is supermodular if and only if $C(v) = M(v)$

2.4.1 Characterization by chains

From [1], a coalitional chain is a subset

$$H = \{A_1, ..., A_k\} \text{ of } 2^N \tag{2.18}$$

such that

$$\begin{gathered} k \geq 1,\ A_1 \neq \emptyset,\ A_i \subset A_{i+1} \\ \text{for } i = 1, ..., k-1 \text{ and } A_k = N \end{gathered} \tag{2.19}$$

We assume $A_0 := \emptyset$. Let F be the set of all coalitional chains in 2^N. The family F is associated with the following scheme of allocating payoffs $x \in \mathbb{R}^n$ among players in a game v:

- The players' organization into chain $H = \{A_1, ..., A_k\}$ depends on their position in the allocation process.
- Each coalition $A_i \setminus A_{i-1}$ can distribute the total amount

  $$x(A_i \setminus A_{i-1}) = v(A_i) - v(A_{i-1}) \tag{2.20}$$

  to its members, for all $i = 1, ..., k$
- No coalition $B \subseteq (A_i \setminus A_{i-1})$ can improve upon x while respecting the order of coalitions given by H, that is

  $$x(B) \geq v(A_{i-1} \cup B) - v(A_{i-1}) \tag{2.21}$$

  for all $B \subseteq (A_i \setminus A_{i-1})$ and all $i = 1, ..., k$

  For any chain, we may now define a subset

  $$M_H(v) = \{x \in \mathbb{R}^n | x \text{ satisfies 2.20 and 2.21}\} \tag{2.22}$$

  And for whole family F the intermediate set is defined as

  $$M(v) = \bigcup_{H \in F} M_H(v) \tag{2.23}$$

The biggest concern is the number of families since it grows as fast as the ordered Bell number [11]. For example, there are 57 003 chains in the family for a ten-player game. For every one of these chains we need to compute the polyhedron that it represents and as stated before, this is not an easy task either. Table for the first five ordered Bell numbers:

Number of players	Number of chains
1	1
2	3
3	13
4	51
5	181

Table 2.1: Ordered Bell number

2.5 Example

To show what all of this means, the glove game from [1] is used as an example. Glove game is a type of coalitional game where the aim is to have a pair of gloves. Some players have only left glove, some players have only right glove. A coalition is needed to have a pair. Our example is simple, containing only three players, so the player set is

$$N = \{1,\ 2,\ 3\} \tag{2.24}$$

Player 1 owns a single left glove, while the remaining players, 2 and 3, owns a single right glove each. The profit of a coalition $A \subseteq N$ is the number of glove pairs the coalition owns. In this particular example, the reward function is

$$v(a) = \begin{cases} 1 & \text{if } A \in \{\{1,2\},\{1,3\},N\} \\ 0 & \text{otherwise} \end{cases} \tag{2.25}$$

First, let's compute the Weber set. It describes every possible distribution of the reward, in this case it is a triangle with vertices $(1,\ 0,\ 0)$, $(0,\ 1,\ 0)$ and $(0,\ 0,\ 1)$.

$$W(v) = \text{conv}\{(1,0,0),\ (0,1,0),\ (0,0,1)\} \tag{2.26}$$

The visual representation:

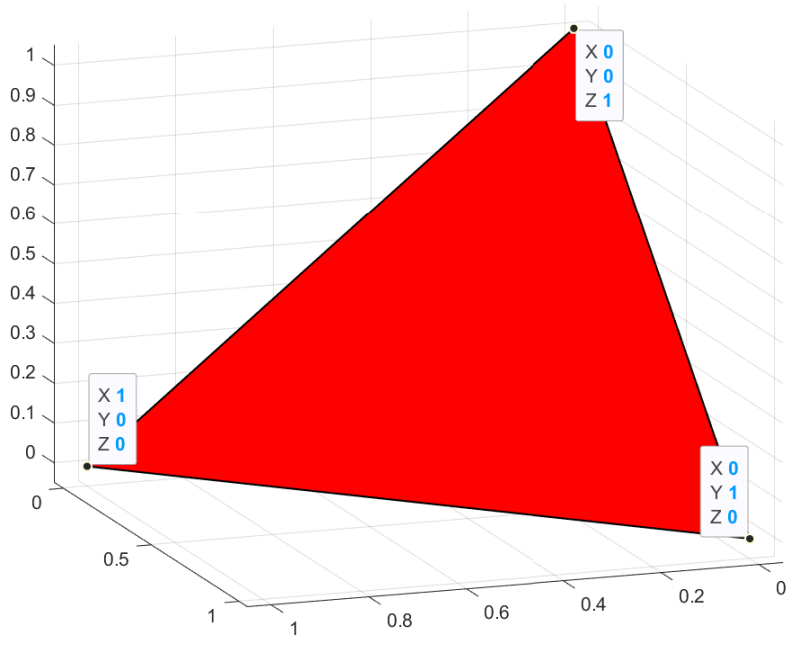

Figure 2.1: Weber set, glove game

The core in this particular game is a single point.

$$C(v) = \{(1, 0, 0)\} \tag{2.27}$$

All the reward goes to player 1, as can be seen. This is because there is a surplus of right-hand gloves on the market, so players 2 and 3 take any arbitrary low payoff, zero. Obviously, they get an equal reward since they both have only the right-hand glove, and if they did not get zero, one would be paid for not contributing at all. Because of that, the first player would be losing his reward. The core maximizes and rationally distributes the reward, so the most stable solution for the grand coalition is that the first player gets all the reward.

The intermediate set is composed of two lines, one connecting $(1,\ 0, 0)$ and $(0,\ 1,\ 0)$, other connecting $(1,\ 0,\ 0)$ and $(0,\ 0,\ 1)$. It represents reward distribution in 2 player coalition, since there is no surplus of anything in this coalition so the reward between the 2 player can be anything from 0 to 1. It is up to external factors like strategies or prices.

$$M(v) = \text{conv}\{(1, 0, 0),\ (0, 1, 0)\} \cup \text{conv}\{(1, 0, 0),\ (0, 0, 1)\} \tag{2.28}$$

The visual representation:

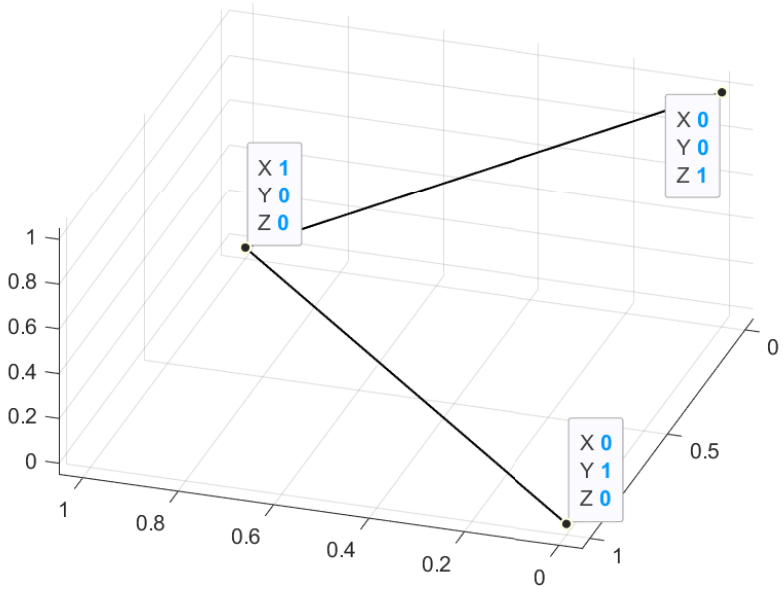

Figure 2.2: The intermediate set, glove game

Chapter 3

Implementation

In this chapter, the implementations in MATLAB and Python are presented and compared. The basic idea behind the computation itself is the same, although they differ in game representations and in the construction of chains, families, inequalities, and equations.

3.1 Game representation

In MATLAB, games are represented by a specially ordered vector, in Python, a dictionary is used.

3.1.1 MATLAB

Every coalitional game is represented by a vector of rewards for all coalitions v. These coalitions are represented by bit stream. The streams are ordered in lexicographical order using matrix, for example:

player 1	player 2	player 3
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(3.1)

Where 1 means a player is present and 0 absent from the coalition. In other words, every row represents a coalition, so each number in the vector v represents a reward for that coalition in that row. It is implemented as a recursive function that builds the matrix from the most upper right corner. From there it is built by doubling itself down and adding a new column to the left. Pseudocode:

```
generate_bin_matrix(n):
    if n == 1 :
        A = [0; 1];
```

```
else:
    rest = generate_bin_matrix(n - 1);
    rest = [rest; rest];
    #append last under itself
    current_c_0 = zeros(1, 2^(n - 1));
    #column of 0
    current_c_1 = ones(1, 2^(n - 1));
    #column of 1
    current_c = [current_c_0 current_c_1];
    #join them
    current_c = transpose(current_c);
return [current_c rest]; #join it all
```

3.1.2 Python

In Python, coalitional games are represented as dictionaries. It is more practical and more natural. The downside is that it takes more work for complex games to write a whole dictionary rather than a simple vector. The key is a tuple representing a coalition, and a value is a reward for that particular coalition. Coalitions not present in the keys are rewarded with a reward equal to 0. As an example, the glove game is used, the same one that was used before. For this game it would look like this:

coalition	reward
(0, 1)	1
(0, 2)	1
(0, 1, 2)	1

(3.2)

Since this is Python, players are indexed from 0, not 1. This representation is way more practical for games with fewer players and primarily for games where a lot of coalitions are rewarded with 0.

3.2 Generating family

As stated, we use representation by chains to compute the intermediate set of a coalitional game. We need to generate all of those chains H and thus get the family F. As previously shown, the number of these chains grows quite rapidly.

3.2.1 MATLAB

In MATLAB implementation of getting a family for a game is divided into multiple helper functions. The first one is finding all the coalitions, including given active players. In other words, find all supersets that include a given subset of players. Pseudocode:

```
find_coalitions(matrix, vector)
    positive = all((matrix - ones(size(matrix, 1), 1)* vector)>=0, 2)
    # from every row the vector is substracted and all rows that
    # still have all positive or equal to zero are selected
    return matrix(positive, :)
```

If we found all coalitions B such that $A \subseteq B, \forall A$, we can generate all chains and join them to make family F. Pseudocode:

```
gen_family(acc_cur, bin_matrix, current):
    if sum(current) > 0:
        acc_cur = [acc_cur; current]

    next_matrix = find_coalitions(bin_matrix, current)
    cell = {}
    if size(next_matrix, 1) == 1:
        cell{end + 1} = acc_cur
    else:
        next_matrix = next_matrix(2 : end, :)
        for i = 1 : size(next_matrix, 1):
            cell = [cell gen_family(acc_cur, next_matrix, next_matrix(i, :))]
            #recursively add rest of the chains to the family

    return cell
```

Now the final initialization function get_family is called. Pseudocode:

```
get_family(n):
    bin_matrix = generate_bin_matrix(n)
    init = bin_matrix(1, :)
    return gen_family([], bin_matrix, init)
```

We initialize the process with a vector of zeros since all coalitions are a superset to this one, where no player is active.

3.2.2 Python

In Python, helper functions are also needed. First one generates all possible coalitions from player power set N, for 3 players it is (0, 1, 2), except for the power set itself, that would be counterproductive later. For this, library *itertools* was used, *combinations* and *chain*. The pseudocode is not needed since it is just a call of *chain* and *combinations*.

Next, a filter function is needed to find all supersets of a given subset of players. In Python, since the game is represented by a dictionary, this function is different than in MATLAB implementation. Pseudocode:

```
filter_func(curr, N):
    ret = []
    for coalition in N:
```

```
        check = True
        if len(curr) >= len(coalition):
            continue
            #only longer coalition can
            #be superset of current one
        for num in curr:
            if num not in coalition:
                check = False
                #check if coalition that is being
                #looked upon contains current one
        if check:
            ret.append([coalition])
    return ret
```

With the filter function, all chains containing a specific coalition may be generated:

```
get_chains(chain_curr, curr, N):
    acc = []
    containing = filter_func(curr, N)
    #find all coalitions containing current one
    for new in containing:
        chain_new = chain_curr.copy()
        chain_new.extend(new)
        #add coalition to chain
        acc.append(chain_new)
        acc.extend(get_chains(chain_new, new, N))
        #append to accumulator and get rest of the chains
        #that extend this one

    return acc
```

This way, the order of coalition in a chain is maintained according to a definition. Using all helper functions mentioned above, a family can be generated:

```
get_family(N):
    coals = coalitions(N)
    ret = []
    for coal in coals:
        ret.append([coal])
        ret.extend(get_chains([coal], coal, coals))

    for sub_chain in ret:
        sub_chain.append(N)
    ret.append([N])

    return ret
```

3.3 Calculating subset

For constructing the whole intermediate set M we need subsets M_H. The basic idea is to build the inequalities and the equations according to the condition stated above. This problem is approached slightly differently in MATLAB and Python implementation. After the mentioned inequalities and equations are built, they are passed to libraries used to calculate the vertex representation of the polyhedron for the particular chain.

3.3.1 MATLAB

The computation was done using *polyhedron* from *MPT 3.0* library [12]. Some of the helper functions shown above are used, mainly to find all coalitions B according to 2.21 and after that create the inequalities and equations, respectively. Pseudocode:

```
get_subset(b, n, chain):
    bin_matrix = generate_bin_matrix(n)
    bin_matrix = bin_matrix(2 : end, :)
    # trimming first row of zeros
    Ae = []
    be = []

    for i = 1:size(chain, 1):
    #Inequalities
        vector = chain(i, :)
        value = b(ismember(bin_matrix, chain(i, :),'rows'))
        if i ~= 1:
        #first one is handled slightly differently,
        #since there is no coalition on i-1 position
            vector = vector - chain(i - 1, :)
            value = value - b(ismember(bin_matrix, chain(i - 1, :),'rows'))

        Ae = [Ae; vector]
        #append left side
        be = [be; value]
        #append right side

    A = []
    b_ = []
    for i = 1:size(chain, 1):
    #Equations
        vector = chain(i, :)
        if i ~= 1:
            vector = vector - chain(i - 1, :)
        n_active = sum(vector)
```

```
        bin_help = generate_bin_matrix(n_active)
        bin_help = bin_help(2:end, :)
        coalitions = zeros(size(bin_help, 1), n)
        indexes = find(vector)
        for j = 1:n_active:
            coalitions(:, indexes(j)) = bin_help(:, j)
        b_i = []
        for j = 1:size(coalitions, 1):
            temp_idx = coalitions(j, :)
            v2 = 0
            if i ~= 1:
            #again, first one is handled slightly differently
                temp_idx = temp_idx + chain(i - 1, :)
                index_2 = ismember(bin_matrix, chain(i - 1, :),'rows')
                v2 = b(index_2, :)
            index_1 = ismember(bin_matrix, temp_idx,'rows')
            v1 = b(index_1, :)
            b_i = [b_i; v1 - v2]
        A = [A; coalitions]
        #append left side
        b_ = [b_; b_i]
        #append right side

    P = Polyhedron('A', -A, 'b', -b_, 'Ae', Ae, 'be', be)
    #polyhedron that represent intermediate subset
    P.minHRep()
    P.irredundantHRep
    #getting rid of redundancies
    P.computeVRep
    #computation of vertex representation

    return P
```

This returns polyhedron with vertex representation calculated that represents the chain in the intermediate set.

3.3.2 Python

Since the game was represented by a dictionary, it needs to be converted into inequalities and equations. After that, matrices for both are completed and can be computed. For matrix manipulation library *numpy* [13] was used and for calculation of vertex representation library *cdd* [14] was used. According to the documentation of *cdd*, the inequalities needs to be shaped like

$$\mathbf{A}x \leq b \tag{3.3}$$

and that differs from [**?**], so the inequalities are needed to be multiplied by -1. After that, these are then shaped into matrix form, again according to

the documentation:

$$[b \ -\mathbf{A}] \tag{3.4}$$

or rather in our case

$$[-b \ \mathbf{A}] \tag{3.5}$$

The code is not provided since the idea is the same as in MATLAB. The main difference here is just the creation of the matrices from dictionaries. That is achieved mainly using tools provided by *numpy*. Then those matrices are converted to *cdd* matrix, and inequalities are marked with linear attribute set to false, and equations have set this attribute to true. After that, this matrix is used to create *cdd* polyhedron object, and vertex representation is computed by this object.

3.4 Intermediate set

Finally, the computation of intermediate subsets is called iteratively for every chain. Results are stored in an array and returned. In MATLAB, they are united in a polyhedron union. In Python there are no such options, so the intermediate set is returned as an array of polyhedrons.
MATLAB pseudocode:

```
get_intermediate_set(n, b):
    F = get_family(n)
    for i = 1:size(F, 2):
        P(i) = get_subset(b, n, cell2mat(F(i)))
        # for every coaliton generate chain and intermediate subset
        # core is an intermediate subset for chain containing only N

    return PolyUnion(P)
    # finally join them into one object and return
```

As said, the idea behind Python implementation is the same, so there is no need to mention the same code in Python.

Chapter 4

Examples

In this chapter, more examples are shown to understand better what the intermediate is, how many polyhedrons it can contain, that it indeed is not a convex object and how it differs from the core and the Weber set. Examples four through six solve calculating extreme rays over four variables. A minimal integer representative for each of the equivalence classes is provided. No visual representation is provided here since these games are four-player, so the space where the polyhedron occurs is four-dimensional, which is quite challenging to visualize. Those three examples show that even for unbounded polyhedrons, the intermediate set can be computed. The last example is a ten-player game. The reason is purely to show the difference between implementations and to show how big the solution can be.

4.1 Example 1

Consider a game, majority voting from [4], with a player set

$$N = \{1,\ 2,\ 3\} \tag{4.1}$$

As the name suggests, it is a game with a reward of either 1, the voting passed, the majority of players agreed, or 0, not enough player agreed so the voting did not pass. So the reward function is

$$v(a) = \begin{cases} 1 & \text{if } |A| \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

The Weber set is the same as in the glove game since as the convex cover of intermediate set does not change.

$$W(v) = \text{conv}\{(1,0,0),\ (0,1,0),\ (0,0,1)\} \tag{4.3}$$

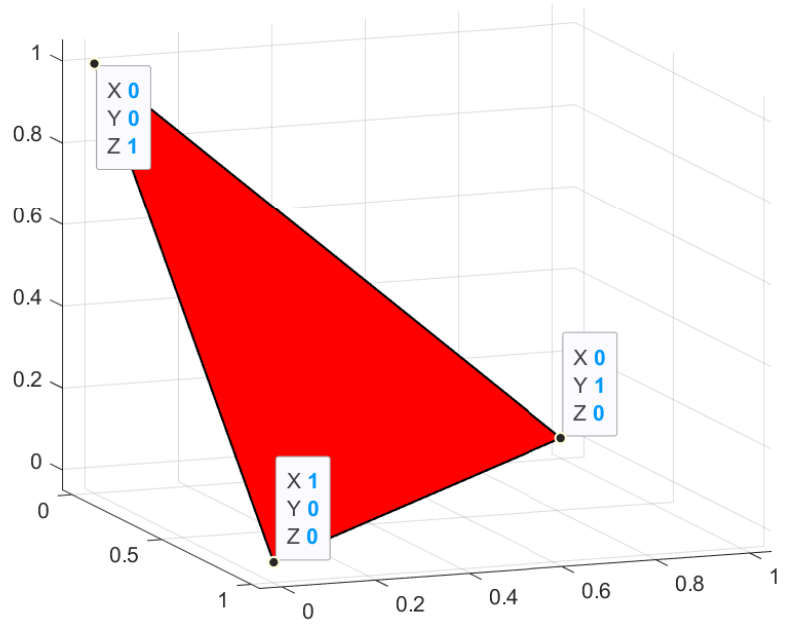

Figure 4.1: Weber set, example 1

The core is empty in this case. It is due to the redundancy of one in the coalition N. In the glove game, player 1 was more important because it held the left glove, the only left glove in the game. However, here all players are equal, so they cannot get all one third, because one is still redundant. But than there is no one to get a lower reward, since, as said, they are all equal so their reward should be equal. This problem has no rational, stable, and fair solution. So the core is

$$C(v) = \emptyset \tag{4.4}$$

The intermediate set, despite the core being smaller, is larger. It is extend by a line connecting $(0,\ 1,\ 0)$ and $(0,\ 0,\ 1)$. The point still stands from the glove game, and just all three two-player coalitions now hold the reward of 1, not just two as it was in the previous example.

$$\begin{aligned} M(v) =& \text{conv}\{(1,0,0),(0,1,0)\}\cup \\ & \text{conv}\{(1,0,0),(0,0,1)\}\cup \\ & \text{conv}\{(0,1,0),(0,0,1)\} \end{aligned} \tag{4.5}$$

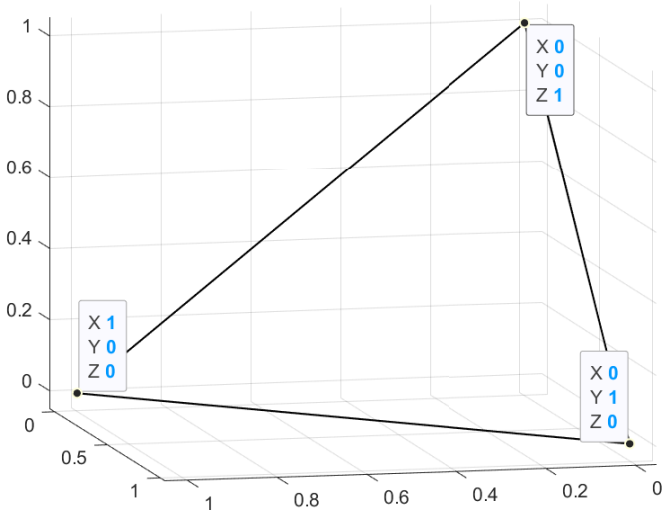

Figure 4.2: The intermediate set, example 1

4.2 Example 2

In this example, the reward is based on how many players are in a coalition. If a player is alone, his reward is zero. For two players, two points, and for three players, three points. The reward function [1]:

$$v(a) = \begin{cases} 2 & \text{if } |A| \geq 2 \\ 3 & \text{if } |A| \geq 3 \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

The Weber set is a hexagon. Its vertices lie in all permutations of the payoff vector $(0,\ 1,\ 2)$.

$$\begin{aligned} W(v) = \text{conv}\{&(2,1,0),\ (1,2,0),\ (0,2,1),\\ &(0,1,2),\ (1,0,2),\ (2,0,1)\} \end{aligned} \tag{4.7}$$

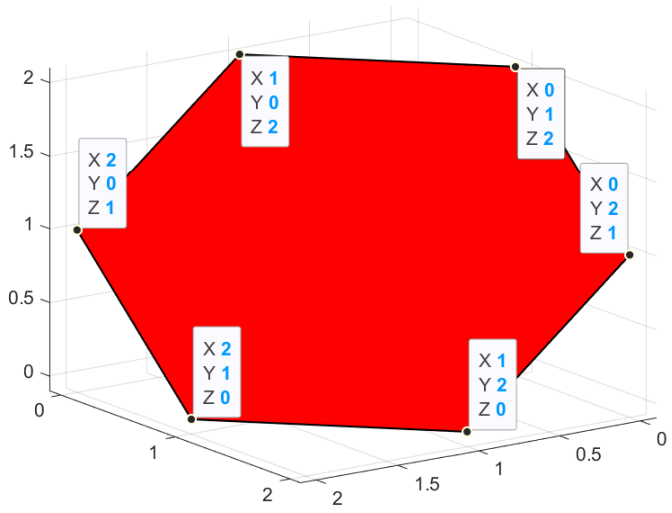

Figure 4.3: The Weber set, example 2

The core is a single point, and it lies in the middle of the hexagon, the Weber set. It also is crossing of all lines from the intermediate set, as it is shown after the core.

$$C(v) = (1, 1, 1) \tag{4.8}$$

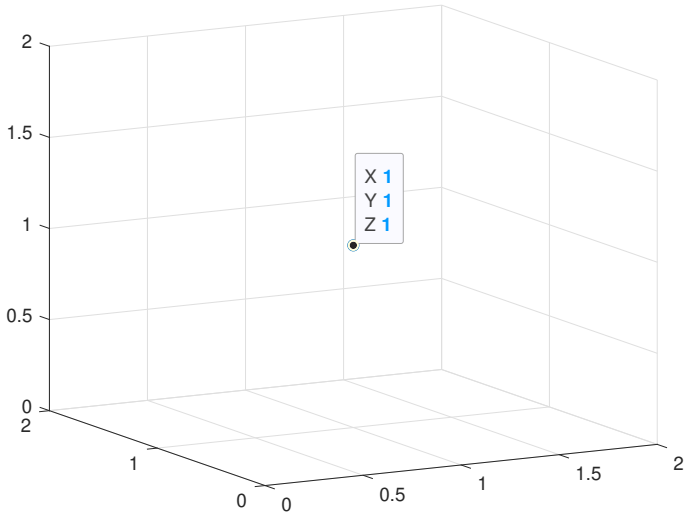

Figure 4.4: The core set, example 2

The intermediate set is composed of all lines connecting opposite vertices

of the Weber set. We obtain that $M_{i,N}(v) = \emptyset$ for every chain $\{i, N\}$.

$$\begin{aligned} M(v) =& \text{conv}\{(2,1,0),\ (1,2,0)\} \cup \\ & \text{conv}\{(0,2,1),(0,1,2),\} \cup \\ & \text{conv}\{(1,0,2),\ (2,0,1)\} \end{aligned} \tag{4.9}$$

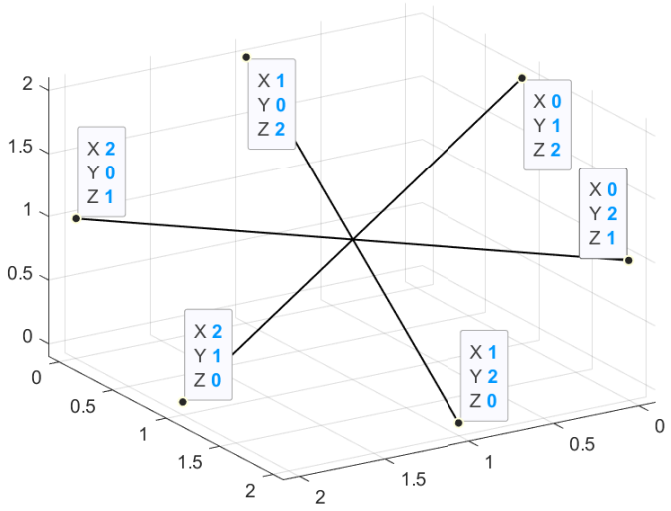

Figure 4.5: The intermediate set, example 2

4.3 Example 3

This game is similar to the previous one. The only change is that for coalitions with size two, the reward is downgraded to only a point [4].

$$v(a) = \begin{cases} 1 & \text{if } |A| \geq 2 \\ 3 & \text{if } |A| \geq 3 \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

However, a slight change like this can have a massive impact on the intermediate set and the core. Since it is not profitable for a two-player coalition anymore and the most reasonable solution is always the grand coalition (1, 2, 3). This results in the Weber set, the core and the intermediate set being identical.

$$\begin{aligned} & W(v) = C(v) = M(v) = \\ & \text{conv}\{(2,1,0),\ (1,2,0),\ (0,2,1), \\ & \qquad (0,1,2),\ (1,0,2),\ (2,0,1)\} \end{aligned} \tag{4.11}$$

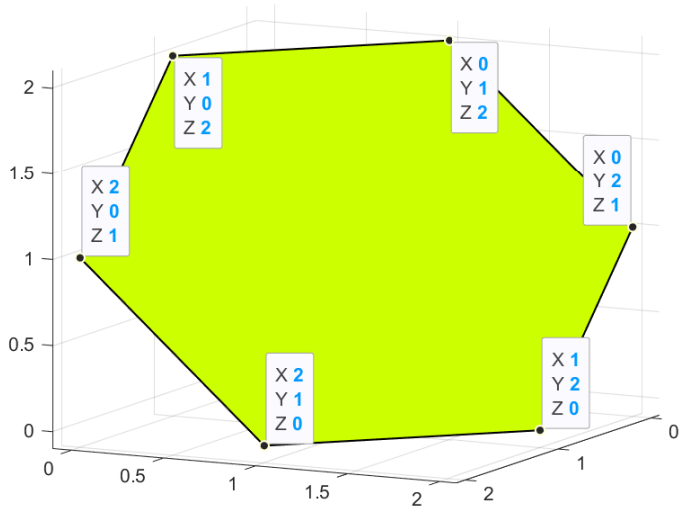

Figure 4.6: The Weber set, the core and the intermediate set, example 3

4.4 Example 4

This game has the reward function according to this graph:

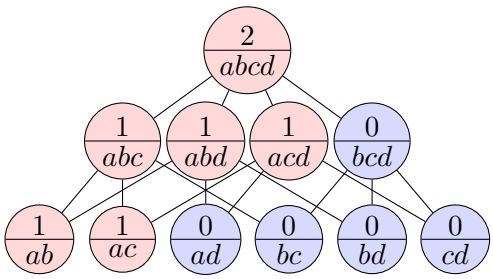

Figure 4.7: Example 4

Where the upper part of the node is the reward and the lower part is the coalition.

The family consists of 51 chains, so the intermediate set is composed by 51 polyhedrons, or less, since some of the subsets may be empty. The final union:

$$\begin{aligned} M(v) =& \text{conv}\{(1,1,0,0),\ (0,1,1,0),\ (1,0,1,0),\ (2,0,0,0),\ (1,0,0,1)\} \cup \\ & \text{conv}\{(1,0,0,1),\ (1,0,1,0),\ (0,1,1,0),\ (0,1,0,1)\} \cup \\ & \text{conv}\{(1,0,0,1),\ (1,1,0,0),\ (0,1,1,0),\ (0,0,1,1)\} \end{aligned} \tag{4.12}$$

4.5 Example 5

The reward function represented by graph:

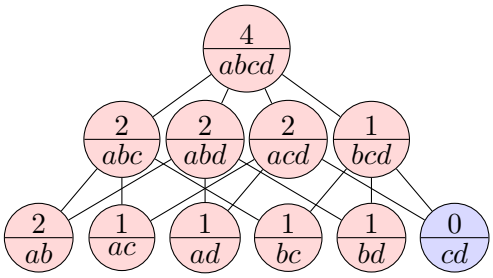

Figure 4.8: Example 5

By adding more rewards, more coalitions are more likely to be rational, so the intermediate set grows significantly.

$$
\begin{aligned}
M(v) = \operatorname{conv}\{&(1,1,2,0),\ (0,2,1,1),\ (2,0,1,1),\ (2,1,0,1),\\
&(1,2,0,1),\ (3,1,0,0),\ (1,3,0,1)\}\cup\\
\operatorname{conv}\{&(2,0,1,1),\ (2,0,2,0),\ (0,2,2,0),\ (0,2,1,1)\}\cup\\
\operatorname{conv}\{&(1,2,0,1),\ (1,3,0,0),\ (0,3,1,0),\ (0,2,1,1)\}\cup\\
\operatorname{conv}\{&(1,1,2,0),\ (1,3,0,0),\ (0,3,0,1),\ (0,1,2,1)\}\cup\\
\operatorname{conv}\{&(1,1,2,0),\ (1,3,0,0),\ (0,3,0,1),\ (0,1,2,1)\}\cup\\
\operatorname{conv}\{&(2,1,0,1),\ (3,1,0,0),\ (3,0,1,0),\ (2,0,1,1)\}\cup\\
\operatorname{conv}\{&(1,1,2,0),\ (3,1,0,0),\ (3,0,0,1),\ (1,0,2,1)\}\cup\\
\operatorname{conv}\{&(0,3,1,0),\ (0,2,1,1)\}\cup\\
\operatorname{conv}\{&(0,3,0,1),\ (0,1,2,1)\}\cup\\
\operatorname{conv}\{&(1,1,2,0),\ (0,1,2,1)\}
\end{aligned}
\tag{4.13}
$$

As can be seen, even after the reduction of redundant points and polyhedrons, the result is still a union of 10 polyhedrons. The rest of the polyhedrons belonging to the rest of the chains are contained in those ten.

4.6 Example 6

The last of the examples on extreme rays. Again, represented by a graph.

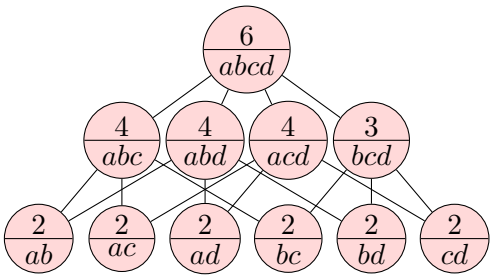

Figure 4.9: Example 6

This intermediate set, after the union, is made of twelve different polyhedrons. The solution is quite long and just from written vectors, quite hard to imagine. Because of this, I do not see it as necessary to provide a solution. All the examples above were computed by both implementations. Both handled it and managed to produce the solution. Generally, the implementation in Python was faster than the implementation in MATLAB.

4.7 Example 7

As mentioned in the introduction to this chapter, this is the last example, a ten-player game. This game is in [15], example XI.4, page 253. The game is represented in ten-dimensional space, the computation is quite difficult. Implementation in MATLAB could not finish this computation and produce a result even after significant time given, about two hours. MATLAB used all of the sixteen Gigabytes of RAM and stalled after that. Implementation in Python handled it in about two to five minutes, depends on processes in the background. Comparison is discussed further in the final chapter. The reward function was:

$$\begin{aligned} v(N) &= 5, \\ v(\{1,3,5,7,9\}) &= 4, \\ v(\{3,5,7,9\}) &= v(\{1,5,7,9\}) = v(\{1,3,7,9\}) = 3, \\ v(\{3.5.7\}) &= v(\{1,5,7\}) = v(\{1,3,7\}) = 2, \\ v(\{3,5,9\}) &= v(\{1,5,9\}) = v(\{1,3,9\}) = 2, \\ v(\{1,2\}) &= v(\{3,4\}) = v(\{5,6\}) = 1, \\ v(\{7,8\}) &= v(\{9,10\}) = 1, \\ otherwise\ v &= 0 \end{aligned} \tag{4.14}$$

The family consists of 57 003 chains. For every one of these chains, a polyhedron had to be computed. After union and reduction of redundant polyhedrons, the intermediate set consists of eight hundred sixty two unique polyhedrons. Polyhedron with highest dimensionality, which is by definition $n - 1$, in this case 9, is a convex cover of two hundred points.

Chapter 5

Conclusion

In the last chapter a comparison between MATLAB and Python implementations form my point of view are given. The time required and difficulties i had with the implementation itself is analyzed, as well as time and efficiency of the computation. Lastly, the whole thesis is summed up.

5.1 Comparison

To begin with, I would like to say that I have much more experience with Python than I have with MATLAB, so this can influence my opinion and my views. I started with MATLAB, and I designed the matrices here. The most significant trouble was to design a way to represent the reward function since MATLAB does not have dictionaries, or at least I am not aware of such a thing. As mentioned before, I chose to represent it as a vector, lexicographically ordered by a binary matrix. Next, I had to find a way to find and select certain coalitions from this matrix to form chains, so I had to implement the find function. All these problems were much simpler in Python since in itself, Python has many handy tools to work with dictionaries and arrays. If such function was not in default Python, it was found in library *numpy* or implemented using said library. Thus the preparation for the computation itself was more straightforward in Python.

In the computation, I had to build matrices with inequalities and equations. This was again simpler in Python. Also, since I was building them row by row, it was more efficient in Python because MATLAB does not cope well with appending a row to matrix every iteration, it is not memory efficient. This may have been caused by my inexperience, and it might be solved quite easily however, as I said, I am not that experienced. Using Python and *numpy*, building the matrices was not hard. Just a few simple tricks were more than enough to form them:

```
for i in range(len(chain)):
    coal = chain[i]
    value = game[coal] if coal in game else 0
    if i > 0:
        coal = list(set(coal).difference(set(chain[i - 1])))
```

```
            if chain[i - 1] in game:
                value = value - game[chain[i - 1]]
        ae_row = np.zeros(len(N) + 1)
        ae_row[np.array(coal) + 1] = 1
        #"activate" all players in coalition
        ae_row[0] = -value
        #give reward to coalition
        #Matrix in specific for to use with cdd
        Ae.append(ae_row)
```

This is a short example of how the equation part was done in Python. To me, it was more natural and intuitive.

The computation itself was faster in Python, and it used less RAM memory in the process. From all the experiments done, the most demanding was the ten-player game mentioned in examples, and, as said, implementation in Python handled it in about two to five minutes. For MATLAB, the 16GB of RAM memory in my computer was not sufficient, so this implementation did not produce any result. However, MATLAB handled the rest of the experiments well, although it was slightly slower than Python. Both implementations produced the same results in terms of correctness.

5.2 Conclusion

The coalitional game theory examines and predicts the formation of coalitions between players and splitting payoffs between them. Common solution concepts are the core and the Weber set. We introduced the intermediate set as a solution lying somewhere in between them. The intermediate set is always nonempty, unlike the core, so it brings at least some information. On the other hand, unlike the Weber set, it is not unnecessarily large. It is a core-like solution but not for only grand coalition N, but also for the whole family of chains F. The intermediate set is generally not a convex polyhedron, unlike the core and the Weber set. It is a union of convex polyhedrons computed for every chain, which, as stated, makes the computation quite costly. Implementation of the computation of the intermediate set has been made using MATLAB with the use of library *MPT 3.0* and Python with the help of libraries *numpy* and *cdd*. The latter was used for the computation itself. In the thesis, the pseudocode has been demonstrated. For future work, both implementations can be optimized, especially the one using MATLAB.

Bibliography

[1] ADAM, Lukáš and Tomáš KROUPA. *The intermediate set and limiting superdifferential for coalitional games: between the core and the Weber set. International Journal of Game Theory* [online]. 2017, 46(4), 891–918. Available from: https://doi.org/10.1007/s00182-016-0557-3

[2] PELEG, Bezalel a Peter SUDHÖLTER. *Introduction to the Theory of Cooperative Games*. Second Edition. Berlin, Germany: Springer-Verlag, 2007. ISBN 978-3-540-72944-0.

[3] KROUPA, Tomáš *Multiagent Systems - Two Lectures on Coalitional Game Theory*(slides)(2019) Available from: https://cw.fel.cvut.cz/b191/_media/courses/be4m36mas/coalitional_games_-_lectures.pdf

[4] KROUPA, Tomáš *Multiagent Systems - Coalitional Games and the Core*(slides)(2020) Available from: https://cw.fel.cvut.cz/wiki/_media/courses/be4m36mas/cg01_lectures.pdf

[5] FORCEY, Stefan and Gabriela HAMERLINCK and Logan KEEFE and William SANDS *Chapter 10 - The Minimum Evolution Problem in Phylogenetics: Polytopes, Linear Programming, and Interpretation, Algebraic and Combinatorial Computational Biology*. Academic Press, 2019. Edited by Raina Robeva and Matthew Macauley. 319–349. DOI: https://doi.org/10.1016/B978-0-12-814066-6.00010-6

[6] GRÜNBAUM, Branko, *Convex Polytopes*, 2nd edition, prepared by Volker Kaibel, Victor Klee, and Günter M. Ziegler, 2003, ISBN 0-387-40409-0, ISBN 978-0-387-40409-7

[7] KROUPA, Tomáš *Optimization - Convex sets and polyhedrons*(slides)(2020) Available from: https://cw.fel.cvut.cz/b192/_media/courses/b0b33opt/10poly.pdf

[8] AVIS, D., FUKUDA, K. *A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra*. Discrete Comput Geom 8, 295–313 (1992). Available from: https://doi.org/10.1007/BF02293050

[9] APTE, Jayant. *Polyhedral Computation* [online]. 2012. Available from: https://faculty.coe.drexel.edu/jwalsh/JayantCHM.pdf

[10] CHVÁTAL, Václav. *Linear Programming*. New York: W. H. Freeman and Company, 240–249, 1983. ISBN 9780716711957.

[11] KLAZAR, Martin. *Bell numbers, their relatives, and algebraic differential equations. Journal of Combinatorial Theory, Series A*, volume 102, number 1, 04.2003, 63–87. DOI: 10.1016/S0097-3165(03)00014-1 Available from: `https://linkinghub.elsevier.com/retrieve/pii/S0097316503000141`

[12] M. HERCEG and M. KVASNICA and C.N. JONES and M. MORARI. *Multi-Parametric Toolbox 3.0*. Proc. of the European Control Conference, 2013, Zürich, Switzerland, July 17–19, 502–510. Available form: `http://control.ee.ethz.ch/~mpt`

[13] HARRIS, C.R., MILLMAN, K.J., VAN DER WALT, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). Available from: `https://doi.org/10.1038/s41586-020-2649-2` DOI: 10.1038/s41586-020-2649-2

[14] *CDD Documentation* [online]. 2015 Available from: `https://pycddlib.readthedocs.io/en/latest/matrix.html`

[15] OWEN, Guillermo. *Game Theory*. Third Edition. Monterey, California: Academic Press, 1995. ISBN 978-0125311519.

[16] SHOHAM, Y. and K. LEYTON-BROWN. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.