On the Design and Synthesis of Voting Games

Exact Solutions for The Inverse Problem

Bart de Keijzer

On the Design and Synthesis of Voting Games

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Bart de Keijzer
born in Amstelveen, the Netherlands

Algorithmics Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
`www.ewi.tudelft.nl`

On the Design and Synthesis of Voting Games

Author: Bart de Keijzer
Student id: 1221779
Email: B.deKeijzer@student.tudelft.nl

Abstract

In many real-world decision making settings, situations arise in which the parties (or: players) involved must collectively make decisions while not every player is supposed to have an equal amount of influence in the outcome of such a decision. The *weighted voting game* is a model that is often used to make such decisions. The amount of influence that a player has in a weighted voting game can be measured by means of various power indices.

Weighted voting games are part of the more general class of simple games. In this thesis, we study the problem of finding for a given class of simple games (including weighted voting games), the game in which the distribution of the influence among the players is as close as possible to a given target value (i.e. power index). We investigate the posibilities that we have for exactly solving this problem. For the case of weighted voting games, we obtain a method that relies on a new efficient procedure for enumerating weighted voting games of a fixed number of players.

The enumeration algorithm we propose works by exploiting the properties of a specific partial order over the class of weighted voting games, for which we prove existence. The algorithm enumerates weighted voting games of a fixed number of players in time exponential in the number of players, but polynomial in the number of games output. As a consequence we obtain an exact anytime algorithm for designing weighted voting games.

We look at various ways to improve on this algorithm. A large improvement follows by exploiting the properties of two specific types of coalitions, which we refer to as roof coalitions and ceiling coalitions. The algorithm, together with these improvements, has been implemented in order to measure the practical performance and to obtain various data on the class of weighted voting games.

Our method for solving the voting game design problem heavily relies on the ability to transform between different representations of simple games, which we refer to as *voting game synthesis problems*. We give an extensive treatment of these synthesis problems, and in particular we prove that there does not exist a polynomial time algorithm that transforms the list of ceiling coalitions of a game into the list of roof coalitions. However, we also show that an output-polynomial time algorithm for this problem actually does exist, by providing one.

Thesis Committee:

Chair:	Prof. Dr. Cees Witteveen, Faculty EEMCS, Delft University of Technology
University supervisors:	Dr. Tomas Klos, Faculty EEMCS, Delft University of Technology
	Dr. Yingqian Zhang, Faculty EEMCS, Delft University of Technology
Committee members:	Prof. Dr. Ir. Karen Aardal, Faculty EEMCS, Delft University of Technology
	Dr. Haris Aziz, Computer Science Department, Ludwig-Maximilians-University Munich

Preface

Dear reader of this thesis, please allow me to use a more informal writing style just for this preface. In this preface I will write in singular first person form (“I”), while the rest of my thesis is written in plural first person form (“we”), because it just felt more natural to use.

Well, it has been a lot of fun working on this thesis. The research presented in here is my own, but it definitely would have been impossible without the help of a specific set of persons, to whom I will give my thanks right here, in this preface. Additionally, I will thank some other people who are not directly associated with the thesis, but more so with all the things that happened around it, during the time that I was writing all of this.

So first of all, I will thank Dr. Tomas Klos and Dr. Yingqian Zhang. Tomas and Yingqian are my supervisors for this project. They gave me lots of very helpful feedback and support. Next, I must thank Tomas and Yingqian again, for a lot of other things. First of all, they have been really friendly to me and I think they are really nice people. Also, they have given me a tremendous amount of support with finding and (succesfully) applying for a PhD position. I could not have managed it without their help. Next, I thank them for co-authoring two papers with me, of which one paper (not related to the topic of this thesis) has been submitted to and got accepted for the Algorithmic Decision Theory 2009 (ADT2009) conference (for the other paper, the decision of acceptance is still pending as of the time writing this). It is great that this happened. Also, Tomas and Yingqian, yet more thanks for all kinds of help I received from you, related to visiting this conference. And finally, many additional thanks to Tomas for making it possible for me to give a presentation at the ILLC seminar, and the feedback and tips you gave me for giving this presentation. Tomas and Yingqian, you have helped me with a lot of things that are very important to me, and I am super grateful for that. I undoubdetly forgot to mention one or more of these; so, also lots of thanks for everything I forgot to list.

Next, I want to thank Prof. Dr. Ir. Karen Aardal, Dr. Haris Aziz, and Prof. Dr. Cees Witteveen for being in my thesis committee. Without you, it would certainly be impossible to obtain my MSc degree. Additionally, I thank Prof. Dr. Cees Witteveen for having such a good research group, for allowing me to do this final thesis project within this group, and for making it possible for me to attend the ADT2009 conference.

Thanks to everyone in the Algorithmics group. I felt welcomed in the group, and everyone there has been very nice to me.

Thanks to Frits van Campen, Menno den Hollander, Dietger van Antwerpen, Herwin Wels, Kevin Dullemond, Ben van Gameren, Alain van den Berg, and Ricky Lindeman for being such good friends, and for being my co-students during my time at TU Delft. You are the reason that my time while I was studying at Delft, is an unforgettable one. I also want to thank the first four in this list for being the people I worked together with so many times during my time at Delft. Working together with you was great.

I want to thank Xiaofan Sun for the many wuxia-films that she gave to me (I just love wuxia!), for being such a good friend, and for being one of the most inspiring persons that I know. I will never forget you.

I thank all my friends for being my friends. Yoram Duijn, Bart Witteveen, Jan Hommes, Dirk Hommes, Bram Gollin, Rene Corbet, Robin van Halem, Jochum Zijlstra, Daan Eijkel, Matthijs Wolzak, Remco de Lange, Youri van Veen, Phu Do, Marie Jimmink, Melanie Stoop, Lex Groot, Jasper Heinsbroek, and anyone I forgot. You do not have much to do with this thesis or my computer science studies at TU Delft, but without you I could not have come this far.

I want to thank Prof. Dr. Krzysztof Apt and Dr. Ulle Endriss for making it possible for me to give a talk at the Computational Social Choice seminar at the ILLC.

I want to thank Dr. Guido Schaefer for accepting me as a PhD student in algorithmic game theory at the CWI. I look forward to starting my PhD project there.

Thank you, Maria Kyropoulou, Andreas Darmann, and Prof. Dr. Ulrich Pferschy, for turning my visit to the ADT2009 conference into a good experience. It was great to meet you, and thanks to you I really had a very memorable time there. I hope I will see you many more times in the future.

Lastly, of course I thank my parents, and my two sisters, just because I can, and also because you are absolutely great. As my near family, you are still the most important people in my life.

Bart de Keijzer
Delft, the Netherlands
November 20, 2009

Contents

List of Algorithms

List of Figures

Chapter 1

Introduction

In many real-world problems that involve multiple agents, for instance elections, there is a need for fair decision making protocols in which different agents have different amounts of influence in the outcome of a decision. *Weighted voting games* are often used in these decision making protocols. In a weighted voting game, some quota is given, and each agent (or also: player) in the game has a certain weight. If the total weight of a coalition of agents exceeds the quota, then that coalition is said to be *winning*, and *losing* otherwise.

Weighted voting games arise in various settings, such as political decision making (decision making among larger, and smaller political parties), stockholder companies (where people with different numbers of shares are supposed to have a different amount of influence), and elections (e.g. in the US Presidential Election, where each state can be regarded as a player, who has a weight equal to its number of electors).

The weight that a player has in a weighted voting game turns out not to be equal to his actual influence in the outcome of the decisions that are made using the weighted voting game. Consider for example a weighted voting game in which the quota is equal to the sum of the weights of all players. In such a game, a player's influence is equal to the influence of any other player, no matter what weight he has. In order to measure a player's influence, or *a priori* power, in such weighted voting games, the notion of a *power index* arose. Computing a power index however, turns out to be a challenge in many cases, so a lot of research has been done on how to compute various power indices efficiently.

In this thesis, instead of analyzing the power of each agent in a voting game, we investigate the problem that has often been referred to as the "inverse problem". We will call this problem the *power index voting game design problem*. In the power index voting game design problem we are given a target power index for each of the agents, and we study how to design a weighted voting game for which the power of each agent is as close as possible to the given power index.

The practical motivation behind our work is obvious: It is desirable to have an algorithm that can quickly compute a fair voting protocol, given that we want each agent to have some specified amount of influence in the outcome. When new decision making bodies must be formed, or when changes occur, such an algorithm may be used to compute a voting method that is as fair as possible.

Only very little work is known which tries to solve this problem. The existing algorithms are all local search methods that do not guarantee an optimal answer. Surprisingly, no algorithm is known for generating an exact answer. Such an algorithm to solve the inverse problem exactly is the topic of this thesis: We are interested in finding the unique game for which the power index of that game is the closest possible to a certain target power index.

It seems that the most straightforward approach to solve the inverse problem would be to simply enumerate *all possible* weighted voting games of n agents, and to compute for each of these weighted voting games its power index. We can then output the game of which the power index is the closest to the given one. Unfortunately, it turns out that enumerating all weighted voting games is not so straightforward.

The problem of designing a weighted voting game that has its power index as close as possible to a given target power index, is a special case of a more general family of problems: we can in principle attempt to design a game with any particular property (instead of just power indices) that we would like it to have. Also, we do not necessarily need to restrict our attention to weighted voting games; we could also try to design any other type of cooperative game. We refer to this collection of problems as *voting game design problems*. In this work, we will also look into this generalization, and see how our approach for solving the power index weighted voting game design problem can also be used for many other variants of voting game design problems.

In the following two sections, we describe respectively what our contributions are, and how the thesis is structured.

1.1 Contributions

In this thesis, we will present the following contributions:

- We provide a general definition for voting game design problems, and show how the power index voting game design problem is indeed a specialised case within this setting.

- We present lower and upper bounds on the cardinalities of various classes of games. As it turns out, for many of these classes there is a very strong connection with certain classes of boolean functions; allowing us to borrow many bounds directly from boolean function theory.

- We investigate thoroughly the problem of transforming various representations for simple games into each other. We give an overview of known results, and we present some original contributions:

  - We restate within the framework of simple games the Hop-Skip-and-Jump algorithm of [37] (originally intended for regular set cover problems) together with its correctness proof, and its proof of polynomial runtime.
  - We prove that it is not possible to transform within polynomial time the roof-representation of a game into a ceiling-representation, and vice versa.

- We present exact algorithms for solving power index voting game design problems, first a doubly exponential one, and subsequently we show that it is possible to obtain a singly exponential algorithm for the special case of weighted voting games. This can be regarded as the main result of this thesis.

  1. At the core of these algorithms lie methods for enumerating classes of games. Therefore, it actually follows that the same approach can be used for solving exactly any type of voting game design problem.
  2. Moreover, if a voting game design problem can be stated as an optimization problem (i.e. a problem with an objective function), as is the case for the power index voting game design problem, then our approach yields an anytime algorithm.
  3. The method that we use for enumerating weighted voting games is based on a new partial order on the class of weighted voting games, that has some specific interesting properties. We think that this result is also interesting from a non-computational, mathematical point of view.

- The algorithm for solving the power index voting game design problem for the case of weighted voting games, mentioned in the previous point, is based on working with families of minimal winning coalitions. We show how it is possible to improve the speed of this algorithm by showing that it suffices to only work with a subset of these minimal winning coalitions: The roof coalitions. Using this idea, we provide various techniques to improve our algorithm. Among these improvements is an output-polynomial time algorithm for outputting the list of ceiling coalitions, given the list of roof coalitions.

- Finally, we implement the aforementioned enumeration algorithm for weighted voting games, in order to measure its performance, obtain some interesting data about the class of weighted voting games, and validate some theoretical results related to weighted voting games.

1.2 Outline of this Thesis

The thesis is divided into seven chapters, and an appendix. The introduction chapter, Chapter 1, is the current chapter that the reader is probably reading right now.

In Chapter 2, we give a short introduction to cooperative games, with an emphasis on simple games, since our thesis deals exclusively with simple games. We will also explain extensively the concept of a power index, because a great part of what motivates the results of this thesis has to do with a problem related to power indices. We define two of the most popular power indices, i.e. the Banzhaf index and the Shapley-Shubik index, and we briefly discuss algorithms for computing them, and the computational complexity of this problem.

We define in Chapter 3 the main problem of interest that we attempt to solve in this thesis: the problem where we are given a target power index, and where we must find a game such that its power index is as close as possible to the given target power index. We explain that this specific problem is part of a more general family of problems, which we coin voting game design problems. This chapter closes with an overview on the existing literature on voting game design problems.

Before directly trying to solve the problem introduced in Chapter 3, we first discuss in Chapter 4 the problem of transforming various representations of games into each other. We give polynomial time algorithms for some of these problems, and also in some cases impossibility results regarding the existence of polynomial time algorithms. Also, in this chapter, we make some statements about the cardinality of certain classes of simple games.

Some of the results given in Chapter 4 are necessary for Chapter 5. In Chapter 5, we devise exact algorithms for the power index voting game design problem (our main problem). We first explain a naive approach for the class of monotone games, and then improve on this exponentially for the case of weighted voting games. We show how this improvement is possible by the existence of a certain partial order with some specific desirable properties. Next, we give various improvements to this algorithm by making use of the concept of roof and ceiling coalitions.

After that, in Chapter 6 we will show various experimental results of a simple implementation of this exact algorithm. Because we can also use these algorithms as enumeration algorithms, we are able to provide some exact information about voting games, such as how many weighted voting games with a fixed number of minimal winning coalitions exist.

We conclude this thesis in 7, with a discussion and some ideas for future work.

Lastly, the appendix provides some background information on order theory and computer science, which is necessary in order to understand some of our results.

Chapter 2

Preliminaries

In this chapter, we will discuss some preliminary knowledge that is required in order to understand the results. Moreover, in Appendix A, we give some additional preliminary information on order theory and computer science that is needed to understand the results in this thesis.

Sections 2.1 and 2.2 comprise an introduction to respectively cooperative game theory and power indices. Not all of the results and definitions in this chapter are necessary to understand the rest of this thesis: some of the material is included simply to give the reader a better understanding of the field that this thesis is about.

2.1 Cooperative Game Theory

In this section we give an introduction to *cooperative game theory*, or *coalitional game theory*. We will give special attention to *simple games*. The class of simple games and its subclasses that we will introduce, will be the central point of attention in this thesis. Unless otherwise noted, all of the information in this section can be looked up in an introductory text on cooperative game theory, for example [38] or in Taylor and Zwicker's book about simple games [50].

2.1.1 Cooperative Games in General

Definition 1 (Coalitional game)**.** A *coalitional game* is a pair (A, v), where $A = \{a_1, \ldots, a_n\}$ is a set of *players* and $v : 2^A \to \mathbb{R}^+ \cup \{0\}$ is a function mapping subsets of players to non-negative real numbers, describing how much collective payoff each coalition of players can gain. In this context, subsets of A are referred to as *coalitions*.

We will often use simply the word *game* to refer to a coalitional game.

Definition 2 (Grand coalition & characteristic function)**.** For a coalitional game (A, v), A is called the *grand coalition*. v is called the *characteristic function* or *gain function*.

Often, a coalitional game is denoted by just the characteristic function v if it is clear what the set of players is.

There exist two important subclasses of games: *monotone games* and *superadditive games*. These are games that satisfy the following two constraints respectively:

Definition 3 (Monotonicity)**.** A game (A, v) is *monotone* if and only if $\forall(S,T) \in (2^A)^2 : S \subset T \to v(S) \leq v(T)$.

We would like to note that in a lot of literature, monotonicity is already assumed in the definition of coalitional games itself. In this thesis, we do not do this, and we define the class of monotone coalitional games to be a subclass of the class of coalitional games.

Definition 4 (Superadditivity)**.** A game (A, v) is *superadditive* if and only if $\forall(S,T) \in (2^A)^2 : S \cap T = \varnothing \to v(S \cup T) \geq v(S) + v(T)$.

Obviously, if a game is superadditive, it is also monotone. So the superadditive games form a subclass of the monotone games.

We have now introduced three different classes of games, and we will later on introduce some more classes. Throughout this thesis, we will be working with these classes extensively. For the discussion, it is therefore convenient to introduce the following notation in order to denote classes of games that are restricted to a fixed numbers of players n:

Definition 5. Let $\mathcal{G}$ be a class of games. Then we use $\mathcal{G}(n)$ to denote the class of games restricted to the set of players $\{1, \ldots, n\}$. In more formal language:

$$\mathcal{G}(n) = \{G \mid G \in \mathcal{G} \land G = (\{1, \ldots, n\}, v)\}.$$

So throughout this thesis, we will also often use the symbol n to refer to the number of players in a game.

2.1.2 Simple Games

Simple games are the games that we will focus on in the remaining chapters. Informally, this class consists of the games where a coalition is either winning or losing, and there is nothing in between.

Definition 6 (Simple coalitional game)**.** A *simple coalitional game* or simply *simple game* is a coalitional game (N, v), where the target set of v is restricted to $\{0, 1\}$. If for a subset S of N, $v(S) = 1$, then S is called a *winning coalition*. If $v(S) = 0$ then S is called a *losing coalition*.

Just as with general coalitional games, we can define the class of *monotone simple games* and *superadditive simple games* in the obvious way.

We will now proceed to define some useful properties of simple games.

Definition 7 (Blocking coalition, proper game, strong game & decisive game)**.** In a simple game (N, v), a coalition $S \subset N$ is a *blocking coalition* if and only if $v(N \backslash S) = 0$. In a *proper simple game*, all winning coalitions are blocking coalitions. In a *strong simple game*, all blocking coalitions are winning coalitions. A *decisive simple game* (or *zero-sum simple game*) is a simple game that is both strong and proper.

Decisive simple games are important for "yes/no"-voting settings, where it must be guaranteed that a decision between two alternatives is made If the coalition of yes-voters is not winning, then the coalition of no-voters should be winning, and vice versa.

Some of the definitions in the remainder of this section are taken or adapted from [2] and [50]. Next, we turn to some syntactic definitions of certain classes of simple games. There are various important ways to represent simple games:

Definition 8 (Representations of simple games)**.** Consider the following six ways to represent coalitional games:

Winning coalition form (N, W) is a *winning coalition form* of a simple game (N, v) if and only if W is the set of all coalitions such that $\forall S \subseteq N : v(S) = 1 \leftrightarrow S \in W$.

Losing coalition form (N, L) is a *losing coalition form* of a simple game (N, v) if and only if L is the set of all coalitions such that $\forall S \subseteq N : v(S) = 0 \leftrightarrow S \in L$.

Minimal winning coalition form $(N, W_{\min})$ is a *minimal winning coalition form* of a simple game (N, v) if and only if $W_{\min}$ is the set of *minimal winning coalitions* of the game (N, v), which means that $\forall S \subseteq N : v(S) = 1 \rightarrow (\exists T \in W_{\min} : S \supseteq T)$. This implies that a simple game has a minimal winning coalition form if and only if it is monotone.

Maximal losing coalition form $(N, L_{\max})$ is a *maximal losing coalition form* of a simple game (N, v) if and only if $L_{\max}$ is the set of *maximal losing coalitions* of the game (N, v), which means that $\forall S \subseteq N : v(S) = 0 \rightarrow (\exists T \in L_{\max} : S \subseteq T)$. This implies that a simple game can be represented in maximal losing coalition form if and only if it is monotone.

Weighted form (W, q), where $W = (w_1, \ldots w_n) \in \mathbb{R^+}^n$ and $q \in \mathbb{R}^+$, is a *weighted form* of a simple game $(N = \{1, \ldots n\}, v)$ if $\forall S \subseteq N : v(S) = 1 \leftrightarrow \sum_{i \in S} w_i \geq q$. q is called the *quota* and w_i is called the *weight of player i*. Simple games that have a weighted form are called *weighted voting games*. It is obvious that every weighted voting game is monotone. However, not every monotone game is a weighted voting game. It can also be easily seen that a weighted voting game (W, q) is proper if and only if $q > \frac{\sum_{w \in W} w}{2}$.

Multiple weighted voting game An *m-multiple weighted voting game* form or *weighted m-majority game* form for a simple game (N, v) is a set

$$\{(W_1 = (w_1^1, \ldots w_n^1), q_1), \ldots, (W_m = (w_1^m, \ldots w_n^m), q_m)\}$$

of m weighted forms such that it holds that $\forall S \subseteq N : v(S) = 1 \leftrightarrow \forall j : \sum_{i \in N} w_i^j \geq q_j$. In words, a coalition must win every weighted voting game that constitutes the multiple weighted voting game.

A weighted voting game is an important type of simple game, because it has a very compact representation. Also, weighted voting games are important because they are used in a lot of practical situations, i.e., in a lot of real-life decision making protocols. For example: elections, political decision making bodies and decision making bodies in stockholder companies.

There is an important theorem regarding m-multiple weighted voting games.

Theorem 9. *Every monotone simple game is representable as an m-multiple weighted voting game.*

The proof is given in the book of Taylor and Zwicker [50]. This proof is constructive. In the construction, m gets very large, while it is often possible to use a much smaller value for m.

We will be using the following notational abuse in the remainder of this chapter, and subsequent chapters: Whenever we are discussing a weighted voting game G with players $N = \{1, \dots, n\}$ and weighted form $((w_1, \dots, w_n), q)$, then for any subset S of N we use $w(S)$ to denote $\sum_{i \in S} w_i$.

Now we will answer the question of which monotone simple games are representable as a weighted voting game. For this we will have to introduce the following two notions.

Definition 10 (Trade robustness & swap robustness)**.** A simple coalitional game (N, v) is *swap robust* if for any two winning coalitions in that game, say S and T, if we swap one player in S with one player in T, then the resulting two coalitions are not both losing.

A simple game (N, v) is *trade robust* if for any multiset $\mathcal{S}$ of winning coalitions in that game, any redistribution of players among the coalitions in $\mathcal{S}$ never results in all coalitions becoming losing.

Clearly, trade robustness implies swap robustness. In [50], the following characterization theorem is proven:

Theorem 11. *A simple coalitional game is a weighted voting game if and only if it is trade robust.*

Another important concept, related to swap robustness and trade robustness is the *desirability relation* (see [50] and [13]). We define the following desirability relation over the players in a simple game.

Definition 12 (Desirability relation)**.** In a simple game $(N = \{1, \dots, |N|\}, v)$, $\succeq_D$ is the *desirability relation* defined by:

- For any $(i, j) \in N^2$: if $\forall S \subseteq N \backslash \{i, j\} : v(S \cup \{i\}) \geq v(S \cup \{j\})$, then $i \succeq_D j$. In this case we say that i is *more desirable* than j.
- For any $(i, j) \in N^2$: if $\forall S \subseteq N \backslash \{i, j\} : v(S \cup \{i\}) = v(S \cup \{j\})$, then $i \sim_D j$. In this case we say that i and j are *equally desirable*.
- For any $(i, j) \in N^2$: if $\forall S \subseteq N \backslash \{i, j\} : v(S \cup \{i\}) \leq v(S \cup \{j\})$, then $i \preceq_D j$. In this case we say that i is *less desirable* than j.
- For any $(i, j) \in N^2$: if $i \succeq_D j$ and not $i \sim_D j$, then $i \succ_D j$. In this case we say that i is *strictly more desirable* than j.

- For any $(i, j) \in N^2$: if $i \preceq_D j$ and not $i \sim_D j$, then $i \prec_D j$. In this case we say that i is *strictly less desirable* than j.

Moreover, if neither $i \succeq_D j$ nor $j \succeq_D i$ holds for some $i, j \in N$, then we say that i and j are *incomparable*.

There exist other variants of the desirability relation, for which different properties hold [13]. In the context of other desirability relations, the desirability relation that we have defined here is refered to as the *individual desirability relation*. Since it is the only desirability relation that we will use in this thesis, we will refer to it as simply the *desirability relation*.

Using the notion of this desirability relation, it is now possible to define the class of *linear games*.

Definition 13 (Linear game)**.** A simple game (N, v) is *linear* if and only if in (N, v) no pair of players in N is incomparable with respect to $\succeq_D$. This is equivalent to saying that in (N, v) the desirability relation over N is complete.

A remarkable result is the following, proved by Taylor and Zwicker in [50]:

Theorem 14. *A game is linear if and only if it is swap robust.*

And hence we immediately obtain the following corollary by the fact that trade robustness implies swap robustness.

Corollary 15. *All weighted voting games are linear games.*

This corollary is also easily seen by noting that in a weighted voting game $G = (N = \{1, \ldots, n\}, v)$ with weighted form $((w_1, \ldots, w_n), q)$, the desirability order is always complete: for any $(i, j) \in N^2$, if $w_i \geq w_j$ it must be that $i \succeq_D j$.

This brings us to the definition of two special classes of games that will be convenient for use in subsequent chapters of this thesis.

Definition 16 (Canonical weighted voting games & canonical linear games)**.** A linear game $G = (N, v)$ is a *canonical linear game* whenever $N = \{1, \ldots, n\}$ and the desirability relation $\succeq_D$ satisfies $1 \succeq_D \cdots \succeq_D n$. When G is also weighted, then G is a *canonical weighted voting game*.

Note that the weight vector of a weighted form of a canonical weighted voting game is always non-increasing.

It is now time to introduce two special ways of representing canonical linear games.

Definition 17 (Left-shift & right-shift)**.** Let N be the set of players $\{1, \ldots, n\}$ and let S be any subset of N. A coalition $S' \subseteq N$ is a *direct left-shift* of S whenever there exists an i with $1 \leq i \leq n$ such that $S' = S \backslash \{i\} \cup \{i - 1\}$. A coalition $S' \subseteq N$ is a *left-shift* of S whenever there is for some $k > 1$ a sequence $(S_1, \ldots, S_k) \in (2^n)^k$ such that

- $S_1 = S$
- $S_k = S'$

- for all i with $1 \leq i < k$, we have that S_{i+1} is a direct-left shift of S_i.

The definitions of *direct right-shift* and *right-shift* are analogous.

Because of the specific desirability order that holds in canonical linear games, a left shift of a winning coalition is always winning, and a right-shift of a losing coalition is always losing. This allows us to represent a linear game in one of the following two forms.

Definition 18 (Roof coalition & Roof form)**.** Let $G = (\{1, \ldots, n\}, v)$ be a canonical linear game. Also, let $W_{\min}$ be G's list of minimal winning coalitions. A minimal winning coalition $S \in W_{\min}$ is a *roof coalition* whenever every right-shift of S is losing. Let W_{roof} denote the set of all roof coalitions of G. (N, W_{roof}) is called the *roof form* of G.

Definition 19 (Ceiling coalition & Ceiling form)**.** Let $G = (\{1, \ldots, n\}, v)$ be a canonical linear game. Also, let $L_{\max}$ be G's list of maximal losing coalitions. A maximal losing coalition $S \in W_{\min}$ is a *ceiling coalition* whenever every left-shift of S is winning. Let W_{ceil} denote the set of all ceiling coalitions of G. (N, W_{ceil}) is called the *ceiling form* of G.

The terminology (“roof” and “ceiling”) is taken from [37], although game theorists have also been calling it *shift-minimal winning coalitions* and *shift-minimal losing coalitions* [50].

In this thesis we will be using the following symbols to denote specific classes of games:

- $\mathcal{G}_{\mathsf{sim}}$ denotes the general class of simple games;
- $\mathcal{G}_{\mathsf{mon}}$ denotes the class of monotone simple games;
- $\mathcal{G}_{\mathsf{lin}}$ denotes the class of linear games;
- $\mathcal{G}_{\mathsf{clin}}$ denotes the class of canonical linear games;
- $\mathcal{G}_{\mathsf{wvg}}$ denotes the class of weighted voting games;
- $\mathcal{G}_{\mathsf{cwvg}}$ denotes the class of canonical weighted voting games.

2.1.3 Representation Languages

Because we will be discussing simple games from a computational perspective, we would like to introduce the concept of *representation languages* for simple games. Note that this is a novel idea that we use for the sake of convenience in this thesis, and is not used in any other literature.

We have introduced several ways of representing simple games:

- by the set of winning coalitions,
- by the set of losing coalitions,
- by the set of minimal winning coalitions,
- by the set of maximal losing coalitions,

- by the set of roof coalitions,
- by the set of ceiling coalitions,
- by a weighted representation,

The idea is to turn these methods representing simple games into languages: Sets of strings, such that the strings are a description of a game according to one of the methods in the list above. This makes it easier to talk about the various ways of representating simple games.

Prior to introducing the languages together with their definitions, we need a way of describing coalitions. Coalitions can be described using their *characteristic vector*.

Definition 20. Let $N = \{1, \ldots, n\}$ be a set of n players. The *characteristic vector* $\vec{\chi}(S)$ of a coalition $S \subseteq N$ is the vector $(\chi(1, S), \ldots, \chi(n, S))$ where

$$\chi(i, S) = \begin{cases} 1 \text{ if } i \in S \\ 0 \text{ otherwise.} \end{cases}$$

Definition 21 (Representation Languages)**.** We define the following langages to represent simple games.

- $\mathcal{L}_W$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of coalitions that are winning in G.
- $\mathcal{L}_{W,\min}$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of minimal winning coalitions in G.
- $\mathcal{L}_L$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of coalitions that are losing in G.
- $\mathcal{L}_{L,\max}$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of maximal losing coalitions in G.
- $\mathcal{L}_{\mathsf{roof}}$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of roof coalitions in G.
- $\mathcal{L}_{\mathsf{ceil}}$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. ℓ represents a simple game G if and only if the set of coalitions that ℓ describes is precisely the set of ceiling coalitions in G.
- $\mathcal{L}_{\mathsf{weights}}$. Strings $\ell \in L_W$ are lists of numbers $\langle w_1, \ldots, w_n, q\rangle$. ℓ represents a simple game G if and only if G has weighted representation $((w_1, \ldots, w_n), q)$.

We will use the following convention: With $\mathcal{L}(n)$ we mean the set of strings in the language $\mathcal{L}$ restricted to n players. Also, let ℓ be a string from a representation language $\mathcal{L}(n)$. Then we write G_ℓ to denote the simple game on players $\{1, \dots, n\}$ that is represented by ℓ.

Definition 22. We say that a class of games $\mathcal{G}$ *is defined by* a language $\mathcal{L}$ if and only if $\forall \ell \in \mathcal{L} : \exists G \in \mathcal{G} : G_\ell = G$ and vice versa $\forall G \in \mathcal{G} : \exists \ell \in \mathcal{L} : G_\ell = G$.

Using the above definition, we see that

- $\mathcal{G}_{\mathsf{sim}}$ is defined by both $\mathcal{L}_W$ and $\mathcal{L}_L$;
- $\mathcal{G}_{\mathsf{mon}}$ is defined by both $\mathcal{L}_{W,\min}$ and $\mathcal{L}_{L,\max}$;
- $\mathcal{G}_{\mathsf{lin}}$ is defined by both $\mathcal{L}_{\mathsf{roof}}$ and $\mathcal{L}_{\mathsf{ceil}}$;
- $\mathcal{G}_{\mathsf{wvg}}$ is defined by $\mathcal{L}_{\mathsf{weights}}$.

2.1.4 Solution Concepts

Now we will briefly return to the general class of coalitional games that are not necessarily simple.

Solution concepts are a central topic in coalitional game theory.

Definition 23 (Solution concepts in cooperative games)**.** Let $v \in \mathcal{G}(N)$. A solution concept is a value $x \in \mathbb{R}^{|N|}$, representing a payoff to each player.

If the assumption is that eventually the grand coalition will form in a game, then we need to split up the gains from the grand coalition among the set of all players. This is a standard assumption in cooperative game theory: games are assumed to be monotone, and thus the highest gain is obtained by the grand coalition, i.e. the case that everyone cooperates. Solution concepts are usually defined from this point of view, so if x is a solution concept for a game $v \in G(N)$, then the sum of all items in this vector equals $v(N)$.

This assumption is not that restrictive though: one can also see a solution concept as a way to divide the gains of *any* coalition among its members. If we want a method to divide the gain of a smaller coalition among its members, then we can just apply the solution concept on the subgame induced on the players in the smaller coalition.

Below, we will describe the Shapley value solution concept. We give special attention to the Shapley value, since it is relevant for our discussion about power indices. However, there are also other popular solution concepts, such as the core [19], the nucleolus [42], and the kernel [8].

The Shapley Value

One of the most well-known solution concepts is the Shapley value. This solution concept was introduced by Lloyd S. Shapley in [43].

Definition 24. Given a coalitional game $(A = (a_1, \ldots, a_n), v)$, the Shapley value φ for this game is the payoff vector $(\varphi_1, \ldots, \varphi_n)$. For $1 \leq i \leq n, \varphi_i$ is defined as follows. Let Π be the set of all permutations of A. Let $\text{Prec}_i : \Pi \to 2^A$ be the function that, given a $\pi \in \Pi$, returns the set of players occurring before a_i in π. Then

$$\varphi_i = \frac{\varphi'_i}{n!},$$

and

$$\varphi'_i = \sum_{\pi \in \Pi} v(\text{Prec}_i(\pi) \cup \{i\}) - v(\text{Prec}_i(\pi)).$$

Remark 25. Whenever we are discussing solution concepts of two or more games simultaneously, we will distinguish them for these games by parametrizing them with the specific game. e.g. for two simple games G' and G, we use the notation $\varphi_i(G)$ and $\varphi_i(G')$ to denote the Shapley values for player i in game G and game G' respectively. Also we sometimes denote a game $G = (N, v)$ simply by its characteristic function v, so then we write $\varphi_i(v)$.

In cooperative game theory research, a common practice is to try to axiomatize a solution concept.

Definition 26 (Axiomatic characterization)**.** An *axiomatization* or *axiomatic characterization* of a solution concept c for a class of games $\mathcal{G}$ is a set of axioms defined using games in $\mathcal{G}$ such that c is the unique solution concept that satisfies all the axioms.

Axiomatic characterizations are useful for assessing the reasonability of solution concepts. For the Shapley value, a lot of axiomatic characterizations have been devised. We now give a clean one in Definition 27 and Theorem 28, devised by Shapley himself, given for example in [11]:

Definition 27 (Game permutation & game addition)**.** Let v be a coalitional game on the players N; let $\pi : N \to N$ be a permutation of N. Then the game (πv) is defined as

$$\forall S \subseteq N : (\pi v)(S) = v(\{\pi(i) \mid i \in S\}).$$

Let v_1 and v_2 be two games on the players N; then the game $(v_1 + v_2)$ is defined as

$$\forall S \subseteq N : (v_1 + v_2)(S) = v_1(S) + v_2(S).$$

Theorem 28. *Let $\mathcal{G}(N)$ be the class of general coalitional games on any set of players $N = \{1, \ldots, |N|\}$. For any $v, v_1, v_2 \in \mathcal{G}(N)$, there is a solution concept $c = (c_1, \ldots, c_{|N|})$ that is the unique solution concept satisfying*

A1 (carrier): *If any $S \subseteq N$ satisfies $\forall T \subseteq N : v(T) = v(T \cap S)$, then $\sum_{i \in S} c_i(v) = v(S)$;*

A2 (permutation): *For any permutation π: $c_i(v) = c_{\pi(i)}(\pi v)$;*

A3 (addition): $c_i(v_1 + v_2) = c_i(v_1) + c_i(v_2)$.

Moreover, c is the Shapley value φ.

2.2 Power Indices

We will discuss power indices in this section. Power indices are measures for the amount of influence that a player has in a simple game. We start in first with a discussion of why we need power indices, in Section 2.2.1. After that, we will introduce three power indices in sections 2.2.2, 2.2.3 and 2.2.4. Finally, in Section 2.2.5 we briefly talk about how to compute these power indices.

2.2.1 Motivation Behind Power Indices

Power indices originally were introduced because it was observed that in weighted voting games, the weight of a player is not directly proportional to the influence he has in the weighted voting game. This is actually quite easy to see through the following trivial example weighted voting game:

$$\left(W, q = \sum_{w \in W} w \right).$$

Here, each agent is in only one winning coalition: the grand coalition. So no matter what the weights of the agents are, they all have the same power.

Now that we know that the weight of a player is not a good measure for the player's power, the question will be: what *is* a good measure for the player's power in a weighted majority game?

Various answers have been given to this question, by various researchers in the area of coalitional game theory. These answers are in the form of *power indices*, which are simply mathematical formulations for values that try to describe the 'true' influence that a player has in a weighted voting game.

Almost all power indices make no assumptions on the true preference of a player in a voting game. So we say power indices measure a player's *a priori* power in a weighted voting game. That is, we attempt to objectively measure the influence that a player has in the outcome of a weighted voting game, without having any statistical information on which votes are likely to be cast by players, or groups of players. To do this, we cannot avoid making certain assumptions, but we let these assumptions be as neutral as possible. For example, in the Banzhaf index that we describe below, it is assumed that each coalition will form with equal probability.

While the need for power indices originally arose from weighted voting games, all of the power indices that have been devised up till now are also suitable for any other kind of simple coalitional game. So, for any simple coalitional game, we can use a power index as a measure of a player's a priori power in it.

In computer science, the focus of power index research naturally lies most of the times in finding algorithms that efficiently compute a power index. However, of course power indices have not only been investigated in computer science: even more research has been done in the field of game theory. Purely game-theoretical research has mostly focused on mathematical analysis of the indices: finding mathematical properties and finding *axiomatic characterizations* of the power indices, just as has been done for the solution concepts of the previous chapter. Analogous to axiomatic characterizations of solution concepts, an axiomatic characterization of a power index p is a set of axioms such that the only formula satisfying all axioms can only be p. Axiomatizations are helpful for discussions on topics like the reasonability of a certain power index in a certain situation. And although axiomatizations do give us some mathematical properties of the power indices, those properties are obvious and follow most of the times immediately from the definition. It turns out that they do not really help us for answering most of the computation-related questions that we are interested in, so here we will not state explicitly any axiomatic characterizations for the power indices.

One quick last note before we move on to the actual power indices. Originally, a power index p was defined to be a vector $(p_1, p_2, \ldots, p_n)$ where $p_i, 1 \leq i \leq n$ is supposed to be a measure of the power for player i (at least, this is the way most of the classic game theory papers define a power index). Later on, in more and more papers this definition was altered a bit, and researchers began to speak of "the power index p_i", i.e. all of the p_i's themselves were defined to be power indices of individual players. In this document, we use both definitions, depending on which definition is convenient. We believe that no confusion will arise from this.

2.2.2 The Shapley-Shubik Index

The very first power index that has been proposed is the Shapley-Shubik index. This is nothing more than the Shapley value of [43] that we introduced in the previous chapter, but now restricted to simple coalitional games. So:

Definition 29 (Shapley-Shubik index & raw Shapley-Shubik index)**.** For a simple coalitional game $(A = \{a_1, \ldots, a_n\}, v)$, let Π be the set of all permutations of all players. For all $1 \leq i \leq n$, let $\text{Prec}_i : \Pi \rightarrow 2^A$ be the function such that $\text{Prec}_i(\pi \in \Pi)$ returns the set of players that occur before player a_i in π. Then, the *Shapley-Shubik index* is $\varphi = (\varphi_1, \ldots, \varphi_n)$, where for $1 \leq i \leq n$:

$$\varphi_i = \frac{\varphi_i'}{n!},$$

and

$$\varphi_i' = \sum_{\pi \in \Pi} v(\text{Prec}_i(\pi) \cup \{i\}) - v(\text{Prec}_i(\pi)).$$

φ_i' is called the *raw Shapley-Shubik index*.

We can interpret the raw Shapley-Shubik index for i as the number of different orders of arrival in which the players can join the coalition, such that the arrival of player i changes a losing coalition into a winning coalition.

Definition 30 (Pivot permutations & pivot players for permutations)**.** Let (A, v) be a simple game and let $\pi \in \Pi$ be a permutation of the players in A such that $v(\text{Prec}_i(\pi) \cup \{i\}) - v(\text{Prec}_i(\pi)) = 1$. We call π a *pivot permutation* for player i. Also, we say that in that case, i is a *pivot player* for π.

The Shapley-Shubik index of player i is the raw Shapley-Shubik index of i divided by $n!$. $n!$ is the total number of possible permutations of all n players, and because in each permutation, exactly one player is a pivot player (if we assume monotonicity), we have

$$\sum_{i=1}^{n} \varphi_i = 1,$$

which is a nice property. As a corollary, the Shapley-Shubik index of a player is always a number between 0 and 1. It is easy to see that the Shapley-Shubik index of a player i can be interpreted as the probability that i will be the player that changes a losing coalition into a winning coalition, if all players would join the coalition in a random order.

There is an alternative well-known definition of the raw Shapley-Shubik index (for a player i) that is sometimes convenient to use:

$$\varphi'_i = \sum_{S \subseteq A \backslash \{i\}} (|S|!(|A| - |S| - 1)!(v(S \cup \{i\}) - v(S))) \tag{2.1}$$

It is not hard to see why this definition is correct: for any $S \subseteq A$ for which it holds that $(v(S \cup \{i\}) - v(S)) = 1$, there are $|S|!$ ways to permute the players in S, and there are $(|A| - |S| - 1)!$ ways to permute the $|A| - |S| - 1$ players outside $S \cup \{i\}$.

The reader may wonder why this power index is called the Shapley-Shubik index if the Shapley-Shubik index is nothing more than the Shapley value, restricted to simple coalitional games. The reason for this is that while Shapley invented the Shapley value, it was the paper [44] written by Shapley and Shubik, where it was pointed out that the Shapley value is an excellent way to measure someone's voting power in a weighted voting game. As we explained, originally the Shapley value was meant as a solution concept for general coalitional games, i.e. to allocate fairly the gain of the grand coalition among the set of players. In this new setting, the Shapley value is used as a way to measure power, so this is a different purpose than the original purpose of the Shapley value. Hence, the Shapley-Shubik index was born.

The axiomatic characterization for the Shapley value (that we gave in the previous chapter) holds if we restrict it to simple games, but not if we restrict it to simple superadditive games or simple monotone games. In general it is not true that an axiomatic characterization for a certain value on a certain set G of games also holds for a subset $G' \subset G$: while it is always true that in this case the value satisfies the axioms, it needs no longer be so that the value is the *unique* value that satisfies the axioms. Shubik states in [11] an axiomatization for the Shapley-Shubik value that holds in simple superadditive and simple monotone games.

2.2.3 The Banzhaf Index

In 1965, two decades after the Shapley-Shubik index, the *normalized Banzhaf power index* was proposed, named after its inventor John F. Banzhaf III [20].

Definition 31 (normalized Banzhaf index & raw Banzhaf index)**.** For a simple coalitional game $(A = (a_1, \ldots, a_n), v)$, the *normalized Banzhaf index* is defined as $\beta = (\beta_1, \ldots, \beta_n)$, where for $1 \leq i \leq n$:

$$\beta_i = \frac{\beta'_i}{\sum_{j=1}^{n} \beta'_j},$$

and

$$\beta'_i = |\{S \subseteq A \backslash \{i\} \mid v(S) = 0 \wedge v(S \cup \{i\}) = 1\}|. \tag{2.2}$$

Here, β'_i is called the *raw Banzhaf index* for player i.

So the raw Banzhaf index is simply the number of coalitions where i is critical for the coalition to win the game.

Definition 32 (Swing coalition & critical player)**.** Let $S \subseteq W$ be a coalition such that $v(S) = 0 \wedge v(S \cup \{i\}) = 1$. We will call S a *swing coalition* for i. In a swing coalition for i, i is called a *critical player* or *swing player* (because he "swings" the outcome for the coalition from losing to winning, if he joins).

Note that the definition we use here differs from the definition in some other literature. Originally, for a simple game (N, v) a swing is defined as a pair (S, S') for which $S, S' \subseteq N$ and $S' \subset S$ with $|S'| = |S| - 1$. However, we think that for our discussion, Definition 32 is more convenient.

From the definition we can see that the Banzhaf index for player i is simply the raw Banzhaf index divided by the sum of all raw Banzhaf indices. This way, β_i is always a number between 0 and 1, and we have the elegant mathematical property that $\sum_{i=1}^{n} \beta_i = 1$.

Coleman reinvented the index in 1971 [6], only after which the index became popular. For this reason, the normalized Banzhaf index form is also known as the *Coleman index* or the *Banzhaf-Coleman index*. Coleman has some other power indices though, see the next section for that. Moreover, the normalized Banzhaf index is also known under the name *standardized Banzhaf index*.

Later, in 1979, the index was revised by Dubey and Shapley. It was noted that because of the denominator ($\sum_{j=1}^{n} \beta'_j$), some useful information was lost. Therefore, Dubey and Shapley proposed in [12] a modification of the Banzhaf index where the denominator is replaced by 2^{n-1}. However, this new version of the Banzhaf index is no longer normalized.

Definition 33 (Absolute Banzhaf index)**.** So this modified Banzhaf index β''_i is defined as

$$\beta''_i = \frac{\beta'_i}{2^{n-1}}.$$

The authors called this index the *absolute Banzhaf index*.

β''_i is precisely the fraction of coalitions containing i, for which i is a critical player. So if each coalition were to form with equal probability, then β''_i is the probability that i is a critical player for the coalition, given that player i is in the coalition.

In [12], an axiomatic characterization of the raw Banzhaf index is given for the general class of simple games.

Actually, as it turned out, the absolute Banzhaf index was invented much earlier, even before the normalized Banzhaf index was proposed. In 1946, the index was described by

Penrose in [39]. So in the literature, the absolute Banzhaf index is sometimes also referred to as the *Penrose index* or the *Banzhaf-Penrose index*.

Compared to the absolute Banzhaf index, the normalized Banzhaf index not only has some undesirable mathematical properties: from a computational point of view, the normalized Banzhaf index is also harder to compute, since (because of the denominator) we need to first calculate the raw Banzhaf indices of all players, before we can calculate the normalized Banzhaf index of one player. This is not the case for the absolute Banzhaf value, where the denominator 2^{n-1} is easy to compute.

Definition 34 (Banzhaf index)**.** In the literature, usually when one talks about the *Banzhaf index*, one means the absolute Banzhaf index. We will also use that convention in this survey: when we use the term *Banzhaf index*, we will mean the *absolute Banzhaf index*. Moreover, in the remainder of this text, we will not use the symbol β_i' to denote the absolute Banzhaf index of player i, but instead we use β_i, and we will use β to denote the vector of absolute Banzhaf indices of all players.

Remark 35. If we compare the second definition of the raw Shapley-Shubik index, Equation 2.1, with Equation 2.2 of the raw Banzhaf index, we see a remarkable resemblance between the two: in the raw Shapley-Shubik index, for each coalition $S \subseteq A\backslash\{i\}$ for which $v(S) = 0 \wedge v(S \cup \{i\}) = 1$, we add $|S|!(|A| - |S| - 1)!$ to the index, while in the raw Banzhaf index, for the same coalitions we simply add 1 to the index. This is the only difference between those two indices.

2.2.4 The Deegan-Packel Index

There are a lot of power indices other than the Banzhaf and Shapley-Shubik indices, although they are not used as often. We will explain in this section one such index that is perhaps the most popular index among these lesser known indices: the *Deegan-Packel* index, proposed by Deegan and Packel in [21]. In this index we consider the set of minimal winning coalitions $W_{\min}$ of a simple game. Informally, for each minimal winning coalition $S \in W_{\min}$, the Deegan-Packel index is obtained by adding $1/|S|$ to the Deegan-Packel index of each member in S.

The Deegan-Packel index was proposed for monotone simple coalitional games, but can also be used for non-monotone simple coalitional games. However, clearly the Deegan-Packel intuitively would not make any sense for coalitional games that are not monotone.

The Deegan Packel index is justified in situations where the *size principle* holds [41]:

> "In social situations similar to n-person, zero sum games with side payments, participants create coalitions just as large as they believe will ensure winning and no larger."

The formal definition of the Deegan-Packel index is:

Definition 36 (Deegan-Packel index)**.** For a simple coalitional game $(A = (a_1, \ldots, a_n), v)$, the *Deegan-Packel index* is defined as $\rho = (\rho_1, \ldots, \rho_n)$, where for $1 \leq i \leq n$:

$$\rho_i = \frac{1}{|W_{\min}|} \sum_{S \in 2^A : a_i \in S \wedge v(S) = 1 \wedge \forall a \in S : v(S \backslash \{a\}) = 0} \frac{1}{|S|}.$$

So, if the assumption is made that players will only form minimal winning coalitions, each minimal winning coalition has an equal probability of forming, and the players in a coalition divide the gains equally, then the Deegan-Packel index of a player represents the player's expected gain.

2.2.5 Computing Power Indices

In this section, we will discuss the problem of computing the three indices introduced so far. This topic has been very widely studied and there are many algorithms available. The discussion in this section will be brief. For a survey of complexity results, exact algorithms and approximation algorithms for computing power indices, see [23].

Computational Complexity

First, we would like to point out that the computation of the Banzhaf and Shapley-Shubik indices is not very easy in general, if the game is given in a weighted representation. This is due to the following theorems:

Theorem 37 ([10, 14])**.** *Let $(W = (w_1, \ldots, w_n), q)$ be any weighted representation of a weighted voting game. Let RAWSHAPLEY* : $\Sigma^* \to \mathbb{N}$ *be the function that returns the raw Shapley-Shubik index φ_i' for game (W, q) when it is given input $\langle W, q, i \rangle$. RAWSHAPLEY is $\#$P-complete.*

Theorem 38 ([40])**.** *Let $(W = (w_1, \ldots, w_n), q)$ be any weighted voting game. Let PIVOT-WVG* : $\Sigma^* \to \mathbb{N}$ *be the function that returns the raw Banzhaf index β_i' for game (W, q) when it is given input $\langle W, q, i \rangle$. PIVOT-WVG is $\#$P-complete.*

Moreover, Aziz points out in [2] that computing the Banzhaf index remains #P-complete when the game is given as a list of minimal winning coalitions.

Algorithms

Many exact algorithms and approximation algorithms are known for computing power indices of games given in various representations. A very important case is of course when the game is given in weighted form.

We just pointed out above that the Banzhaf and Shapley-Shubik power indices can probably not be computed in polynomial time in that case. For the Deegan-Packel index, no #P-completeness results are known, but for the case that the game is given in weighted form, no polynomial time algorithm is known either. However, we can trivially obtain a polynomial time algorithm from the definition of the Deegan-Packel index in the case that the game is given as a list of minimal winning coalitions. Matsui and Matsui present in [34] an output-polynomial time enumeration algorithm that outputs the set of minimal winning coalitions in a game given in weighted form. This then yields an algorithm running in time polynomial in the number of players and the number of minimal winning coalitions, for computing the Deegan-Packel index for a weighted representation of a game.

For the Banzhaf and Shapley-Shubik indices, an important result is known for a special case of weighted representations of games: In the case that the game is given in weighted form, and all weights are integers, it is known that the problem can be solved in pseudo-polynomial time, by using a dynamic programming method based on generating functions [5, 33].

In the more general case where the game is given in weighted form, but where the weights are not required to be integers, the fastest known approach for computing the Banzhaf index or Shapley-Shubik index is by Klinz and Woeginger [25] achieving a runtime in $O^*((\sqrt{2})^n)$.

Chapter 3

Voting Game Design — The Problem Statement

In this chapter, we will introduce the problem that we call the *voting game design* problem: the problem of finding a simple game that satisfies a given requirement as well as possible. We will focus on the problem of finding games in which a power index of the game is as close as possible to a given target power index.

This chapter consists of two sections. In section 3.1, we will state the actual problem that we are concerned with in this thesis. Moreover, we will introduce a framework that allows us to discuss the problem in a convenient way. In section 3.2, we will discuss the work that has already been done on these problems.

3.1 A Framework for Voting Game Design Problems

We define a voting game design problem as an optimization problem where we are given three parameters $f, \mathcal{G}$ and $\mathcal{L}$. In such a voting game design problem we must minimize some function $f : \mathcal{G} \rightarrow \mathbb{R}^+ \cup \{0\}$, with $\mathcal{G}$ being some class of simple games. $\mathcal{L}$ is a representation language for $\mathcal{G}$. We require the game that we output to be in the language $\mathcal{L}$.

Definition 39 (($f, \mathcal{G}, \mathcal{L}$)-voting game design (($f, \mathcal{G}, \mathcal{L}$)-VGD))**.** Let $\mathcal{G}$ be a class of simple games, let $\mathcal{L}$ be a representation language for $\mathcal{G}$, and let $f : G \rightarrow \mathbb{R}^+ \cup \{0\}$ be a function. The *($f, \mathcal{G}, \mathcal{L}$)-voting game design problem* (or *($f, \mathcal{G}, \mathcal{L}$)-VGD*) is the problem of finding an $\ell \in \mathcal{L}$ such that $G_\ell \in \mathcal{G}$ and $f(G_\ell)$ is minimized.

Hence, f can be seen as a function indicating the error, or the distance from the game that we are ideally looking for. By imposing restrictions on the choice of f, and by fixing $\mathcal{G}$ and $\mathcal{L}$, we can obtain various interesting optimization problems. The cases that we will focus on will be where f is a function that returns the distance from a certain target power index.

Definition 40 (($g, \mathcal{G}, L$)-power index voting game design (($g, \mathcal{G}, \mathcal{L}$)-PVGD))**.** Suppose $\mathcal{G}$ is a class of games, and $\mathcal{L}$ is a representation language for a class of games. Furthermore, suppose $g : \mathcal{G} \rightarrow \mathbb{R}^n$ is a function that computes a certain power index (e.g. the Shapley-Shubik index or the Banzhaf index) for any game in $\mathcal{G}$. Then, the *$(g, \mathcal{G}, L)$-power index voting game design problem* (or *$(f, \mathcal{G}, \mathcal{L})$-PVGD*) is the $(f, \mathcal{G}, \mathcal{L})$-VGD problem with f restricted to those functions for which there exists a vector $(p_1, \ldots, p_n)$ such that for each $G \in \mathcal{G}$,

$$f(G) = \sqrt{\sum_{i=1}^{n}(g(i, G) - p_i)^2}.$$

In words, in a $(g, \mathcal{G})$-PVGD problem we must find a weighted voting game in the class $\mathcal{G}$ that is as close as possible to a given target power index $(p_1, \ldots, p_n)$ according to power index function g. We measure the error as the square root of the sum of squared errors, i.e. the euclidian distance in $\mathbb{R}^n$ between the power index of a game, and the target power index. In principle, we could also use alternative rules for this; for our discussion it does not really matter: We will see that the methods we use to solve these problems work well as long as the error function is not hard to compute, given g.

We can analyze this problem for various power index functions, classes of games, and representation languages. So, instances of such problems are then represented by vectors $(p_1, \ldots, p_n)$. A particularly interesting case that we will focus on are the problems $(\beta, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD and $(\varphi, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD. This is the problem of finding a weighted voting game in weighted representation, that is as close as possible to a certain target Banzhaf index or Shapley-Shubik index.

3.2 Known Work

Although some specific variants of $(f, \mathcal{G}, \mathcal{L})$-PVGD problems are mentioned sporadically in the literature, not many serious attempts to solve this problem are known to us. Often, the problem is referred to as *the inverse problem*. We chose deliberately not to use this name for the problem, since it does not suggest immediately that it is in any way connected to the theory of simple games. *The inverse problem* is a name that could in principle be used for many other problems that do not at all have to do with cooperative game theory.

We know of only a few papers where the authors propose algorithms for ($(f, \mathcal{G}, \mathcal{L})$-PVGD). One of them is by Fatima et al. for the case of $(\varphi, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD) [16], i.e., the case of finding a weighted voting game in weighted representation for the Shapley-Shubik index. This algorithm works essentially as follows: It first receives as input a target Shapley-Shubik index and a vector of initial weights. After that, the algorithm enters an infinite loop where repeatedly the Shapley-Shubik index is computed, and the weight vector is updated according to some rule. The Shapley-Shubik index is computed using a linear time randomized approximation algorithm, proposed in [15, 17] by the same authors. For updating the weights, the authors propose two different rules of which they prove that by applying the rule, the Shapley-Shubik index of each player cannot get worse. Hence, the proposed algorithm is an *anytime* algorithm: an algorithm that can be terminated anytime, but gets closer to the optimal answer the longer the algorithm runs. No analysis on the approximation error is done, although the authors mention in a footnote that analysis will be done in future work. The runtime of one iteration of the algorithm is shown to be $O(n^2)$ (where n denotes the number of players).

Another attempt is by Aziz et al. for the $(\beta, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD variant of the problem. The approach the authors present here [3] resembles that of Fatima et al. in that their algorithm repeatedly updates the weight vector in order to get get closer to the target power index. The algorithm gets as input a target Banzhaf index. As an initial step, an integer weight vector is guessed according to a normal distribution approximation. Subsequently, the algorithm enters an infinite loop, and consecutively computes the Banzhaf index and updates the weight. For computing the Banzhaf index, the *generating function method* is used [4, 33, 5]. This is an exact pseudopolynomial time method that works only when the weights in the weighted representation of a game are integers. Therefore, the output of the algorithm is always an integer weighted representation, contrary to the method of Fatima et al. for which the output is a real weighted representation. The updating is done by interpolating a best fit curve. This results in a real weight vector. To obtain integer weights, the weight vector is rounded to integers, but prior to that it is multiplied by a suitable constant that reduces the error when rounding to integers. This can be done because of the following easy theorem, which we mention here because we will use it later on in the discussion of our own approach:

Theorem 41. *Let $G \in \mathcal{G}_{\mathsf{wvg}}(N)$ be a weighted voting game with $N = \{1, \ldots, n\}$, and let $\ell = ((w_1, \ldots, w_n), q)$ be a weighted representation for G. For any $\lambda \in \mathbb{R}^+$, we have that $\ell' = ((\lambda w_1, \ldots, \lambda w_n), \lambda q)$ is a weighted representation for G.*

Proof. For any coalition $C \subseteq N$ such that $w_\ell(C) < q$:

$$w_{\ell'}(C) = \sum_{i \in C} \lambda w_i = \lambda \sum_{i \in C} w_i = \lambda w_\ell(C) < \lambda q,$$

and for any coalition $C \subseteq N$ such that $w_\ell(C) \geq q$:

$$w_{\ell'}(C) = \sum_{i \in C} \lambda w_i = \lambda \sum_{i \in C} w_i = \lambda w_\ell(C) \geq \lambda q.$$

□

For Aziz's approach, nothing is known about convergence, so it is not certain whether this method is *anytime* just like Fatima's algorithm. Also, no approximation guarantee is given. Moreover, not much is known about the practical performance of this algorithm (one example is presented of this algorithm working on a specific input). The runtime of an iteration of the algorithm is not given.

Leech proposes in [29, 31] an approach that largely resembles the method of Aziz et al.: it is the same, with the exception that a different updating rule is used. The method that Leech uses for computing the Banzhaf index is not mentioned. The focus in this paper is on the results that are obtained after applying the method to the 15-member EU council (also see [30]), and to the board of governors of the International Monetary Fund.

Because the three approaches that we just discussed do not give an analysis of various aspects of the methods they use, we consider these algorithms to be mostly heuristic methods (with possibly an exeption of the algorithm of Fatima et al., of which it is shown that the algorithm convergences).

Alon and Edelman observe that we need to know *a priori* estimates of what power indices are achievable in simple games, in order to analyze the accuracy of these kinds of iterative algorithms, i.e. there is a need for information about the distribution of power indices in the unit simplex[1]. As a first step into solving this problem, they prove in [1] a specific result for the case of the Banzhaf index for monotone games.

Also, some applied work has been done on the design of voting games. In two papers, one by Laurelle and Widgren [28] and one by Sutter [49], the distribution of voting power in the European Union is analyzed and designed using iterative methods that resemble the algorithm of Aziz [3].

[1]The definition of *unit simplex* is given in Appendix A

Chapter 4

Voting Game Synthesis

The method that we will propose for solving the power index voting game design problem involves transforming between different representations for classes of simple games. Therefore, we will in this chapter first explore in general the problem of transforming representations into each other. We call these problems *Voting Game Synthesis problems*, inspired by the term *threshold synthesis* used in [37] for finding a weight vector for a so-called *threshold function*, to be defined later in this chapter.

In Section 4.1, we will start this chapter by first finding out what we can say about the number of various types of voting games. We state the synthesis problem formally in Section 4.2. In Section 4.3 we will look at how to solve it.

4.1 On Cardinalities of Classes of Simple Games

In some variants of the Voting Game Synthesis problem, we want to transform a simple game into a specific representation language that defines only a subclass of the class of games that is defined by the input-language. It is interesting to know what fraction of a class of games is synthesizable in which language, i.e. we are interested in the cardinalities of all of these classes of simple games. Apart from the synthesis problem, this is probably an interesting question in its own right, and we will see that we also need this information in order to analyse the algorithms for the voting game design problems that we will present in the next chapters.

In Section 4.1.1, we will discuss the amounts of monotone simple games and linear games on n players. In Section 4.1.2, we will look at the number of weighted voting games on n players.

4.1.1 Monotone Simple Games and Linear Games

Let us start off with the cardinality of the class of monotone games of n players: $\mathcal{G}_{\mathsf{mon}}(n)$. This class is defined by language $\mathcal{L}_{W,\min}$, i.e. each game in this class can be described by a set of minimal winning coalitions (MWCs), and for each possible set of MWCs $W_{\min}$ there is a monotone game G such that the MWCs of G are precisely $W_{\min}$. We see therefore that the amount of monotone games on n players is equal to the amount of families of MWCs on n players. In a family of MWCs, there are no two coalitions S and S' such that S' is a superset of S. In other words: all elements in a family of MWCs are pairwise incomparable with respect to $\supseteq$, or: A family of MWCs on the set of players $N = \{1, \ldots, n\}$ is an antichain in the poset $(2^N, \supseteq)$. Hence, the amount of antichains in this poset is equal to $|\mathcal{G}_{\mathsf{mon}}(n)|$. Counting the number of antichains in this poset is a famous open problem in combinatorics, known as *Dedekind's problem* and $|\mathcal{G}_{\mathsf{mon}}(n)|$ is therefore also referred to as the *nth Dedekind number* D_n. Dedekind's problem was first stated in [9]. To the best of our knowledge, exact values for D_n are known only up to $n = 8$. We will return to the discussion of the Dedekind number in Section 5.1, where we will also mention some known upper and lower bounds for it. For now, let us simply say that D_n grows quickly in n: as n gets larger, D_n increases super-exponentially, but sub-doubly-exponentially.

For linear games, we know only of the following lower bound on the number of canonical linear games. The prove that we give here is from [37]:

Theorem 42. *For large enough n,*

$$|\mathcal{G}_{\mathsf{clin}}(n)| \geq 2^{(\sqrt{\frac{2}{3}}\pi 2^n)/(n\sqrt{n})}.$$

Proof. $|\mathcal{G}_{\mathsf{clin}}(n)|$ is equal to the number of antichains in the poset $(2^N, \preceq_{\mathsf{ssrs}})$, where for two coalitions $S \subseteq N$ and $S' \subseteq N$, we have $S \preceq_{\mathsf{ssls}} S'$ if S is a superset of a left-shift of S'. $(2^N, \preceq_{\mathsf{ssrs}})$ is a ranked poset: consider the following rank function r

$$\begin{aligned} r \;\; : \;\; & 2^N \to \mathbb{N} \\ & S \mapsto \sum_{i \in S} n - i + 1. \end{aligned}$$

It can be seen that in $(2^N, \preceq_{\mathsf{ssrs}})$, a coalition S covers another coalition S' if $r(S) = r(S') - 1$, therefore this is a valid rank function. A set of points of the same rank is an antichain in $(2^N, \preceq_{\mathsf{ssrs}})$. Let A_k denote the set of points of rank k. k is at most $\frac{n(n+1)}{2}$. For each coalition S in A_k, its complement $N \backslash S$ is in $A_{n(n+1)/2-k}$; therefore $|A_k| = |A_{n(n+1)/2-k}|$. It is shown in [48] that the sequence $(|A_1|, \ldots, |A_{n(n+1)/2}|)$ is unimodal, i.e. first non-increasing, then non-decreasing. By this fact and the fact that $|A_k| = |A_{n(n+1)/2-k}|$, it must be the case that the largest antichain is $|A_{n(n+1)/4}|$. $|A_{n(n+1)/4}|$ is equal to the number of points $(x_1, \ldots, x_n)$ satisfying $x_1 + 2x_2 + \cdots + nx_n = \frac{n(n+1)}{4}$, and this number of points is equal to the middle coefficient of the polynomial $(1+q)(1+q^2)\cdots(1+q^n)$. It is shown in [36] that this middle coefficient is asymptotically equal to

$$\frac{\sqrt{\frac{2}{3}}\pi 2^n}{n\sqrt{n}}$$

Since every subset of an antichain is an antichain, there must me more than

$$2^{(\sqrt{\frac{2}{3}}\pi 2^n)/(n\sqrt{n})}$$

antichains in $(2^N, \preceq_{\mathsf{ssrs}})$. $\square$

4.1.2 Weighted Voting Games

To our knowledge, in game theory literature there has not been any research in the amount of Weighted Voting Games on n players. Fortunately there is a closely related field of research, called *threshold logic* (see for example [35]), that has some relevant results.

Definition 43 (Boolean threshold function, realization, **LT**)**.** Let f be a boolean function on n boolean variables. f is a *(boolean) threshold function* when there exists a weight vector of real numbers $r = (r_0, r_1, \dots r_n) \in \mathbb{R}^{n+1}$ such that $r_1x_1 + \cdots + r_nx_n \geq r_0$ iff $f(x_1, \dots, x_n) = 1$. We say that r *realizes* f. We denote the set of threshold functions of n variables $\{x_1, \dots, x_n\}$ by $\mathbf{LT}(n)$.[1]

Threshold functions resemble weighted voting games, except for that we talk about *boolean variables* instead of *players* now. Also, an important difference between threshold functions and weighted voting games is that $r_0, r_1, \dots, r_n$ are allowed to be negative for threshold functions, whereas $q, w_1, \dots, w_n$, must be non-negative in weighted voting games.

Zunic presents in [54] an upper bound on the number of threshold functions of n variables $|\mathbf{LT}(n)|$:

$$|\mathbf{LT}(n)| \leq 2^{n^2-n+1}. \tag{4.1}$$

Also, the following asymptotic lower bound is known, as shown in [53]: For large enough n, we have

$$|\mathbf{LT}(n)| \geq 2^{n^2(1-\frac{10}{\log n})}. \tag{4.2}$$

From these bounds, we can deduce some easy upper and lower bounds for $|\mathcal{G}_{\mathsf{wvg}}|$. First we observe the following property of the set of threshold functions on n variables.

Definition 44 (Non-negative threshold function, non-negative weight vector)**.** Let $\mathbf{LT}^+(n)$ be the set of *non-negative threshold functions* of variables $\{x_1, \dots, x_n\}$: threshold functions $f \in \mathbf{LT}(n)$ for which there exists a *non-negative weight vector* r that realizes f, i.e. r realizes f and only has non-negative entries.

Theorem 45. $|\mathcal{G}_{\mathsf{wvg}}(n)| = |\mathbf{LT}^+(n)|$

Proof sketch. There is an obvious one-to-one correspondence between the games in $\mathcal{G}_{\mathsf{wvg}}(n)$ and the threshold functions in $\mathbf{LT}^+(n)$. When f is a non-negative threshold function, it has a non-negative weight vector that realizes it, say $r = (r_0, r_1, \dots, r_n)$. $((r_1, \dots, r_n), r_0)$ is then a weighted form of some weighted voting game $G \in \mathcal{G}_{\mathsf{wvg}}(n)$. Using this idea, it is easy to see that $|\mathcal{G}_{\mathsf{wvg}}(n)| \geq |\mathbf{LT}^+(n)|$ and $|\mathcal{G}_{\mathsf{wvg}}(n)| \leq |\mathbf{LT}^+(n)|$.□

Now we can establish an upper bound on the number of weighted voting games.

Corollary 46. $|\mathcal{G}_{\mathsf{wvg}}(n)| \leq 2^{n^2-n+1}$.

Proof. This follows directly from Theorem 45, the upper bound (4.1), and the fact that $|\mathbf{LT}^+(n)| \subseteq |\mathbf{LT}(n)|$. □

[1]"LT" stands for "Linear Threshold function".

We will proceed by obtaining a lower bound on the number of weighted voting games.

Theorem 47. *For any n, for every threshold function in $\mathbf{LT}(n)\backslash\mathbf{LT}^+(n)$ and each $r \in \mathbb{R}^{n+1}$ that realizes f, there exists a $r' \in (\mathbb{R}^+)^{n+1}$ such that r is obtained by negating some of the entries in r', and r' is a realization of some threshold function in $\mathbf{LT}^+(n)$.*

Proof. Suppose for contradiction that there exists a threshold function $f \in \mathbf{LT}(n)\backslash\mathbf{LT}^+(n)$ for which this statement does not hold. f is a threshold function, so it has a realization $r' \in \mathbb{R}^{n+1}$. Now negate the negative entries in r' to obtain r. r is a non-negative weight vector, and for every non-negative weight vector there exists a non-negative threshold function that is realized by it. This contradicts the assumption. □

By these easy facts, the following lower bound follows.

Corollary 48. *For large enough n, it holds that $|\mathcal{G}_{\mathsf{wvg}}(n)| \leq 2^{n^2(1-\frac{10}{\log n})-n-1}$*

Proof. Let f be a non-negative threshold function and let r be a non-negative weight vector that realizes f. There are 2^{n+1} possible ways to negate the elements of r, so there are at most $2^{n+1}-1$ threshold functions $f' \in \mathbf{LT}(n)\backslash\mathbf{LT}^+(n)$ such that f' has a realization that is obtained by negating some of the elements of r. From this fact, and Theorem 47, it follows that $|\mathbf{LT}^+(n)| \geq \frac{|\mathbf{LT}(n)|}{2^{n+1}}$. Next, by Theorem 45 we obtain $|\mathcal{G}_{\mathsf{wvg}}(n)| \geq |\frac{\mathbf{LT}(n)}{2^{n+1}}|$. Now by using (4.2) we get $|\mathcal{G}_{\mathsf{wvg}}(n)| \geq \frac{2^{n^2(1-\frac{10}{\log n})}}{2^{n+1}} = 2^{n^2(1-\frac{10}{\log n})-n-1}$. □

We have obtained this lower bound on the number of weighted voting games by upper-bounding the factor, say k, by which the number of threshold functions is larger than the number of non-negative threshold functions. If we could find the value of k exactly, or lower-bound k, then we would also be able to sharpen the upper bound on the number of weighted voting games.

What about the canonical case, $\mathcal{G}_{\mathsf{cwvg}}(n)$? $\mathcal{G}_{\mathsf{cwvg}}(n)$ is a subset of $\mathcal{G}_{\mathsf{wvg}}(n)$, and for each non-canonical weighted voting game there exists a permutation of the players that makes it a canonical one. Since there are $n!$ possible permutations, it must be that $|\mathcal{G}_{\mathsf{cwvg}}(n)| \geq \frac{|\mathcal{G}_{\mathsf{wvg}}(n)|}{n!}$, and thus we obtain that

$$|\mathcal{G}_{\mathsf{cwvg}}(n)| \geq \frac{2^{n^2(1-\frac{10}{\log n})-n-1}}{n!}$$

for large enough n.

Unfortunately, we are again only able to upper-bound the factor by which the number of weighted voting games is larger than the number of canonical weighted voting games, so it seems hard to say much more about the cardinality of $\mathcal{G}_{\mathsf{cwvg}}(n)$.

4.2 The Synthesis Problem for Simple Games

In a *Voting Game Synthesis Problem*, we are interested in transforming a given simple game from one representation language into another representation language.

Definition 49 (Voting Game Synthesis (VGS) Problem)**.** Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages and let $f_{\mathcal{L}_1 \to \mathcal{L}_2} : \mathcal{L}_1 \to \mathcal{L}_2 \cup \{\mathsf{no}\}$ be the function that, on input ℓ,

- outputs no when G_ℓ is not in the class of games defined by $\mathcal{L}_2$, and
- otherwise maps a string $\ell \in \mathcal{L}_1$ to a string $\ell' \in \mathcal{L}_2$ such that $G_\ell = G_{\ell'}$.

In the *($\mathcal{L}_1, \mathcal{L}_2$)-Voting Game Synthesis Problem*, or *($\mathcal{L}_1, \mathcal{L}_2$)-VGS Problem*, we are given a string $\ell \in \mathcal{L}_1$ and we must compute $f_{\mathcal{L}_1 \to \mathcal{L}_2}(\ell)$.

4.3 Algorithms and Solutions for Voting Game Synthesis

In this section we will discuss algorithms and hardness results for various VGS problems. In sections 4.3.1, 4.3.2 and 4.3.3 we will consider respectively the problems of

- transforming games into weighted representation ($\mathcal{L}_{\mathsf{weights}}$);
- transforming games into roof- or ceiling-representation ($\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceiling}}$);
- transforming games into the languages $\mathcal{L}_W, \mathcal{L}_{W,\min}, \mathcal{L}_L, \mathcal{L}_{L,\max}$.

4.3.1 Synthesizing Weighted Representations

Of central importance in the next chapter of this thesis, is the fact that the problem ($\mathcal{L}_{W,\min}, \mathcal{L}_{\mathsf{weights}}$)-VGS has a polynomial time algorithm. This is a non-trivial result and was first stated in [37] by Peled and Simeone. In [37], the problem is stated in terms of set-covering problems. Because this algorithm is central to the approach we take in the next chapter for solving the PVGD-problem, we will here restate the algorithm in terms of simple games, and we will give a proof of its correctness and polynomial runtime.

We first introduce a new total order on the set of coalitions 2^N of a set of players $N = \{1, \ldots, n\}$.

Definition 50 (Positional representation)**.** Let $S \subseteq N = \{1, \ldots, n\}$ be a coalition. The *ith position* $p(i, S)$ of S is defined to be the player a in S such that $|\{1, \ldots, a\} \cap N| = i$. The *positional representation* of S, $\mathsf{pr}(S)$, is defined as the n-dimensional vector $(p'(1, S), \ldots, p'(n, S))$ where

$$p'(i, S) = \begin{cases} 0 \text{ if } |S| < i, \\ p(i, S) \text{ otherwise.} \end{cases}$$

for all i with $1 \leq i \leq n$.

As an example: if we have $N = \{1, \ldots, 5\}$ and $S = \{1, 4, 5\}$, then $\mathsf{pr}(S) = (1, 4, 5, 0, 0)$.

Definition 51 (PR-lexi-order)**.** The *PR-lexi-order* is the total order $(2^N, \preceq_{\mathsf{pr}})$, where for two coalitions $S \subseteq N$ and $S' \subseteq N$: $S \preceq_{\mathsf{pr}} S'$ if and only if $\mathsf{pr}(S)$ *lexicographically precedes* $\mathsf{pr}(S')$. A vector $\vec{v}$ lexicographically precedes another vector $\vec{v'}$ when there exists a i such that $v_i < v'_i$ and for all $j < i$ it holds that $v_i = v_j$.

For example, we see that for $N = \{1, \dots, 5\}$, we have $\{1, 2, 3\} \preceq_{\mathsf{pr}} \{1, 3, 5\}$. The least element of $(2^N, \preceq_{\mathsf{pr}})$ is $\varnothing$ and the greatest element of $(2^N, \preceq_{\mathsf{pr}})$ is N.

Now, we will introduce some operations that we can apply to coalitions. For this, the reader should recall definitions 17 and 20 from Chapter 2.

Definition 52 (fill-up, bottom right shift, bottom left shift, truncation, immediate successor)**.** Let N be the set of players $\{1, \dots, n\}$ and let $S \subseteq N$ be a coalition. The functions a and b are defined as follows.

- $b(S)$ is the largest index j such that $\chi(j, S) = 1$.
- $a(S)$ is the largest index j such that $\chi(j, S) = 0$ and $\chi(j + 1, S) = 1$ (if such a j does not exist, then $a(S) = 0$).

Now we can define the following operations on S:

- The *fill-up* of S: $\mathsf{fill}(S) = S \cup \{b(S) + 1\}$ (undefined if $S = N$).
- The *bottom right shift* of S: $\mathsf{brs}(S) = S \cup \{b(S)+1\} \backslash \{b(S)\}$ (undefined if $b(S) = n$).
- The *truncation* of S: $\mathsf{trunc}(S) = S \backslash \{a(S) + 1, \dots n\}$.
- The *immediate successor* of S:

$$\mathsf{succ}(S) = \begin{cases} \mathsf{fill}(S) \text{ if } n \notin S \ , \\ \mathsf{brs}(S \backslash \{n\}) \text{ if } n \in S \ . \end{cases}$$

The immediate successor operation is named as such because it returns the successor of S in the total order $(2^N, \preceq_{\mathsf{pr}})$.

One last concept we need is that of a *shelter* coalition.

Definition 53 (Shelter)**.** A *shelter* is a minimal winning coalition S such that $\mathsf{brs}(S)$ is losing or undefined.

Note that the set of roof coalitions of a canonical linear game is a subset of the set of shelter coalitions of that game.

The Hop-Skip-and-Jump Algorithm

We are now ready to state the algorithm. The input to the algorithm is a string ℓ in $\mathcal{L}_{W,\min}$, i.e. the list of characteristic vectors describing the set of minimal winning coalitions $W_{\min}$. The four main steps of the algorithm are:

1. Check whether G_ℓ is a linear game. If not, then stop. When it turns out that the game is linear, find a permutation of the players that turns the game into a canonical linear game. In the remaining steps, we assume that G_ℓ is a canonical linear game.
2. Generate a list of shelters $\mathcal{S}$, sorted according to the PR-lexi-order.

3. Use $\mathcal{S}$ as input for the *Hop-Skip-and-Jump* algorithm. The Hop-Skip-and-Jump algorithm will give as output the set of all maximal losing coalitions $L_{\max}$. This step, the Hop-Skip-and-Jump algorithm, is the most non-trivial part, and we will explain it in detail below.

4. Use $W_{\min}$ and $L_{\max}$ to generate the following system of linear inequalities, and solve it for any choice of q in order to find the weights $w_1, \dots, w_n$:

$$\begin{aligned} w_1\chi(1,S) + \cdots + w_n\chi(n,S) \geq q, \forall S \in W_{\min} \\ w_1\chi(1,S) + \cdots + w_n\chi(n,S) < q, \forall S \in L_{\max} \end{aligned} \tag{4.3}$$

   If this system of linear inequalities has no solutions, then G_ℓ is not weighted; and otherwise the weights that have been found are the weights of the players, and q is the quota: $((w_1, \dots, w_n), q)$ is a weighted form of the weighted voting game.

The first step of the algorithm is easy if we use an algorithm by Aziz, given in [2]. This algorithm decides whether a monotone simple game represented as a listing of minimal winning coalitions is a linear game, and if so it outputs a strict desirability order[2]. From the strict desirability order, the required permutation of the players directly follows.

The generation of the sorted list of shelters is clearly doable in polynomial-time: We can easily check for each minimal winning coalition whether its bottom right shift is losing.

Linear programs are solvable in a time that is polynomial in the size of the linear program, by Karmarkar's algorithm [22] for example. So for the linear program of the fourth part of the algorithm we will have to show that its size is a polynomial in n and the amount of minimal winning coalitions, i.e., we will have to show that there are only polynomially many more maximal losing coalitions than that there are minimal winning coalitions. This follows from the fact that the Hop-Skip-and-Jump algorithm (see below) runs in polynomial time and hence can output only a polynomial number of coalitions. Lastly, the fact that we can choose any q we want follows from Theorem 41.

The hard part that now remains is part three of the algorithm: outputting the list of maximal losing coalitions, given a sorted list of shelter coalitions. This is what the Hop-Skip-and-Jump-algorithm does. We will now state this algorithm, prove it correct and show that the runtime is a polynomial in the amount of players n and the number of shelter coalitions t. From this polynomial runtime it then also follows that $|L_{\max}|$ is polynomially bounded in $|W_{\min}|$.

The pseudocode for the Hop-Skip-and-Jump algorithm is given in Algorithm 1. The basic idea is to traverse all coalitions in order, according to the PR-lexi-order, and output those coalitions that are maximal losing coalitions. During this process, we will be able to skip huge intervals of coalitions in order to achieve a polynomial run-time.

We will now proceed by giving a correctness-proof of this algorithm.

Theorem 54. *Algorithm 1 outputs only maximal losing coalitions.*

[2]With this, we mean that the algorithm outputs a list $\vec{P} = (P_1, \dots, P_j)$ such that $\{P_1, \dots, P_j\}$ is a partition of the set of all players N, where the players of any set in this partition are pairwise equally desirable and for all i and j with $i > j$ we have that any player in P_i is strictly more desirable than any player in P_j.

Algorithm 1 The Hop-Skip-and-Jump algorithm. A polynomial-time algorithm that outputs the set of maximal losing coalitions of a monotone simple game G on players $N = \{1, \ldots, n\}$, given the sorted list of shelters of G as input. An assumption we make in this algorithm is that the empty-coalition does not occur in the list of shelters. If it does, it becomes a trivial task to output the list of maximal losing coalitions, so it is a safe assumption.

```
1: nextshelter := first shelter on the list. {Output $N$ and stop if the list is empty.}
2: currentcoalition := $\varnothing$ {Start with the least coalition in the PR-lexi-order.}
3: loop
4:   while currentcoalition $\neq$ nextshelter$\setminus\{b($nextshelter$)\}$ do
5:     if $n \notin$ currentcoalition then
6:       currentcoalition := fill(currentcoalition)
7:     else
8:       output currentcoalition
9:       currentcoalition := brs(trunc(currentcoalition)) {Stop if undefined.}
10:    end if
11:  end while
12:  if $n \notin$ nextshelter then
13:    currentcoalition := brs(nextshelter)
14:  else
15:    output currentcoalition
16:    currentcoalition := succ(nextshelter) {Stop if nextshelter = $\{n\}$.}
17:  end if
18:  nextshelter := next shelter on the list.
19: end loop
```

Proof. There are three places at which Algorithm 1 outputs coalitions: line 1, 8 and line 15.

At line 1, a coalition is only output when the list of shelters is empty. When this list is empty, it means there are no winning coalitions, so N is the only maximal losing coalition.

At line 15 we see that currentcoalition $\subset$ nextshelter, and nextshelter is a minimal winning coalition, so currentcoalition must be losing. Also, at line 15, $n \in$ nextshelter. This means that any superset of currentcoalition is a superset of a leftshift of nextshelter, and therefore winning. So we conclude that currentcoalition is a maximal losing coalition. This establishes that at line 15, all coalitions output are maximal losing coalitions.

Now we need to show the same for line 8. For this, we first need to prove the following invariant.

Lemma 55. *When running Algorithm 1, directly after executing line 2, line 18, and each iteration of the while-loop of line 4,* currentcoalition *is a losing coalition.*

Proof. We prove all three cases separately.

- *Directly after executing line 2,* currentcoalition is the empty coalition and thus losing by assumption.

- *Directly after executing line 18*, we have two subcases:

  Case 1: *After the last time the execution of the algorithm passed line 11, lines 12 and 13 were executed while lines 14–16 were skipped.* In this case, $\mathsf{currentcoalition}$ is a bottom right shift of a shelter, so $\mathsf{currentcoalition}$ is losing by the definition of a shelter.

  Case 2: *After the last time the execution of the algorithm passed line 11, lines 14–16 were executed while lines 12 and 13 were skipped.* In this case, $\mathsf{currentcoalition}$ is a direct successor of a shelter s containing player n, by definition of the direct successor function, $\mathsf{currentcoalition}$ is a subset of s and hence losing.

- *Directly after each iteration of the while-loop of line 4.* We can use induction for this final case. By the preceding two cases in this list, that we proved, we can assume that $\mathsf{currentcoalition}$ is losing when the while-loop is entered. It suffices now to show that $\mathsf{currentcoalition}$ is losing after a single repetition of the while-loop. We divide the proof up again, in two cases:

  Case 1: *During the execution of the while-loop, lines 5 and 6 were executed while lines 7–9 were skipped.* Then $\mathsf{currentcoalition}$ is a fill-up of a losing coalition, say l. Let i be the agent that was added by the fill-up, i.e., $\mathsf{currentcoalition} = l \cup \{i\}$. Suppose for contradiction that $\mathsf{currentcoalition}$ is winning; then $i \in \mathsf{nextshelter}$ and $i - 1 \in \mathsf{nextshelter}$. It must also be true that $l \subseteq \mathsf{nextshelter}$ because otherwise l is a right-shift of $\mathsf{nextshelter}$ and therefore winning (the induction hypothesis states that l is losing). Therefore $l = \mathsf{nextshelter} \backslash \{b(\mathsf{nextshelter})\}$. But then execution would have left the loop because of line 4. Contradiction.

  Case 2: *In the execution of the while-loop, lines 7–9 were executed while lines 5 and 6 were skipped.* $\mathsf{currentcoalition}$ is a bottom right shift of a truncation of a losing coalition. A truncation of a losing coalition is losing, and a bottom right shift of a losing coalition is losing, so $\mathsf{currentcoalition}$ is losing.

□

From the lemma above, it follows that at line 8, $\mathsf{currentcoalition}$ is losing. To show that it is also maximal, we divide the proof up in three cases:

Case 1: *The execution of the algorithm has never passed line 11.* In this case $\mathsf{currentcoalition}$ at line 8 is obtained by a series of successive fill-ups starting from the empty coalition, and $n \in \mathsf{currentcoalition}$. This means that $\mathsf{currentcoalition} = N$, so $\mathsf{currentcoalition}$ is clearly maximal.

Case 2: *The execution of the algorithm did pass line 11 at least once, and the last time that execution has done so lines 12 and 13 were executed while lines 14–16 were skipped.* In this case we have that at line 8, $\mathsf{currentcoalition}$ is obtained by a series of fill-ups of a bottom right shift of a shelter-coalition s. It follows that adding any player to $\mathsf{currentcoalition}$ will turn $\mathsf{currentcoalition}$ into a winning coalition, because $\mathsf{currentcoalition}$ would then become a superset of a left-shift of s. So $\mathsf{currentcoalition}$ is maximal.

Case 3: *The execution of the algorithm did pass line 11 at least once, and the last time that execution has done so lines 14–16 were executed while lines 12 and 13 were skipped.* In this case we have at line 8 that $\mathsf{currentcoalition}$ is the successor of a shelter s that has player n in it. By the definition of the successor function we get that adding any player to $\mathsf{currentcoalition}$ would make it a superset of a left-shift of s, and thus winning. So $\mathsf{currentcoalition}$ is maximal.

□

Theorem 56. *Algorithm 1 outputs all maximal losing coalitions.*

Proof. By Theorem 54 we have that Algorithm 1 outputs only maximal losing coalitions, so what suffices is to show that the intervals of coalitions that Algorithm 1 does not output, do not contain any losing coalitions.

Let s be a coalition that is not output by Algorithm 1. There are several cases possible.

Case 1: *There is a point when the execution of the algorithm has just passed line 6, such that* $\mathsf{currentcoalition} = s$. In that case s is losing, following from Lemma 55.

Case 2: *There is a point when the execution of the algorithm has just passed line 8, such that* $\mathsf{currentcoalition} \preceq_{\mathsf{pr}} s \preceq_{\mathsf{pr}} \mathsf{brs}(\mathsf{trunc}(\mathsf{currentcoalition}))$. Now s is a direct right-shift of a point s' that the algorithm has output. s' is maximal losing so s is not maximal losing.

Case 3: *There is a point when the execution of the algorithm has just passed line 12, such that* $\mathsf{currentcoalition} \preceq_{\mathsf{pr}} s \preceq_{\mathsf{pr}} \mathsf{brs}(\mathsf{nextshelter})$. Here we have that s is either a right-shift of $\mathsf{currentcoalition}$ or a left-shift of a superset of $\mathsf{nextshelter}$. In the former case, s is not a maximal losing coalition because it is a right-shift of $\mathsf{currentcoalition}$, and $\mathsf{currentcoalition}$ is not a maximal losing coalition because it is a strict subset of the bottom right shift of $\mathsf{nextshelter}$, which is also losing. In the latter case, s is winning, so s can not be maximal losing.

□

By the two theorems above, we have established that the Hop-Skip-and-Jump algorithm works correctly. Now we will also show that it runs in polynomial time.

Theorem 57. *Algorithm 1 runs in time $O(n^3t)$ (where t is the number of shelter coalitions).*

Proof. When repeatedly executing the while-loop of line 4, lines 5 and 6 can be executed only n consecutive times, before lines 7–9 are executed. Line 9 can be executed at most n times in total, given that the execution does not leave the while-loop (after n times, the operation done at line 9 is undefined, and execution stops). It follows that the while-loop is executed at most n^2 consecutive times before execution leaves the while-loop. Each time lines 12–18 are executed, one shelter is taken from the list, so lines 12–18 are executed only t times. The fill-up operation, bottom right shift operation, successor operation and truncation operation can all be implemented in $O(n)$ time. So, bringing everything together, we arrive at a total runtime of $O(n^3t)$. □

4.3.2 Synthesizing Roof- and Ceiling-representations

Now let us consider the problem of synthesizing various representations of games into the roof- and ceiling-representation of a canonical linear game.

Let us start with the problem $(\mathcal{L}_{\mathsf{W}}, \mathcal{L}_{\mathsf{roof}})$-VGS. This problem boils down to solving the $(\mathcal{L}_{\mathsf{W},\min}, \mathcal{L}_{\mathsf{roof}})$-VGS problem, since $(\mathcal{L}_{\mathsf{W}}, \mathcal{L}_{\mathsf{W},\min})$-VGS is easy (just check for each coalition in W whether it is minimal, and if so, it is in $W_{\min}$). The same holds for the problems $(\mathcal{L}_{\mathsf{L}}, \mathcal{L}_{\mathsf{ceil}})$-VGS and $(\mathcal{L}_{\mathsf{L},\max}, \mathcal{L}_{\mathsf{ceil}})$-VGS.

Solving $(\mathcal{L}_{\mathsf{W},\min}, \mathcal{L}_{\mathsf{roof}})$-VGS is also not very difficult. As pointed out before in this chapter, we can use a polynomial-time algorithm in [2] in order to check whether a monotone simple game given as a list of minimal winning coalitions is weighted, and we can obtain the strict desirability order if this is the case. It could be that it turns out the game is linear, but not canonical. If we wish, we are then also able to permute the players so that we end up with a canonical linear game. After that, all that we have to do is check for each minimal winning coalition C whether each of its direct right shifts (no more than n direct right-shifts are possible) are losing coalitions. If that is the case, then C must be a roof. For the problem $(\mathcal{L}_{\mathsf{L},\max}, \mathcal{L}_{\mathsf{ceil}})$-VGS, the situation is analogous.

What also follows now, is that the problems $(\mathcal{L}_{\mathsf{W},\min}, \mathcal{L}_{\mathsf{ceil}})$-VGS can be solved in polynomial time: the Hop-Skip-and-Jump algorithm of the previous section is able to generate in polynomial a list of maximal losing coalitions from the list of minimal winning coalitions. The problem $(\mathcal{L}_{\mathsf{L},\max}, \mathcal{L}_{\mathsf{roof}})$-VGS is also solvable in polynomial time by running a "dual" version of the Hop-Skip-and-Jump algorithm where we

- permute the players according to the permutation π that permutes the players into a sequence where the players are ordered in *ascending* desirability, i.e., the least desirable player is now player 1, and the most desirable player is player n.
- treat losing coalitions as winning coalitions and vice versa, during the execution of the Hop-Skip-and-Jump algorithm.

once the Hop-Skip-and-Jump algorithm is done, we have a list of coalitions. For each C in this list, $\pi^{-1}(C)$ is a minimal winning coalition.

As a consequence, by polynomial time solvability of $(\mathcal{L}_{\mathsf{W}}, \mathcal{L}_{\mathsf{W},\min})$-VGS and $(\mathcal{L}_{\mathsf{L}}, \mathcal{L}_{\mathsf{L},\max})$-VGS we also have that $(\mathcal{L}_{\mathsf{L}}, \mathcal{L}_{\mathsf{roof}})$-VGS and $(\mathcal{L}_{\mathsf{W}}, \mathcal{L}_{\mathsf{roof}})$-VGS admit a polynomial time algorithm.

Is the problem $(\mathcal{L}_{\mathsf{ceiling}}, \mathcal{L}_{\mathsf{roof}})$-VGS solvable in polynomial time? This turns out to not be the case. We will give a family of examples in which the number of roof coalitions is exponential in n, while the number of ceiling coalitions is only polynomial in n. As a consequence, any algorithm that generates the list of roofs from the list of ceilings will run in exponential time in the worst case. By symmetry it also follows that $(\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceiling}})$-VGS is not solvable in polynomial time.

This example that we are now going to give, together with its analysis, is not so straightforward. Let us first define the following specific type of coalition.

Definition 58 ((k, i)-encoding coalition)**.** Let $N = \{1, \dots, n\}$ be a a set of players such that $n = 4i$ for some $i \in \mathbb{N}$. For any k satisfying $0 \leq k < 2^i - 1$, the (k, i)*-encoding* *coalition* $S_{k,i} \subseteq N$ is then defined as

$$\{4(j-1)+2, 4(j-1)+3 \mid \text{The } j\text{th bit in the binary representation of } k \text{ equals } 0.\} \quad \cup$$
$$\{4(j-1)+1, 4(j-1)+4 \mid \text{The } j\text{th bit in the binary representation of } k \text{ equals } 1.\}$$

For example, $S_{2,2} = \{1,4,6,7\}$, and $S_{5,3} = \{1,4,6,7,9,12\}$.

Definition 59 (i-bit roof game)**.** Let $N = \{1,\dots,n\}$ be a a set of players such that $n = 4i$ for some $i \in \mathbb{N}$. The *i-bit roof game* on N, denoted $G_{i-\mathsf{bit}}$, is the canonical linear game such that the set of roof coalitions of G is $\{S_{0,i},\dots,S_{2^i-1,i}\}$.

For example, the 2-bit roof game, $G_{2-\mathsf{bit}}$, consists of the roofs $\{\{2,3,6,7\}, \{2,3,5,8\}, \{1,4,6,7\}, \{1,4,5,8\}\}$. $G_{i-\mathsf{bit}}$ is well-defined for all i because the binary representations of two arbitrary i-bit numbers k and k' differ in at least one bit. Therefore, $S_{i,k}$ is not a superset of a left-shift of $S_{i,k'}$ and hence the set of roofs that we have defined for $G_{i-\mathsf{bit}}$ is indeed a valid set of roofs (i.e. there are no two roofs such that one is a left shift of another).

$G_{i-\mathsf{bit}}$ has $2^i = 2^{\frac{n}{4}}$ roofs, i.e. an exponential number in n. We will show that the number of ceilings in $G_{i-\mathsf{bit}}$ is only polynomially bounded (in fact, linear in n). First let us use the following definitions for convenience.

Definition 60 (Accepting roof set)**.** Let $G \in \mathcal{G}_{\mathsf{clin}}(n)$ be a canonical linear game on players $N = \{1,\dots,n\}$. Let $C \subseteq N$ be a coalition, let a be a number such that $1 \le a \le |C|$, and let p be the ath most desirable player in C. The *accepting set of roofs of the ath player in C*, denoted $A(C,a)$, is the set consisting of those roof coalitions R for which the ath most desirable player in R is greater than or equal to p, or $|R| < a$.

Alternatively stated, the accepting roof set say the ath most desirable player in a coalition is the set of all roofs R such that the ath player in R is a less desirable.

It is important to now observe that the following fact holds.

Lemma 61. *In a canonical linear game, a coalition C is winning if and only if $\bigcap_a A(C,a) \neq \varnothing$.*

Proof. This Lemma is in fact an equivalent statement of the fact that a coalition is winning if and only if it is a left shift of a winning coalition. □

Using the notion of an accepting roof set, we can prove the following technical lemma.

Lemma 62. *Let C be a ceiling of $G_{i-\mathsf{bit}}$ with more than one direct left shift and let p be an arbitrary player that we can apply the direct left shift operation on, i.e., let p be a player such that $C_1 = C \cup \{p-1\} \backslash \{p\}$ is a direct left shift of C. Also, let a be the number such that p is the ath most desirable player in C. Then $p = 2a$.*

Proof. Observe that for all b it holds that every roof of $G_{i-\mathsf{bit}}$ has as its bth most desirable player either player $2b-1$ or player $2b$. By construction of $G_{i-\mathsf{bit}}$, the number of roofs of $G_{i-\mathsf{bit}}$ that contain player $2b-1$ is $\frac{2^i}{2}$, and the number of roofs that contain player $2b$ is also $\frac{2^i}{2}$.

C has more than one direct left shift, so there must be another player p', $p' \neq p$, such that $C_2 = C \cup \{p' - 1\} \backslash \{p'\}$ is a direct left shift of C.

First we will show that $p \leq 2a$. Assume therefore that $p > 2a$. Now we have that $|A(C, a)| = 0$, so then $|A(C_2, a)| = 0$ and hence $\bigcap_a A(C_2, a) = \varnothing$. We see that C_2 is losing, but C_2 is a direct left shift of ceiling C, so C_2 is winning. This is a contradiction, so $p \leq 2a$.

Now we will show that $p \geq 2a$. Assume therefore that $p < 2a$. Now we have that $|A(C, a)| = 2^i$, so then $A(C_1, a) = 2^i$. Now it must be that $\bigcap_a A(C_1, a) = \bigcap_a A(C, a)$. But $\bigcap_a A(C, a) = \varnothing$ because C is losing, and therefore $\bigcap_a A(C_1, a) = \varnothing$ so C_1 is losing. C_1 is also winning, because it is a left shift of ceiling C. This is a contradiction, so $p \geq 2a$.

Now that we know that $p \geq 2a$ and $p \leq 2a$, we can conclude that $p = 2a$. ☐

Lemma 63. *In $G_{i-\text{bit}}$, a ceiling does not have more than two direct left shifts.*

Proof. For contradiction, let C be a ceiling with more than two direct left shifts. Let k be the number of direct left shifts of C, and let $P = \{p_1, \ldots, p_k\}$ be the set containing the players of C that we can apply the direct left shift operation on (we say that we can apply the direct left shift operation on a player q when $C \cup \{q - 1\} \backslash \{q\}$ is a left shift of C). Let $\mathcal{A} = \{a_1, \ldots, a_k\}$ then be the numbers such that p_j is the a_jth most desirable player in C, for all i with $1 \leq j \leq k$. For any $j \in \{1, \ldots, i\}$ and any $b \in \{0, 1\}$, let $R(j, b)$ denote the following set of roofs of $G_{i-\text{bit}}$:

$$R(j, b) = \{S_{k,i} \mid \text{The } j\text{th bit of the binary representation of } k \text{ is } b. \}$$

Observe that by the previous lemma, there is a k-tuple of bits $(b_1, \ldots, b_k) \in \{0, 1\}^k$ such that for all j with $1 \leq j \leq k$:

$$A(C, a_j) = R(\lceil p_j/4 \rceil, k_j).$$

There are now two cases:

Case 1: *All of the players $\{p_1, \ldots, p_k\}$ are in different multiples of 4, i.e., $\lceil p_1/4 \rceil \neq \lceil p_2/4 \rceil \neq \cdots \neq \lceil p_k/4 \rceil$.* Then by the properties of the binary numbers, the intersection $\bigcap_{a \in \mathcal{A}} A(C, a) = \bigcap_{p \in P} R(\lceil p/4 \rceil, b)$ is not empty, therefore C must be winning, which is in contradiction with C being a ceiling. So this case is impossible.

Case 2: *There are two players p and p', both in P, that are in the same multiple of 4, i.e., $\lceil p/4 \rceil = \lceil p'/4 \rceil$.* Assume without loss of generality that $p < p'$. Then $A(C, a) \cap A(C, a') = \varnothing$. But then we would be able to apply a direct left shift on player p'' without turning C into a winning coalition, i.e., $C \cup \{p'' - 1\} \backslash \{p''\}$ is winning. But C is a ceiling, so that is a contradiction.

From the previous lemma it follows that there can not be more than two players that are the same multiple of 4, so these two cases are indeed exhaustive. Both cases are impossible, so we must reject the assumption that there exists a ceiling C with more than two left shifts. ☐

It is easy to see that there are no more than $O(n^5)$ coalitions with two left shifts, there are no more than $O(n^3)$ coalitions with one left shift, and there are no more than $O(n)$ coalitions with no left shifts. so we get the following corollary.

Corollary 64. *The game $G_{i-\mathsf{bit}}$ (on $n = 4i$ players) has $O(n^5)$ ceilings.*

In fact, using careful analysis we think it is possible to show that the number of ceilings is only linear in n, but proving that is not in scope with our goal that we just achieved: we can now conclude that $\{G_{i-\mathsf{bit}} \mid i \in \mathbb{N}\}$ is an infinite family of examples in which there are exponentially many more roofs than ceilings. Hence, finally we obtain:

Corollary 65. *there is no polynomial time algorithm for ($\mathcal{L}_{\mathsf{ceiling}}, \mathcal{L}_{\mathsf{roof}}$)-VGS and ($\mathcal{L}_{\mathsf{ceiling}}, \mathcal{L}_{\mathsf{roof}}$)-VGS.*

4.3.3 Other Voting Game Synthesis Problems & Summary of Results

In this section we will discuss some of the remaining variants of the Voting Game Synthesis problem that we did not discuss in the other sections. At the end of this section, Table 4.1 summarizes all of the results that we have discussed and obtained in this chapter.

First of all, Freixas et al. investigate in [18] the $(\mathcal{L}_1, \mathcal{L}_2)$-VGS problem for $\mathcal{L}_1$ and $\mathcal{L}_2$ being any choice of language in $\{\mathcal{L}_W, \mathcal{L}_{W,\min}, \mathcal{L}_L, \mathcal{L}_{L,\max}\}$. Most of their results follow rather easily from what has already been said in this chapter. One of their results that do not yet follow from this chapter is that $(\mathcal{L}_W, \mathcal{L}_L)$-VGS and $(\mathcal{L}_L, \mathcal{L}_W)$-VGS does not have a polynomial time algorithm. This is not hard to see: there are instances where there are exponentially many more losing coalitions than that there are winning coalitions. Consider for instance the game in which only the grand coalition is winning. In this game there are $2^n - 1$ losing coalitions, so it takes exponential time to list them all. This game is also a canonical linear game and a weighted voting game, so even if we restrict games to be weighted, or canonical linear, it still holds that $(\mathcal{L}_W, \mathcal{L}_L)$-VGS and $(\mathcal{L}_L, \mathcal{L}_W)$-VGS do not have polynomial time algorithms.

In [18] it is also shown that $(\mathcal{L}_{W,\max}, \mathcal{L}_{L,\min})$-VGS is in general not polynomial time solvable. The authors show this by giving a family of examples of monotone games which have exponentially many more maximal losing coalitions than minimal winning coalitions. However, the Hop-Skip-and-Jump algorithm that we described does actually solve $(\mathcal{L}_{W,\max}, \mathcal{L}_{L,\min})$-VGS in polynomial time for the restriction to linear games.

Another set of Voting Game Synthesis Problems that we have not yet discussed is the $(\mathcal{L}_{\mathsf{weights}}, \mathcal{L}_2)$-VGS case, for any choice of $\mathcal{L}_2$. In this case it always holds that there is no polynomial time algorithm for the problem:

- When $\mathcal{L}_2 = \mathcal{L}_W$, consider the weighted voting game in which the quota is 0. Now there are 2^n minimal winning coalitions, so the output is exponentially larger than the input. The case $\mathcal{L}_2 = \mathcal{L}_L$ is analogous, but now we take a weighted voting game in which the quota is larger than the sum of all weights, so that there are no winning coalitions.

- When $\mathcal{L}_2 = \mathcal{L}_{W,\min}$ or $\mathcal{L}_2 = \mathcal{L}_{L,\max}$, we see that the weighted voting game in which every player's weight is 1 and the quota is $\lfloor \frac{n}{2} \rfloor$ has an exponential amount of minimal winning coalitions and maximal losing coalitions: any coalition of size $\lfloor\frac{n}{2}\rfloor$ is minimal winning, and any coalition of size $\lfloor\frac{n}{2}\rfloor - 1$ is maximal losing. There are respectively $\binom{n}{\lfloor\frac{n}{2}\rfloor}$ and $\binom{n}{\lfloor\frac{n}{2}\rfloor-1}$ such coalitions. By using Stirling's approximation, we can see that both these expressions are exponential in n.

- When $\mathcal{L}_2 = \mathcal{L}_{\text{roof}}$ it follows directly from the proof of Theorem 42 that the weighted voting game in which player i gets weight $n - i + 1$ and the quota is equal to $n(n + 1)/4$, has an exponential number of roofs. We do not know whether there is also a weighted voting game with an exponential number of ceilings, but we suspect so.

That completes our study of the voting game synthesis problem. As said before, table 4.1 summarizes all of the results that we have discussed and obtained. It indicates for each variant of the Voting Game Synthesis problem whether it is solvable in polynomial time (P), or does not have a polynomial time algorithm (EXP). We see that the complexities of three problems remain open, namely the problems of

- transforming roof-representations of canonical linear games into weighted representations,
- transforming ceiling-representations of canonical linear games into weighted representations,
- transforming weighted-representations of weighted voting games into ceiling representations.

Table 4.1: Time complexities of the various $(\mathcal{L}_1, \mathcal{L}_2)$-VGS problems that we have discussed in this chapter.

$\mathcal{L}_2 \rightarrow$ $\mathcal{L}_1 \downarrow$	$\mathcal{L}_W$	$\mathcal{L}_{W,\min}$	$\mathcal{L}_L$	$\mathcal{L}_{L,\max}$	$\mathcal{L}_{\text{roof}}$	$\mathcal{L}_{\text{ceil}}$	$\mathcal{L}_{\text{weights}}$
$\mathcal{L}_W$	-	P	EXP	P	P	P	P
$\mathcal{L}_{W,\min}$	EXP	-	EXP	EXP (P if linear)	P	P	P
$\mathcal{L}_L$	EXP	P	-	P	P	P	P
$\mathcal{L}_{L,\max}$	EXP	EXP (P if linear)	EXP	-	P	P	P
$\mathcal{L}_{\text{roof}}$	EXP	EXP	EXP	EXP	-	EXP	?
$\mathcal{L}_{\text{ceil}}$	EXP	EXP	EXP	EXP	EXP	-	?
$\mathcal{L}_{\text{weights}}$	EXP	EXP	EXP	EXP	EXP	?	-

Chapter 5

Solving PVGD Problems

From the existing literature on the power index voting game design problem, we see that researchers have only considered heuristic methods for the case where a weighted representation must be output. Even stronger: the weighted representation is the only representation that current voting game design algorithms *internally* work with. No other methods of representing a game have even been considered.

A second fact that stands out is that (as of yet) an exact algorithm for voting game design problems is not known. An interesting question that we could ask is whether there even *exists* a method to exactly compute the optimal answer to a voting game design problem. There exists an *infinite* number of weighted representations for each WVG (this follows from Theorem 41). Hence, it seems not that surprising that no algorithm has been developed to exactly solves the problem.

Nevertheless, it turns out that we can fortunately answer this question positively: there do exist exact algorithms for voting game design problems. In what follows in this chapter, we will look into exact algorithms for various power index voting game design problems. Of course, the most important among these problems is the variant in which we must find a weighted voting game, and output it in a weighted representation.

We approach the voting game design problem by devising an enumeration method that generates every voting game relatively efficiently. First, we devise a "naive" method that enumerates all monotone games in doubly exponential time (Section 5.1). Subsequently, in Section 5.2, for the case of weighted voting games we improve on this runtime exponentially by showing how to enumerate all WVGs within exponential time. Although the runtime of this enumeration method is still exponential, we will see that the algorithm for the power index weighted voting game problem that results from this has the *anytime* property: the longer we run it, the better the result becomes. Also, we are guaranteed that the algorithm eventually finds the *optimal* answer. The enumeration method is based on exploiting a new specific partial order on the class of weighted voting games. This partial order can be considered interesting in its own right, from a mathematical point of view.

Prior to reading this chapter, the reader may wish to go over some of the material on order theory, that is given in the first section of Appendix A.

5.1 Monotone Game Design

In this section we will consider the power index voting game design problem for the class of monotone games $\mathcal{G}_{\mathsf{mon}}$. There are four representation languages that can be used for monotone games:

- $\mathcal{L}_W$, the winning coalition listing;
- $\mathcal{L}_L$, the losing coalition listing;
- $\mathcal{L}_{W,\min}$, the minimal winning coalition listing;
- $\mathcal{L}_{L,\max}$, the maximal losing coalition listing.

From these languages, we obtain the following four different power index voting game design problems: $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_W)$-PVGD, $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_L)$-PVGD, $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD and $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{L,\max})$-PVGD. For f we can then choose any power index. Of these problems, the cases of $\mathcal{L}_{W,\min}$ and $\mathcal{L}_{L,\max}$ are the most interesting, because these languages both define the class of monotone games.

We do not know of any practical situations in which this problem occurs. Therefore, we will only address this problems briefly and show for theoretical purposes that the optimal answer is computable. We do this by giving an exact algorithm.

An exact algorithm that solves $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD or $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{L,\max})$-PVGD must search for the antichain of coalitions that represents the game that has a power index closest to the target power index. This antichain of coalitions could either be a set of minimal winning coalitions, or a set of maximal losing coalitions. In either way, a simple exact algorithm for this problem would be one that considers every possible antichain, and computes for each antichain the power index for the game that the antichain represents.

Algorithm 2 describes the process more precisely for the case that the representation language is $\mathcal{L}_{W,\min}$. We will focus on $\mathcal{L}_{W,\min}$ from now on, because the case for $\mathcal{L}_{L,\max}$ is entirely analogous. An algorithm for the languages $\mathcal{L}_W$ and $\mathcal{L}_L$ can be obtained by applying the transformation algorithm discussed in the last chapter.

From line 3, we see that we need to enumerate all antichains on the grand coalition. As we already said in Section 4.1, the amount of antichains we need to enumerate is D_n, the nth Dedekind number.

Because the n'th Dedekind number D_n quickly grows very large (in n), line 3 is what gives the algorithm a very high complexity. The following bounds are known [24].[1]

$$2^{(1+c'\frac{\log n}{n})E_n} \geq D_n \geq 2^{(1+c2^{-n/2})E_n}, \tag{5.1}$$

where c' and c are constants and E_n is the size of the largest antichain on an n-set. Sperner's theorem tells us the following about E_n [47]:

[1]Also, Korshunov devised an asymptotically equal expression [26]: $D_n \sim 2^{C(n)} e^{c(n)2^{-n/2}+n^2 2^{-n-5}-n2^{-n-4}}$ with $C(n) = \binom{n}{\lfloor \frac{n}{2} \rfloor}$ and $c(n) = \binom{n}{\lfloor \frac{n}{2} \rfloor - 1}$. In [26], this expression is described as the number of monotone boolean functions, which is equal to the nth Dedekind number.

Algorithm 2 A straightforward algorithm for solving $(f, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD. The input is a target power index $\vec{p} = (p_1, \dots, p_n)$. The output is an $\ell \in \mathcal{L}_{W,\min}$ such that $f(G_\ell)$ is as close as possible to $\vec{p}$.

```
1:  bestgame := $0$ {bestgame keeps track of the best game that we have found, represented
    as a string in $\mathcal{L}_{W,\min}$.}
2:  besterror := $\infty$ {besterror is the error of $f(G_{\mathsf{bestgame}})$ from $\vec{p}$, according to the sum-
    ofsquarederrors measure.}
3:  for all $\ell \in \mathcal{L}_{W,\min}$ do
4:    Compute $f(G_\ell) = (f(G_\ell, 1), \dots, f(G_\ell, n))$.
5:    error := $\sum_{i=1}^{n}(f(G_\ell, i) - p_i)^2$.
6:    if error $<$ besterror then
7:      bestgame := $\ell$
8:      besterror := error
9:    end if
10: end for
11: return bestgame
```

Theorem 66 (Sperner's theorem)**.**

$$E_n = \binom{n}{\lfloor \frac{n}{2} \rfloor}.$$

From Sperner's theorem and Stirling's approximation (Theorem 94) of the factorial function, we get

$$E_n \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \tag{5.2}$$

A possible approach to enumerate antichains would be to simply enumerate each family of coalitions, and check if that family is an antichain. Is this approach an efficient one?

In total, there are 2^{2^n} families of coalitions[2]. Now let us suppose that D_n equals the upper bound of (5.1). Substituting the tight bound of 5.2 into the upper bound of (5.1), we get

$$D_n \leq 2^{(1+c'\frac{\log n}{n})k\frac{2^n}{\sqrt{n}}} \tag{5.3}$$

for some constants k and c'.

From (5.3), we then see that

$$D_n^{\frac{\sqrt{n}}{(1+c'\frac{\log n}{n})k}} \leq 2^{2^n}.$$

This means that the number of all families of subsets on an n-set is exponential in n relative to the Dedekind number. Hence, the Dedekind number is super-exponential but sub-doubly-exponential in n. Therefore, the approach of simply checking all families of coalitionsis not

[2]This is true because N is a set of n players. Then, the powerset of N is the set of all possible coalitions, and thus the powerset of the powerset of N is the set of all sets of coalitions.

efficient (in the sense that there is an exponential amount of families of colitions that are not antichains).

We will not explore this enumeration problem any further, for the reason that even an efficient enumeration method for antichains would not result in any practically applicable version of Algorithm 2, simply because the Dedekind number is very large.

So, leaving this as it is for now, we conclude that the algorithm achieves a running time in $O^*(2^{2^n} \cdot h(n))$, where $h(n)$ is the time it takes to execute one iteration of the for-loop in Algorithm 2. As long as $h(n)$ is not super-exponential, it holds that $O^*(2^{2^n} \cdot h(n)) \in O^*(2^{2^{n(1+\epsilon)}})$ for any $\epsilon > 0$. Fortunately, all known power indices are known to be computable in at most exponential time. Of course, if the power index can be computed in polynomial time, then we achieve a runtime of $O^*(2^{2^n})$, i.e. we may take 0 for ϵ.

5.2 Weighted Voting Game Design

Having given a simple but very slow algorithm for the PVGD-problem for the very general class of monotone games, we will now see that we can do much better if we restrict the problem to smaller classes. More precisely, we will restrict ourselves to the class of weighted voting games: $\mathcal{G}_{\mathsf{wvg}}$. This class is contained in the class of linear games $\mathcal{G}_{\mathsf{lin}}$, and therefore also contained in the class of monotone games $\mathcal{G}_{\mathsf{mon}}$. For this reason, we can represent a game in $\mathcal{G}_{\mathsf{wvg}}$ using any representation language that we have thus far introduced.

Because weighted voting games are arguably the only class of games that we really actually encounter in the real world, and the algorithm that we will devise has potential to be applied in practice, this section could be considered the main contribution of this thesis. As has been said in Section 3.2, all known research on voting game design problems has focused on this specific variant, $(f, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD, with f being either the Banzhaf index or the Shapley-Shubik index. The methods that have been proposed up until now are all local search methods without an approximation guarantee. Here, we will give an exact algorithm for this problem that runs in exponential time. What will turn out to make this algorithm interesting for practical purposes, is that it can be used as an anytime-algorithm: we can stop execution of this algorithm at any time, but the longer we run it, the closer the answer will be to the optimum. The advantage of this algorithm over the current local search methods is obviously that we will not get stuck in local optima, and it is guaranteed that we eventually find the optimal answer.

5.2.1 Preliminary Considerations

Before proceeding with formally stating the algorithm, let us first consider the question of which approach to take in order to find an exact algorithm for designing weighted voting games.

A possible approach to solve the $(f, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem is to use Algorithm 2 as our basis, and check for each monotone game that we find whether it is a weighted voting game. We do the latter by making use of the Hop-Skip-and-Jump algorithm that we described in the previous chapter. This indeed results in an algorithm that solves the problem, but this algorithm would be highly unsatisfactory: firstly, we noted in the previous section that it is not known how to enumerate antichains efficiently. Secondly, the class of weighted voting games is a subclass of the class of monotone games: in fact, we will see that there are much less weighted voting games than monotone games.

The main problem we face for the $(f, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem is the fact every weighted voting game $G \in \mathcal{G}_{\mathsf{wvg}}$ has an infinite number of weighted representations, i.e. strings in $\mathcal{L}_{\mathsf{weights}}$ that represent G. This is easily seen from Theorem 41: we can multiply the weight vector and the quota with any constant in order to obtain a new weight vector that represents the same game. On top of that, it is also possible to increase or decrease a player's weight by some amount without "changing the game".

For this reason, it will be hard to find an exact algorithm that works internally with the weighted representations of games: an exact algorithm for this problem will probably have to consider many different weighted voting games, but when using the weighted representation of a game, it is hard to know whether or not the algorithm has already considered the same game at an earlier point in time.

All of the local search methods that try to solve the PVGD-problem use the weighted representation internally, but if we want to solve this problem exactly, we will probably have to resort to using other representations. In our case, we will initially be working with $\mathcal{L}_{W,\min}$, the minimal winning coalition listings, because for each weighted voting game there is only one unique listing of minimal winning coalitions, instead of an infinite amount.

5.2.2 A New Structural Property for the Class of Weighted Voting Games

Let us now develop the necessary theory behind the algorithm that we will propose. We will focus only on the canonical class of weighted voting games, since for each non-canonical weighted voting game there is a canonical one that can be obtained by merely permuting the players.

The algorithm we will propose is based on a new structural property that allows us to enumerate the class of canonical weighted voting games efficiently: We will define a new relation $\supseteq_{\mathsf{MWC}}$ and we will prove that for any number of players n the class $\mathcal{G}_{\mathsf{cwvg}}(n)$ forms a graded poset with a least element under this relation.

Definition 67 ($\supseteq_{\mathsf{MWC}}$)**.** Let G_1 and G_2 be any two monotone games. Let $W_{\min,1}$ and $W_{\min,2}$ be their respective sets of minimal winning coalitions. Then, we say that $G_1 \supseteq_{\mathsf{MWC}} G_2$ if and only if $W_{\min,1} \supseteq W_{\min,2}$.

Theorem 68. *For each n, $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ is a graded poset with rank function*

$$\begin{array}{cccc} \rho & : & \mathcal{G}_{\mathsf{wvg}}(n) & \rightarrow & \mathbb{Z} \\ & & G & \mapsto & |W_{\min}(G)|, \end{array}$$

where $W_{\min}(G)$ is the set of minimal winning coalitions of G. Moreover, $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ has a least element of rank 0.

Observe that the theorem above is simply a more formal way of saying: "*Consider an arbitrary weighted voting game of n players, and look at its list of minimal winning coalitions. There is a minimal winning coalition in this list, such that if we remove that* *coalition, we obtain a list of winning coalitions that represents yet another weighted voting game of n players.*"

Proof (Theorem 68). By definition, $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ is a valid poset. In order to prove that the poset is graded under the rank function ρ that is specified in the theorem, we will show by construction that

Lemma 69. *For any game $G \in \mathcal{G}_{\mathsf{cwvg}}(n)$ with a non-empty set $W_{\min}$ as its set of minimal winning coalitions, there is a coalition $C \in W_{\min}$ and a game $G' \in \mathcal{G}_{\mathsf{cwvg}}(n)$ so that it holds that $W_{\min} \backslash C$ is the set of minimal winning coalitions of G'.*

From Lemma 69, the remaining part of the theorem automatically follows: the game with no minimal winning coalitions is the only weighted voting game with rank 0, and clearly is the least element of the poset.

To prove Lemma 69, we first prove the following two preliminary lemmas.

Lemma 70. *Let $G = (N = \{1, \ldots, n\}, v)$ be a weighted voting game, and let $\ell = ((w_1, \ldots, w_n), q)$ be a weighted representation for G. For each player i there exists an $\epsilon > 0$ such that $\ell' = ((w_1, \ldots, w_i + \epsilon', \ldots w_n), q)$ is a weighted representation for G for all $\epsilon' < \epsilon$.*

Informally, what this lemma is telling us, is that it is always possible to increase the weight of a player by some amount without changing the game.

Proof. ℓ is a weighted representation for G, so for each C such that $v(C) = 1$ we have that $w_\ell(C) \geq q$, and for each C such that $v(C) = 0$ we have that $w_\ell(C) < q$. Define L_i as the set of losing coalitions containing player i. Now consider a coalition $C \in L_i$ for which it holds that for all $C' \in L_i : w_\ell(C') \leq w_\ell(C)$.

Because $w_\ell(C) < q$, it must be that $q - w_\ell(C)$ is greater than 0. If we increase w_i in ℓ by a number strictly between 0 and $q - w_\ell(C)$ to obtain ℓ', then clearly no losing coalition in G_ℓ is a winning coalition in $G_{\ell'}$. Moreover, all winning coalitions in G_ℓ are also winning coalitions in $G_{\ell'}$, because we only *increased* the weight of a player, and we did not change the quota. ☐

Lemma 71. *Let $G = (N = \{1, \ldots, n\}, v)$ be a weighted voting game. There exists a weighted representation $\ell \in \mathcal{L}_{\mathsf{weights}}$ such that for all $(C, C') \in (2^N)^2, C \neq C'$ for which $v(C) = v(C') = 1$, it is true that $w_\ell(C) \neq w_\ell(C')$.*

Or, informally stated again: for a weighted voting game there exists a weighted representation such that all winning coalitions have a different weight.

Proof. Let $\ell = ((w_1, \ldots, w_n), q)$ be a weighted representation for G for which there exists a $(C, C') \in (2^N)^2$ with $w_\ell(C) = w_\ell(C')$, $C \neq C'$ and $v(C) = v(C') = 1$. We will show how to obtain an ℓ' from ℓ such that $G_\ell = G_{\ell'}$ and $w_{\ell'}(C) \neq w_{\ell'}(C'')$ for any other coalition $C'' \in N$ with $v(C'') = 1$. This process can then be repeated to obtain a weighted representation for G under which the weights of all winning coalitions differ.

The procedure works as follows: it can be assumed without loss of generality that there is a player i in C but not in C'. By Lemma 70, there is an $\epsilon > 0$ such that $\ell' =$

$((w_1, \ldots, w_i + \epsilon', w_n), q)$ is a weighted representation for G for any $0 < \epsilon' < \epsilon$. ℓ is then a weighted representation with $w_{\ell'}(C) \neq w_{\ell'}(C')$, so this *almost* proves the lemma; we must only make sure the we adjust i's weight in such a way that C's weight does not become equal to any other coalition. This can clearly be done: Consider the set of winning coalitions W_i containing player i, and let $D \in W_i$ be a coalition such that $C \neq C''$ and for all $D' \in W_i \backslash C$, we have $w_\ell(D) < w_\ell(D')$. If we make sure that $0 < \epsilon' < \min\{w_\ell(D) - w_\ell(C), \epsilon\}$, then $w'_\ell(C)$ is clearly different from $w'_\ell(C'')$ for any $C'' \subseteq N$. $w(D) - w(C)$ is a positive number, so this is possible if D exists. If D does not exist, then it suffices to take ϵ' simply strictly between 0 and ϵ. □

Using Lemma 71, we can prove Lemma 69, which establishes Theorem 68.

Proof (Lemma 69). Let $G = (\{1, \ldots, n\}, v)$ be a canonical weighted voting game. Let $W_{\min}$ be its set of minimal winning coalitions and let $\ell = ((w_1, \ldots, w_n), q)$ be a weighted representation for which it holds that all winning coalitions have a different weight. By Lemma 71, such a representation exists. We will construct an ℓ'' from ℓ for which it holds that it is a weighted representation of a canonical weighted voting game with $W_{\min} \backslash C$ as its list of minimal winning coalitions, for some $C \in W_{\min}$.

Let i be the highest-numbered player that is in a coalition in $W_{\min}$ (i.e. i is the least desirable non-dummy player). Let $C \in W_{\min}$ be the minimal winning coalition containing i for which it holds that $\forall C' \in W_{\min} : (C' \neq C \wedge i \in C) \rightarrow w_\ell(C') > w_\ell(C)$. Now obtain $\ell' = ((w_1, \ldots, w_i - (w_\ell(C) - q), \ldots, w_n), q)$. Clearly, $G_{\ell'} = G_\ell = G$ and $w_{\ell'}(C) = q$. Moreover, all minimal winning coalitions in $W_{\min}$ that contain player i have a different weight under ℓ'.

What we will do next is decrease i's weight by an amount that is so small, that the only minimal winning coalition that turns into a losing coalition is C. Note that under ℓ', minimal winning coalition C is still the lightest minimal winning coalition containing i. Let $C' \in W_{\min}$ then be be the second-lightest minimal winning coalition containing i. Clearly, by decreasing i's weight (according to ℓ') by a positive amount smaller than $w_{\ell'}(C') - w_{\ell'}(C)$, coalition C will become a losing coalition and all other minimal winning coalitions will stay winning. □

□

In Figure 5.1, $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$ is depicted graphically. Note that this is *not* precisely the Hasse diagram of the poset $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$ (see the explanation in the caption of this figure; the reason that we do not give the Hasse diagram is because the Hasse diagram is not a very convenient way of representing $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$).

Next, we show that $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ is not a tree for $n \geq 4$. When we will state our algorithm in the next section, it will turn out that this fact makes things significantly more complex.

Theorem 72. *For any $n \geq 4$, $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ is not a tree.*

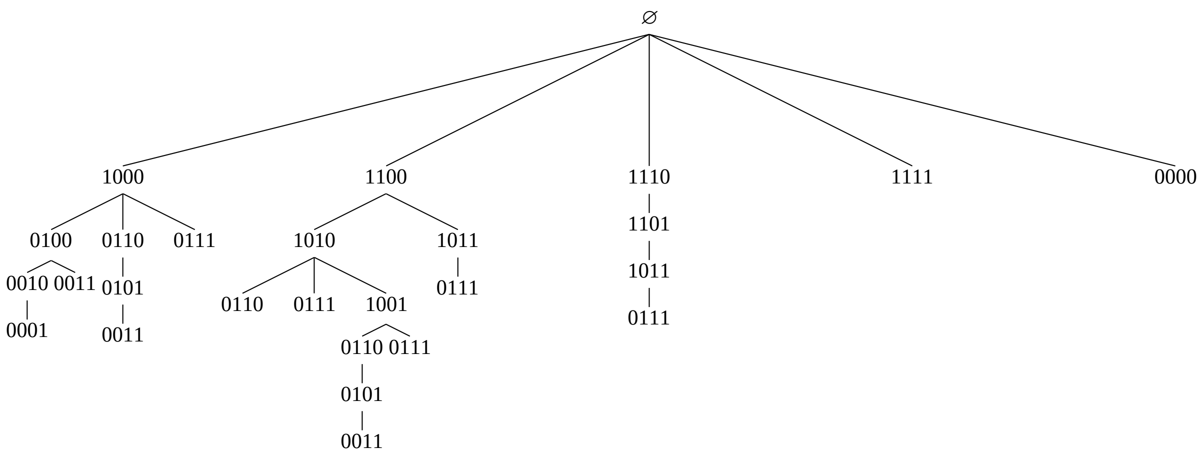

Figure 5.1: Graphical depiction of $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$. Each node in this graph represents a canonical weighted voting game of four players. It should be read as follows: each node has the characteristic vector of a minimal winning coalition as a label. The set of minimal winning coalitions of a game that corresponds to a certain node n in the graph, are those coalitions that are described by the set of vectors that is obtained by traversing the path from the top node to n. The top node corresponds to the canonical weighted voting game with zero minimal winning coalitions (i.e. every coalition loses).

Proof. We will show an example of a game in $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$ for which there are multiple games that cover it. The poset $(\mathcal{G}_{\mathsf{cwvg}}(4), \supseteq_{\mathsf{MWC}})$ is in that case not a tree. For $n > 4$ a similar example is obtained by adding dummy players to the example that we give.

Consider the following weighted representation of a canonical weighted voting game over players $\{1, \dots, 4\}$:

$$\ell = ((3, 2, 2, 1), 4).$$

The set of characteristic vectors $C_{\min,\ell}$ of minimal winning coalitions of G_ℓ is as follows:

$$C_{\min,\ell} = \{1100, 1010, 0110, 1001\}.$$

Next, consider the weighted voting games ℓ' and ℓ'':

$$\begin{aligned} \ell' &= ((3, 1, 1, 1), 4) \\ \ell'' &= ((1, 1, 1, 0), 2), \end{aligned}$$

with respectively the following sets of characteristic vectors of minimal winning coalitions:

$$\begin{aligned} C_{\min,\ell'} &= \{1100, 1010, 1001\}, \\ C_{\min,\ell''} &= \{1100, 1010, 0110\}. \end{aligned}$$

It can be seen that $C_{\min,\ell'} = C_{\min,\ell} \backslash \{0110\}$ and $C_{\min,\ell'} = C_{\min,\ell} \backslash \{1001\}$.

The example we gave is for the set of canonical weighted voting games of four players. This can easily be extended to more than four players: simply add any amount of dummy players to the three example-games. $\square$

5.2.3 The Algorithm

We will use the results from the previous section to develop an exponential-time exact algorithm for $(f, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{W,\min})$-PVGD, and also for $(f, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD. The way this algorithm works is very straightforward: Just as in algorithm 2, we enumerate the complete class of games (weighted voting games in this case), and we compute for each game (that is output by the enumeration algorithm) the distance from the target power index.

Recall that the problem with Algorithm 2 was that the enumeration procedure is not efficient. For the restriction to weighted voting games, we are able to make the enumeration procedure more efficient. We will use Theorem 68 for this: The key is that it is possible to generate the minimal winning coalition listing of canonical weighted games of rank i fairly efficiently from the minimal winning coalition listing of canonical weighted voting games of rank $i-1$.

The following theorem shows us how to do this. To state this theorem, we will first generalize the truncation-operation from Definition 52.

Definition 73 (Right-truncation)**.** Let $S \subseteq N$ be a coalition on players $N = \{1, \dots, n\}$. The *ith right-truncation* of S, denoted $\mathsf{rtrunc}(S, i)$, is defined as

$$\mathsf{rtrunc}(S,i) = \begin{cases} S \backslash \{p_i, \dots n\} \text{ if } 0 < i \leq |S|, \\ S \text{ if } i = 0, \\ \text{ undefined otherwise }, \end{cases}$$

where p_i is the ith greatest-numbered player among the players in S.

Theorem 74. *For any n, let $(G, G') \in \mathcal{G}_{\mathsf{cwvg}}(n)^2$ be a pair of canonical weighted voting games such that G covers G' in $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$. Let $W_{\min,G}$ and $W_{\min,G'}$ be the sets of minimal winning coalitions of G and G' respectively, and let $L_{\max,G}$ and $L_{\max,G'}$ be the sets of maximal losing coalitions of G and G' respectively. There is a $C \in L_{\max,G}$ and an $i \in \mathbb{N}$ with $0 \leq i \leq n$ such that $W_{\min,G'} = W_{\min,G} \cup \mathsf{rtrunc}(C, i)$.*

Proof. Because G covers G', by definition there is a coalition $C' \notin W_{\min,G}$ such that $W_{\min,G'} = W_{\min,G} \cup C'$. Clearly C' can not be a superset of any coalition in $W_{\min,G}$, so it must be a subset of a coalition in $L_{\max,G}$. Suppose for contradiction that C' is not a right-truncation of a maximal losing coalition $C \in L_{\max,G}$. Then there is left shift C'' of C' such that C'' is a subset of a coalition in $L_{\max,G}$, which means that C'' is not a superset of any coalition in $W_{\min,G}$, hence C'' is also not a superset of any coalition in $W_{\min,G'}$. So C'' is a losing coalition in G'. But G' is a canonical weighted voting game, hence G' is also a canonical linear game. By the fact that canonical linear games have the total desirability relation $1 \succeq_D \cdots \succeq_D n$, C'' is a winning coalition in G' because it is a left shift of the winning coalition C'. This is a contradiction. ☐

From Theorem 74, it becomes apparent how to use $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ for enumerating the class of n-player canonical weighted voting games. We start by outputting the n-player weighted voting game with zero minimal winning coalitions. After that, we repeat the following process: generate the $\mathcal{L}_{W,\min}$-representation of all canonical weighted voting games with i minimal winning coalitions, using the set of canonical weighted voting games games with $i-1$ minimal winning coalitions (also represented in $\mathcal{L}_{W,\min}$). Once generated, we have the choice to output the games in their $\mathcal{L}_{W,\min}$-representation or in their $\mathcal{L}_{\mathsf{weights}}$-representation, by using the transformation algorithm presented in the previous chapter.

Generating the set of games of i minimal winning coalitions works as follows: For each game of $i-1$ minimal winning coalitions, we obtain the set of maximal losing coalitions by using the Hop-Skip-and-Jump algorithm presented in the previous chapter. Next, we check for each maximal losing coalition C whether there is a right-truncation of C that we can add to the set of minimal winning coalitions, such that the resulting set is a weighted voting game. Again, testing whether a game is a weighted voting game is done by using the algorithm given in the previous chapter. If a game turns out to be weighted, we can save it and output it.

There is one remaining problem with this approach: It outputs duplicate games. If $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$ were a tree, then this would not be the case, but by Theorem 72 it is not a tree for any $n \geq 4$. Therefore, we have to do a *duplicates-check* for each weighted voting game that we find. We have to check whether we did not already generate it. In principle, this seems not to be so difficult: For each game that we find, sort its list of minimal winning coalitions, and check if this list of coalitions already occurs in the array of listings of minimal winning coalitions that correspond to games that we already found. The problem with this is that the list grows very large, so these checks are then very time and space consuming operations.

We will therefore use a different method for doing this "duplicates-check". We use the following algorithm for this check: Suppose that we have found an n-player canonical weighted voting game G of i minimal winning coalitions by adding a coalition C to a minimal winning coalition listing of a canonical weighted voting game that we have already found. We first sort G's list of minimal winning coalitions. After that, we check for each coalition C' that occurs before C in this sorted list, whether C''s removal from the list results in a list of minimal winning coalitions of a canonical weighted voting game. If there is such a C', then we *do not* output the game G. As one can see, there is only one C such that there is not such a C', so by using this method, it is certain that each canonical weighted voting game will be generated only once.

Algorithm 3 gives the pseudocode for this enumeration method.

Theorem 75. *Algorithm 3 runs in* $O^*(2^{n^2+2n})$ *time.*

Proof. Lines 8 to 18 are executed at most once for every canonical weighted voting game. From Theorem 66 we know that any list of minimal winning coalitions has fewer than $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ elements. So by the runtime of the Hop-Skip-and-Jump algorithm, line 7 runs in time $O(n\binom{n}{\lfloor \frac{n}{2} \rfloor}^2 + n^3\binom{n}{\lfloor \frac{n}{2} \rfloor})) = O(n^2\sqrt{n}2^n)$. Within an iteration of the outer loop (line 4), lines 10 to 13 are executed at most $n\binom{n}{\lfloor \frac{n}{2} \rfloor} = O(\sqrt{n}2^n)$ times (because $L_{\max}$ is also an antichain, so Sperner's theorem also applies for maximal losing coalitions). The time-complexity of one execution of lines 10 to 13 is as follows.

- At line 10 we must solve a linear program, taking time $O(n^{4.5}\binom{n}{\lfloor \frac{n}{2} \rfloor}) = O(n^4 2^n)$ using Karmarkar's interior point algorithm [22].

Algorithm 3 An enumeration algorithm for the class of n player canonical weighted voting games.

```
1: {games[$i$] will contain the list of canonical weighted voting games that have $i$ minimal
   winning coalitions. The games are represented in language $\mathcal{L}_{W,\min}$. games[0] is our
   starting point. First we output the $n$-player canonical weighted voting game with zero
   minimal winning coalitions.}
2: Output $((0,\dots,0),1)$.
3: games[0] := $\{\varnothing\}$
4: for $i := 1$ to $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ do
5:   for all $W_{\min} \in$ games[$i-1$] do
6:     {Obtain the maximal losing coalitions:}
7:     $L_{\max} :=$ hopskipjump($W_{\min}$)
8:     for all $C \in L_{\max}$ do
9:       for $j := 1$ to $n$ do
10:        if isweighted($W_{\min} \cup$ rtrunc($C, i$)) then
11:          if $W_{\min} \cup$ rtrunc($C, i$) passes the duplicates-check (see discussion above)
             then
12:            Output the weighted representation of the voting game with minimal
               winning coalitions $W_{\min} \cup$ rtrunc($C, i$)).
13:            Append $W_{\min} \cup$ rtrunc($C, i$)) to games[$i$].
14:          end if
15:        end if
16:      end for
17:    end for
18:  end for
19: end for
```

- At line 11, we must execute the duplicates-check. This consists of checking for at most $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ sets of minimal winning coalitions whether it is weighted. This involves running the Hop-Skip-and-Jump algorithm, followed by solving a linear program. So in total this takes $O(n^3\sqrt{n}2^{2n})$.

- Lines 12 and 13 clearly take linear time.

Bringing everything together, we see that a single pass through lines 6 to 16 costs us $O(n^4 2^{3n})$ time. As said, these lines are executed at most $|\mathcal{G}_{\mathsf{cwvg}}|(n)$ times. We know that $|\mathcal{G}_{\mathsf{wvg}}(n)| \in O(2^{n^2-n})$ (see previous chapter), and of course $|\mathcal{G}_{\mathsf{cwvg}}(n)| < |\mathcal{G}_{\mathsf{wvg}}(n)|$, so lines 6 to 16 are executed at most $O(2^{n^2-n})$ times, and therefore the runtime of the algorithm is $O(2^{n^2+2n}n^4) = O^*(2^{n^2+2n})$. □

Although the runtime analysis of this algorithm that we gave is not very precise, the main point of interest that we want to emphasize is that this method runs in exponential time, instead of doubly exponential time. We can also show that this algorithm runs in a time that is only polynomially greater than the amount of data output:

Theorem 76. *Algorithm 3 runs in output-polynomial time.*

Proof. Lines 8 to 18 are executed less than $|\mathcal{G}_{\mathsf{cwvg}}(n)|$ times. From the previous chapter, we have as a lower bound that $|\mathcal{G}_{\mathsf{cwvg}}(n)| \in \Omega(2^{n^2(1-\frac{10}{\log n})}/n!2^n)$. One execution of lines 6 to 16 costs $O(n^4 2^{3n})$ time, and thus one iteration takes

$$O(n^4 2^{3n}) \in O(2^{n^2(1-\frac{10}{\log n})}/n!2^n) \in O(|\mathcal{G}_{\mathsf{cwvg}}(n)|)$$

time, resulting in the fact that the algorithm runs in $O(|\mathcal{G}_{\mathsf{cwvg}}(n)|^2)$ time. □

Remark 77. We can not give a very sharp bound on the space complexity of the algorithm, because we do not know anything about the maximum cardinality of an antichain in $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$. However, it can be seen that it is also possible to generate the games in this poset in a depth-first manner, instead of a breadth-first manner like we do now. In that case, the number of space that needs to be used is bounded by the maximum length of a chain in $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$. This is a total amount of $O(\frac{2^n}{\sqrt{n}})$ space.

Now that we have this enumeration algorithm for weighted voting games, we can use the same approach as in algorithm 2 in order to solve the $(f, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem: for each game that is output, we simply compute the power index of that game and check if it is closer to the optimum than the best game we have found up till that point. This works best when f is efficiently computable, as is the case with the Deegan-Packell index (when the input is in $\mathcal{L}_{W,\min}$).

5.3 Improving the Algorithm

Algorithm 3 is in its current state not that suitable for solving the $(f, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem in practice. In this section we will make several improvements to the algorithm, so that it results in an enumeration algorithm of which we expect that it outputs canonical weighted voting games at a steady rate. We will see that this gives us a practically applicable anytime-algorithm for the $(f, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem when $f = \varphi$ or $f = \beta$ (the Banzhaf index or the Shapley-Shubik index).

First, in section 5.3.1, we will see how we can make the system of linear inequalities 4.3 smaller. In section 5.3.2, we will improve Theorem 74 in order to more quickly find new potential minimal winning coalitions to extend our weighted voting games with. Lastly, in Section 5.3.3 we give an output-polynomial time algorithm for enumerating all ceiling coalitions, given a set of roof coalitions.

It is important to see that these three improvements combined, eliminate the need to keep track of the complete lists of minimal winning coalitions and maximal losing coalitions of the weighted voting games that we enumerate. Instead, it suffices to only keep track of the sets of roof coalitions and ceiling coalitions.

5.3.1 An Improved Linear Program for Finding the Weight Vector of a WVG

When finding a weight vector for a weighted voting game of which we obtained the maximal winning coalitions and minimal losing coalitions, we proposed in the previous chapter to do this by solving the system of inequalities (4.3). In [37] it is noted that we can make this system much more compact.

First of all we can reduce the number of inequalities in our system by observing that a minimal winning coalition C which is not a roof, always has a higher total weight than at least one roof, in a canonical weighted voting game. This is because C is a superset of a left shift of some roof. In the same way, a maximal losing coalition which is not a ceiling, always has a lower total weight than at least one ceiling. Therefore if we add the inequalities $w_1 \geq \cdots \geq w_n$ to our system of inequalities (4.3), then we can remove a lot of other inequalities from (4.3) because it then suffices to only make sure that out of all minimal winning coalitions, only the roofs have a higher weight than q; and out of all maximal losing coalitions, only the ceilings have a lower total weight than q.

Secondly, we can reduce the number of variables (weights) in (4.3) by noting that if two players i and $i+1$ are equally desirable, then $w_i = w_{i+1}$. Therefore, we need only one representative variable from each set D of players for which it holds that that

1. the players in D are pairwise equally desirable,

2. and any player in $N \backslash D$ is strictly less or strictly more desirable than a player in D.

By reducing the number of inequalities and variables in this way, we can in most cases drastically decrease the time it takes to find a solution.

5.3.2 A Better Way of Finding New Minimal Winning Coalitions

Theorem 74 allows us to find potential minimal winning coalitions that we can extend our weighted voting games with. We will now see that we do not really need to consider every right-truncation of every maximal losing coalition: In fact, we only need to look at ceiling coalitions.

Theorem 78. *For any n, let $(G, G') \in \mathcal{G}_{\mathsf{wvg}}(n)^2$ be a pair of weighted voting games such that G covers G' in $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$. Let $W_{\min,G}$ and $W_{\min,G'}$ be the sets of minimal winning coalitions of G and G' respectively, and let $L_{\mathsf{ceil},G}$ and $L_{\mathsf{ceil},G'}$ be the sets of ceiling coalitions of G and G' respectively. There is a $C \in L_{\mathsf{ceil},G}$ and an $i \in \mathbb{N}$ with $0 \leq i \leq n$ such that $W_{\min,G'} = W_{\min,G} \cup \mathsf{rtrunc}(C, i)$.*

Proof. Let $L_{\min,G}$ and $L_{\min,G'}$ be the sets of minimal winning coalitions of games G and G' respectively. Because G covers G', by definition there is a coalition $C' \notin W_{\min,G}$ such that $W_{\min,G'} = W_{\min,G} \cup C'$. Suppose for contradiction that C' is not a right-truncation of a ceiling $L_{\mathsf{ceil},G}$. By Theorem 74, C' is a right-truncation of a coalition in $L_{\min,G}$. But then it is a subset of a left shift of a ceiling $C \in L_{\mathsf{ceil},G}$. Then there is left shift C'' of C' such that C'' is a subset of a coalition in $L_{\max,G}$, which means that C'' is not a superset of any coalition in $W_{\min,G}$, hence C'' is also not a superset of any coalition in $W_{\min,G'}$. So C'' is a losing coalition in G'. But G' is a canonical weighted voting game, hence G' is also a canonical linear game. By the fact that canonical linear games have the total desirability relation $1 \succeq_D \cdots \succeq_D n$, C'' is a winning coalition in G' because it is a left shift of the winning coalition C'. This is a contradiction. $\square$

5.3.3 An Output-polynomial Time Algorithm for Obtaining the Ceiling-list from the Roof-list

In section 4.3.2, we derived that the $(\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceil}})$-VGS problem does not have a polynomial time algorithm because the output may be exponentially sized in the input. Nevertheless, it is certainly interesting to try to come up with an as efficient as possible algorithm for this problem, considering that such an algorithm can be used in combination with the improvements of the previous section for finding weight vectors for weighted voting games. If we have a good algorithm for $(\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceil}})$-VGS, then using that algorithm is certainly preferred to using the Hop-Skip-and-Jump algorithm (described in Section 4.3.1), because in a canonical linear game there are always less roof coalitions than minimal winning coalitions, and less ceiling coalitions than maximal losing coalitions.

We will now present an output-polynomial time algorithm for $(\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceil}})$-VGS. The algorithm that we present is based on the following observation.

Definition 79 (i-prefix, prefix)**.** Let S be a coalition on $N = \{1, \dots, n\}$. The i-*prefix* of S is the set $S' = S \cap \{1, \dots, x\}$ where x is the agent such that $|S'| = i$. If for some i a coalition S is an i-prefix of another coalition S', then we say that S is a *prefix* of S'.

Informally, the i-truncation of a coalition s is simply S restricted to the i most desirable players that are in S.

Theorem 80. *Let $G \in \mathcal{G}_{\mathsf{clin}}(n)$ be a canonical linear game on players $N = \{1, \dots, n\}$, let $S \subseteq N$ be a coalition, let a be the least desirable player of S, and let $\mathcal{C}$ be the set of ceilings of G. Then S is a $|S|$-prefix of a ceiling $C \in \mathcal{C}$ if and only if there exists a number $i \geq 0$ such that*

1. *$S \cup \{a+1, \dots, a+i\}$ is winning in G or a ceiling,*
2. *and $S \cup \{n-i+1, \dots, n\}$ is losing in G.*

Proof. ($\Rightarrow$) Let S be a $|S|$-prefix of a ceiling $C \in \mathcal{C}$. In case $S = C$, the proof is trivial. In case $S \neq C$ then the coalition $S \cup \{a+1, \dots, a+|C|-|S|\}$ is either equal to C or a left shift of C (hence winning), and $S \cup \{n-|C|+|S|+1, \dots, n\}$ is either equal to C or a left shift of C (hence losing in both cases).

($\Leftarrow$) Let S be a coalition and let i be a number such that $S \cup \{a+1, \dots, a+1\}$ is winning or a ceiling, and $S \cup \{n-i+1, \dots n\}$ is losing. There are two cases: $S \cup \{a+1, \dots, a+i\}$ is either winning or a ceiling. In the latter case it follows immediately that S is a $|S|$-prefix of a ceiling, namely of S itself, so we assume that $S \cup \{a+1, \dots, a+i\}$ is winning. From the fact that $S \cup \{n-i+1, \dots n\}$ is losing, it follows that there must be a left shift S' of $\{a+1, a+i\}$ such that $S \cup S'$ is losing and all direct left shifts of $S \cup S'$ are winning. $S \cup S'$ can only be a prefix of a ceiling and therefore S is also a prefix of a ceiling. $\square$

The algorithm we will give uses Theorem 80 to find and extend the prefixes of ceiling coalitions. Once it has found the j-prefix of a ceiling $C \in \mathcal{C}(i)$, there are at most n coalitions that can be the (j+1)-prefix of C. Theorem 80 provides us with a fast method to test whether a coalition is a prefix of a ceiling.

The (high level) pseudocode for the algorithm is given in Algorithm 4. Line 3 can be implemented by applying Theorem 80.

Algorithm 4 An algorithm that outputs all ceiling coalitions of a canonical linear game on players $N = \{1, \ldots, n\}$ that is represented as a list of roof coalitions. The input is a string $\ell \in \mathcal{L}_{\mathsf{roof}}$. For any i with $0 \leq i \leq n$, the variable P_i represents the prefixes of cardinality i of ceiling coalitions.

```
1: $P_0 := \{\varnothing\}$ {For any coalition $C$, the empty coalition is a 0-prefix of $C$.}
2: for $i = 1$ to $n$ do
3:     Using $P_{i-1}$, generate all prefixes $S$ of ceilings such that $|S| = i$, and store them in $P_i$
4:     output all ceilings in $P_i$ and remove them from $P_i$.
5: end for
6: return
```

The correctness of this algorithm is obvious. We will now show that it can be implemented to run in output-polynomial time.

Theorem 81. *Let $\ell \in \mathcal{L}_{\mathsf{roof}}(n)$ be a list of roofs of a canonical linear game $G_\ell \in \mathcal{G}_{\mathsf{clin}}(n)$ on players $N = \{1, \ldots, n\}$. Let $\mathcal{C}$ be the set of all ceilings of G_ℓ. On input ℓ, Algorithm 4 runs in time $O(n^3 \cdot |\ell| \cdot |\mathcal{C}|)$ and is hence an output-polynomial time algorithm.*

Proof. Line 3 can be implemented by applying Theorem 80: during iteration j of the for-loop this involves checking for each coalition $C \in P_j$ (let a be C's least desirable player), whether there is an i such that $C \cup \{a+1, \ldots a+i\}$ is winning or a ceiling and $C \cup \{a-i+1, \ldots, i\}$ is losing. P_j is the set of all prefixes of ceiling coalitions, so $|P_j| \leq \mathcal{C}$.

Checking whether a coalition is winning, losing, or a ceiling are easy operations and take time at most $O(n^2 \cdot |\ell|)$: They all require scanning the list of roofs ℓ and checking whether or not the coalition is a left shift of one of the roofs. Checking whether a coalition is a ceiling additionaly requires checking whether all (at most n) direct left shifts are winning. The for loop is executed n times.

Bringing everything together, we end up with a total runtime of $O(n^3 \cdot |\ell| \cdot |\mathcal{C}|)$. □

Chapter 6

Experiments

In this chapter, we will discuss the results obtained from some experiments that we have performed by implementing Algorithm 3, and the algorithm for $(\beta, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD that directly follows from it, where β denotes the normalized Banzhaf index. We think that it is interesting to run our algorithm for some small choices of n to see at what point it becomes intractable, and to obtain some interesting data about the class of canonical weighted voting games (e.g. the number of WVGs on n players). Also, we want to know about the error convergence rate of the algorithm for larger values of n, when solving the problem to optimality is intractable.

So the experiments we perform are related to

- the time-performance of the algorithm (for small n);
- the error-convergence behaviour of the algorithm (for larger n, when it becomes intractable to run the algorithm to completion);
- obtaining the exact number of weighted voting games of n players, in order to compare this to the theoretical bounds;
- obtaining the number of weighted voting games of n players for fixed numbers of minimal winning coalitions;

In Section 6.1 we will give some important information about the implementation itself. Section 6.2 gives a description of our experiments. Lastly, in Section 6.3 we will give the results to these experiments.

6.1 Implementation Details

We have implemented Algorithm 3 together with all of the optimizations described in Section 5.3. The implementation language that we chose is C.

During execution of the algorithm, it is necessary to solve a large amount of linear programs. For doing this, we have chosen to make use of the GNU Linear Programming Toolkit [32]. This is an open-source C library. The GNU Linear Programming Toolkit is not the fastest software that is available for solving linear programs. Most commercial software claims to achieve a performance that is about 10 to 100 times better. However, the reason we chose to use the GNU Linear Program toolkit is because commercial linear programming software is not available to us, and because we think that the GNU Linaer Programming Toolkit allows us to implement our algorithm with relatively the most ease.

As said in the introduction of this chapter, this implementation solves the $(\beta, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem, where β is the *normalized* Banzhaf index. This means that for each weighted voting game that is output by our enumeration algorithm, we must invoke a procedure for computing the normalized Banzhaf index. In our case, we simply use the brute-force approach to compute the normalized Banzhaf index. By using more advanced algorithms for this, it will probably be possible to obtain a large improvement in the runtime of this implementation of the algorithm.

Two variants of the enumeration algorithm have been implemented: The first one uses the standard breadth-first approach, that sequentially generates all weighted voting games of i minimal winning coalitions, for increasing i. The second one uses the depth-first method mentioned in Remark 77 (in Section 5.2.3).

6.2 Experiments

We perform our experiments on a computer with an Intel Core2 Quad Q9300 2.50GHz CPU with 2GB SDRAM Memory. The operating system is Windows Vista. We compiled our source code using gcc 3.4.4, included in the DJGPP C/C++ Development System. We compiled our code with the *–O3* compiler flag.

For doing the experiments, we need input data: instances that we use as input for the algorithm. Instances can in principle be any vector of real numbers. However, normalized Banzhaf indices are always vectors whose elements sum to 1, i.e. normalized Banzhaf indices are members of the unit simplex[1]. So any useful and realistic input for our algorithm is in this case a non-increasing vector in the n-dimensional unit simplex. Our input data therefore consists of samples of such vectors that were taken uniformly at random. We have generated those samples according to the procedure described in [46].

We performed the following experiments.

Experiment 1: For up to 8 players, we measured the CPU time it takes for the enumeration algorithm to output all games, for both the breadth-first and the depth-first method. From these experiments we obtain the exact number of canonical weighted voting games of n players for all n between 1 and 8. We also measure the additional runtime that is necessary when we include the computation of the Banzhaf index into the algorithm.

Experiment 2: We use the enumeration algorithm to compute for all n with $1 \leq n \leq 8$ and all m with $0 \leq m \leq \binom{n}{\lfloor n/2 \rfloor}$, the exact amount of canonical weighted voting games on n players with m minimal winning coalitions.

[1]The definition of *unit simplex* is given in Appendix A

Experiment 3: For n between 1 and 7, we compute for 1000 random instances the *average optimal error*. That is, the average error that is attained out of 1000 random instances (i.e. uniform random vectors in the n-dimensional unit-simplex), when the algorithm is allowed to run to completion on these instances. We also report the worst error that is attained among these 1000 instances. The error function we use is the square root of the sum of squared errors, as stated in Definition 40. The reason that we use this specific error measure is because it has a nice geometric interpretation: it is the Euclidian distance between the target (input) vector and the closest point in the unit simplex that is the normalized Banzhaf index of a weighted voting game. We will therefore denote this error function by *the Euclidian error*.

Experiment 4: For $n \in \{10, 15, 20\}$, we measure the error-convergence behaviour of the algorithm: the Euclidian error as a function of the amount of time that the algorithm runs. We again do this experiment for both the breadth-first and the depth-first version of the algorithm. For each of these three choices of n, we perform this experiment for 10 random instances, and for each instance we allow the algorithm to run for one minute.

6.3 Results

For experiment 1, the runtimes are given in Figure 6.1. Note that this graph is plotted on a log-scale. From the graph we see that for all four versions of this algorithm, there is relative not much difference in the runtime. This means that the inclusion of the Banzhaf index computation procedure does not add a significant amount of additional runtime. One should not forget however, that these results are displayed on a logarithmic scale. When we compare the runtimes for 8 players with each other for example, we see that the runtime of the depth-first search version without Banzhaf index computation is 21 minutes, while it is 26 minutes when we include the computation of the Banzhaf index into the algorithm. When we use the breadth first search approach instead, the runtime is only 16 minutes. In general, the breadth first search method is a lot faster than the depth first search method.

The number of canonical weighted voting games on n players, for $1 \leq n \leq 8$, is displayed in Figure 6.2. Even for these small values of n, we can already clearly see the quadratic curve of the graph on this log-scale, just as the theoretical bounds from Section 4.1 predict. In Table 6.1, we state the number of canonical weighted voting games as exact numbers.

For experiment 2, the results are displayed in Figure 6.3. Note that on the vertical axis we have again a log-scale. We see that for each of these choices of n, most of the canonical weighted voting games have a relatively low amount of minimal winning coaltions relative to the maximum amount of winning coalitions $\binom{n}{\lfloor n/2 \rfloor}$.

The Euclidian errors computed in experiment 3 are displayed in Figure 6.4. We see that the errors decrease as n gets larger. We also see that the worst case optimal error can be much worse than the average case. We want to emphasize that these are results computed over only 1000 random instances, therefore these worst case optimal errors serve only as a lower bound for the worst case optimal error over all possible instances.

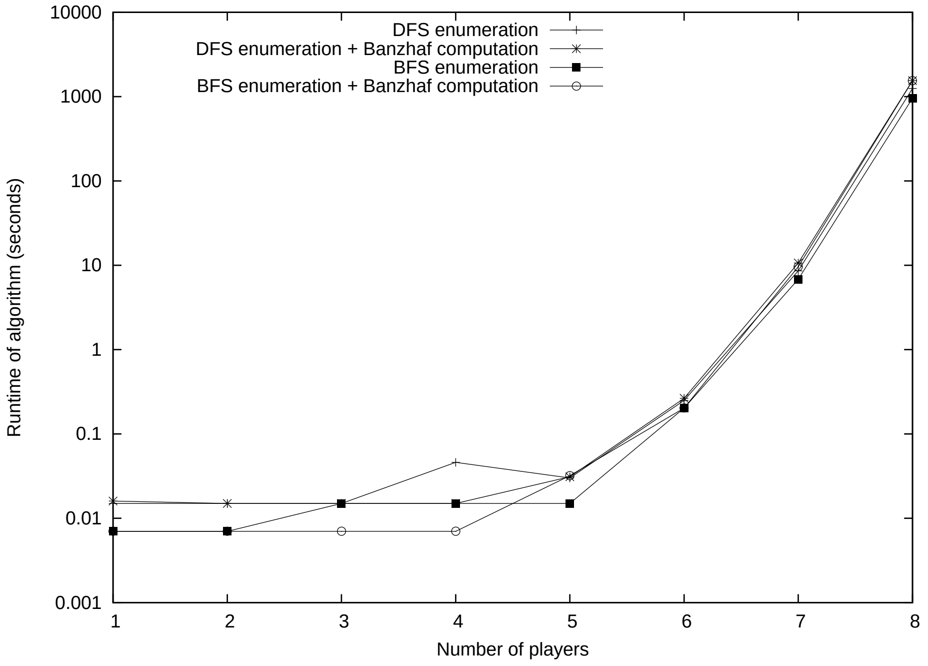

Figure 6.1: Runtimes of Algorithm 3 for 1 to 8 players, for both the breadth first search and the depth first search variant of the algorithm, both with and without the Banzhaf index computation procedure included.

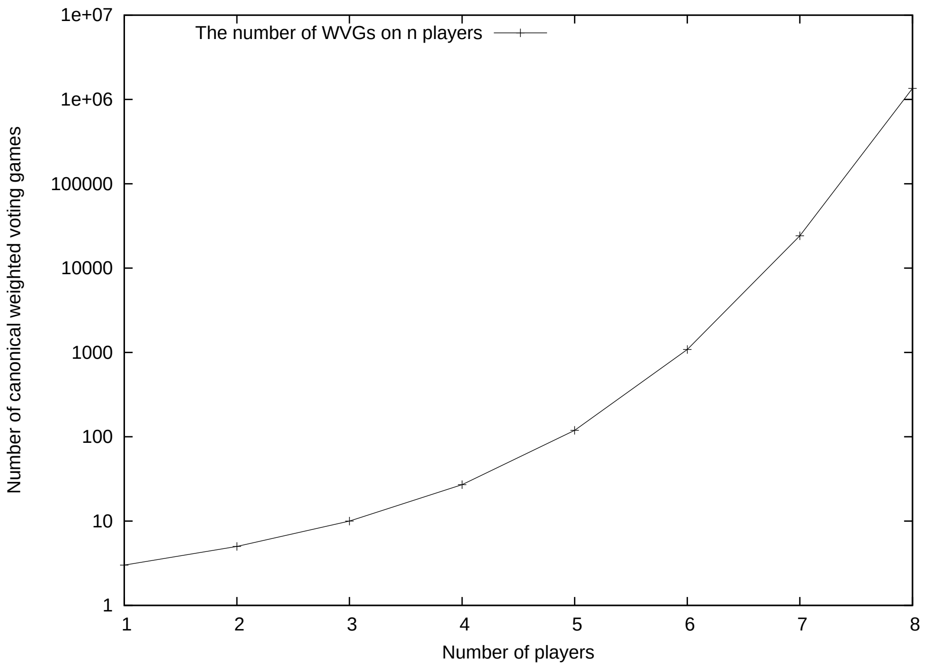

Figure 6.2: The number of canonical weighted voting games on n players, for $1 \leq n \leq 8$.

Table 6.1: Exact values for the number of weighted voting games on n players, for $1 \leq n \leq 8$.

n	$\lvert\mathcal{G}_{\mathsf{cwvg}}(n)\rvert$
1	3
2	5
3	10
4	27
5	119
6	1083
7	24162
8	1353220

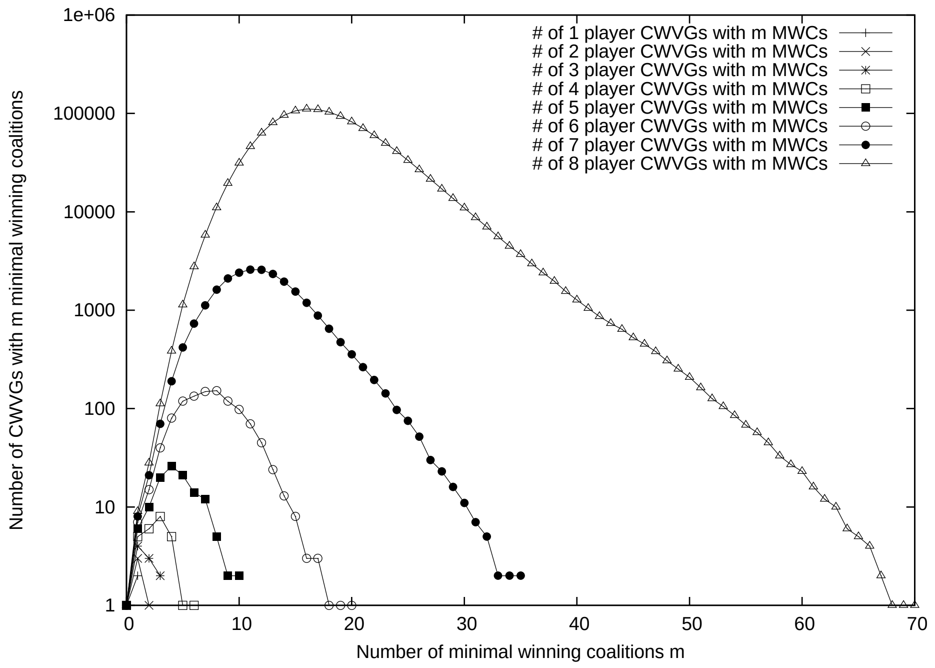

Figure 6.3: The number of canonical weighted voting games (y-axis) on n players, for $1 \leq n \leq 8$, with m minimal winning coalitions (x-axis).

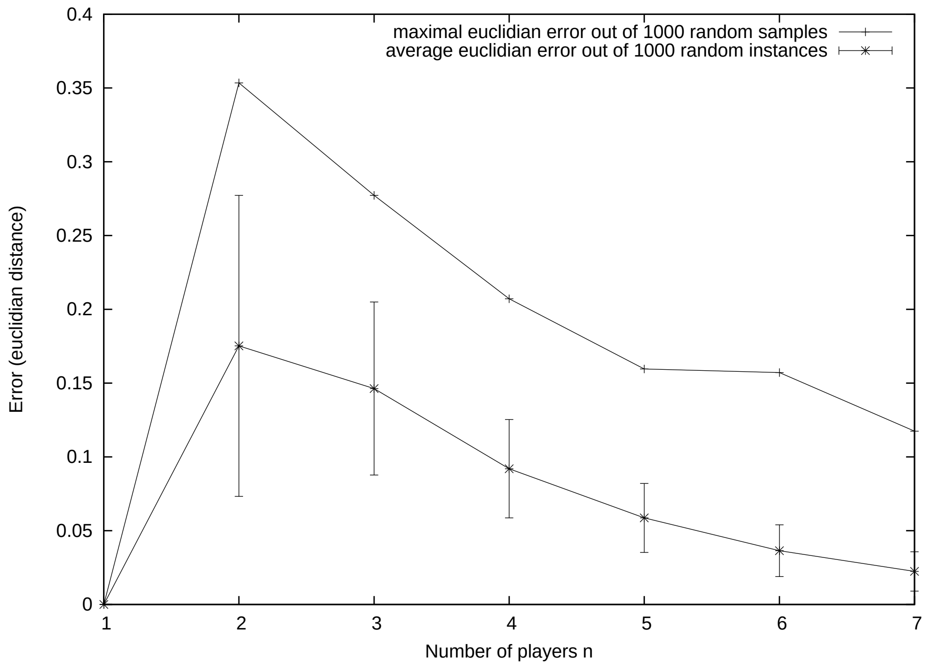

Figure 6.4: Optimal Euclidian error of 1000 random n player instances, for $1 \leq n \leq 7$. The error bars indicate one standard deviation.

For experiment 4, we see no possibility for a meaningful or interesting visualisation of its results. Experiment 4 confirms for us that this enumeration-approach of solving PVGD problems is not so practical for larger values of n. Our hopes were that the anytime-property of the algorithm would account for a quick convergence to a low (but not necessarily optimal) error; even for large values of n. It turns out that this is not really the case. In all cases (i.e. for $n = 10$, $n = 15$ and $n = 20$, for all of the 10 random instances), the error-convergence is high during approximately the first second that the algorithm runs. After that, the frequency by which improvements in the error occur seems to decrease exponentially. Moreover, after the first second, the improvements made are without exception only tiny. The average errors obtained after letting the algorithm run for one minute are as follows:

- For $n = 10$, after one minute, the average error over the 10 instances was 0.055234 for the breadth first variant, and 0.1705204 for the depth first variant.
- For $n = 15$, after one minute, the average error over the 10 instances was 0.0983193 for the breadth first variant, and 0.2018266 for the depth first variant.
- For $n = 20$, after one minute, the average error over the 10 instances was 0.1475115 for the breadth first variant, and 0.2399217 for the depth first variant.

From this, we see that for $n = 10$, the breadth first search method still gives us reasonably nice results within a minute. But when we increase the number of players to 15 and

20, we see that the results quickly get worse; especially when we compare the results to the expected average optimal error (that we obtain by extrapolation of the results of experiment 3).

Another interesting observation is that these errors for the depth first variant are much worse than the errors for the breadth first variant. An explanation for this is that the Banzhaf indices of the games are scattered more evenly across the unit simplex in the case of the breadth first variant. On the contrary, for a small time interval like 1 minute, we expect the depth first variant to enumerate a lot of games for which the Banzhaf indices are close, since many of the games enumerated by the breadth first method cover each other in $(\mathcal{G}_{\mathsf{cwvg}}(n), \supseteq_{\mathsf{MWC}})$.

A final comment we would like to make is that when n gets larger, the output rate of the enumeration algorithm gets slower. Of course, this is explained by the fact that many of the operations in the algorithm must now be performed on games with more players. Especially this slowdown is caused by the computation of the Banzhaf index that is done for every game. In our current implementation, computing the Banzhaf index takes tight exponential time in n.

In general, our current implementation is crude: many procedures in this implementation are still far from optimal. We expect that it is possible to attain a huge improvement in the performance of this algorithm by optimizing the code.

Chapter 7

Discussion & Future Work

In this thesis, we have derived the first *exact* algorithm for solving power index weighted voting game design problems. We have shown that such a problem is always solvable for any class of games, but the guarantee on the worst-case runtime that we can give is unfortunately only doubly exponential. For the important case of weighted voting games, we have derived an anytime method that runs in exponential time, and we have constructed various additional techniques that we can use to speed this algorithm up.

This algorithm is based on an enumeration procedure for the class of weighted voting games: it works by simply enumerating every game, and verifying for each game whether it lies closer to the target power index than the games that we encountered up until that point. For this reason, the algorithm has the anytime property: as we run this algorithm for a longer period of time, the algorithm will enumerate and check more games, so the answer will get better.

Also, because this idea of enumeration is so generic, this method can not only be used to solve power index voting game design problems: in principle it can also be used to solve any other voting game design problem. The only thing we have to adapt is the verification-part of the algorithm (i.e. the part that checks the property in question for each of the games that the enumeration procedure outputs); the enumeration procedure can be left in tact.

Finally, we implemented a simple, non-optimized version of the algorithm in order to do some experiments, and in order to extract some statistical information about the class of weighted voting games. We have computed some exact values for the number of canonical weighted voting games on n players with m minimal winning coalitions, for various choices of n and m. We have seen that even for small n, it is already visible that the number of weighted voting games grows quadratically on an exponential scale, precisely according to the known asymptotic bounds. We have also looked at the runtime of the algorithm, and seen that running the algorithm to completion becomes intractable at approximately $n = 10$ (on the computer that we performed the experiments with, it would take about one day to run the algorithm for $n = 9$). Lastly, for larger values of n we have seen that our algorithm (or at least our current implementation) is not of much use for practical purposes, because the error does not converge as quickly as we would want to. We think that we can attain a huge speedup by optimizing the code, and by using better linear programming software; but in addition to that we will also need to make optimizations in the algorithm in order to overcome this problem.

Note that in most real-life examples, the number of players in a weighted voting game is rather small: usually 10 to 50 agents are involved. For future work, the goal is then of course to get this algorithm to yield good results within a reasonable amount of time, when the number of players is somewhere in this range. There is still a lot of room for improving the proposed algorithm itself, and also the current implementation that we have built.

We think that it will be interesting to study in more depth the partial order that we presented in this thesis, both from a from a computational perspective and from a purely mathematical perspective. With regard to the design of weighted voting games, we suspect that it is possible to prune a lot of "areas" in this partial order, i.e., we think that it is possible to skip the enumeration of a lot of games in the partial order, in many cases.

We are curious about how an algorithm performs that searches through the partial order in a greedy manner, or what will happen if we use some other (possibly heuristic) more intelligent methods to search through the partial order. We wonder whether we can use such a search method while still having an approximation guarantee, or maybe even preserving the guarantee that the optimal game will be found. Lastly, we can also use this idea as a postprocessing step, to extend the existing algorithms by Fatima et al. [16] and Aziz et al. [3], that we mentioned in Section 3.2. With this, we mean that it might be a good idea to first run the algorithm of Fatima or Aziz, in order to obtain a good initial game. Subsequently we can try to search through the neighborhood of the game in the partially ordered set, in order to find improvements.

Lastly, some related questions for which it would be interesting to obtain an answer are about the computational complexity of the power index voting game design problem, and also about the polynomial-time-approximability. The runtime of our current algorithm implies that the problem is in EXPTIME for the case of weighted voting games, but that seems to be all we can say about it. We do not expect the problem to be complete for EXPTIME, but on the other hand, at the moment we do not at all have any ideas on how to prove hardness for this problem for any complexity class whatsoever. It seems very difficult to find a polynomial-time reduction from any known problem that is hard for some complexity class: none of these problem seems to be in any way comparable to our PVGD problem. Also for questions related to approximability of PVGD problems, we currently do not know anything. It might be interesting to look into this question for future research.

Bibliography

[1] N. Alon and P. H. Edelman. The inverse Banzhaf problem. *Social Choice and Welfare*, June 2009.

[2] H. Aziz. Complexity of comparison of influence of players in simple games. In *Proceedings of the 2nd International Workshop on Computational Social Choice (COMSOC-2008)*, pages 61–72, 2008.

[3] H. Aziz, M. Paterson, and D. Leech. Efficient algorithm for designing weighted voting games. In *Proceedings of the IEEE Computer Society, 11th IEEE International Multitopic Conference*, 2007.

[4] J. Bilbao, J. Fernández, A. Losada, and J. López. Generating functions for computing power indices efficiently. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 8(2):191–213, December 2000.

[5] S. F. Brams and P. J. Affuso. Power and size: a new paradox. *Theory and Decision*, 7:29–56, 1976.

[6] J. S. Coleman. Control of collectives and the power of a collectivity to act. *Lieberman, Bernhardt, Social Choice*, pages 192–225, 1971.

[7] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2 edition, 2002.

[8] M. Davis and M. Maschler. The kernel of a cooperative game. *Naval Research Logistics Quarterly*, 12:223–259, 1965.

[9] R. Dedekind. Uber zerlegungen von zahlen durch ihre grssten gemeinsammen teiler. *Gesammelte Werke*, 1:103–148, 1897.

[10] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Oper. Res.*, 19(2):257–266, 1994.

[11] P. Dubey. On the uniqueness of the Shapley value. *International Journal of Game Theory*, 4(3):131–139, September 1975.

[12] P. Dubey and L. S. Shapley. Mathematical properties of the Banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, 1979.

[13] E. Einy. The desirability relation of simple games. *Mathematical Social Sciences*, 10(2):155–168, 1985.

[14] P. Faliszewski and L. Hemaspaandra. The complexity of power-index comparison. In *In Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, pages 177–187. Springer-Verlag Lecture Notes in Computer Science, june 2008.

[15] S. S. Fatima, M. Wooldridge, and N. R. Jennings. A randomized method for the Shapley value for the voting game. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2007)*, pages 955–962, Honolulu, Hawaii, May 2007.

[16] S. S. Fatima, M. Wooldridge, and N. R. Jennings. An anytime approximation method for the inverse Shapley value problem. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2008)*, pages 935–942, Estoril, Portugal, May 2008.

[17] S. S. Fatima, M. Wooldridge, and N. R. Jennings. A linear approximation method for the Shapley value. *Artificial Intelligence*, 172(14):1673–1699, 2008.

[18] J. Freixas, X. Molinero, M. Olsen, and M. J. Serna. The complexity of testing properties of simple games. *CoRR*, abs/0803.0404, 2008.

[19] D. B. Gillies. Solutions to general non-zero-sum games. In A. W. Tucker and R. D. Luce, editors, *Contributions to the Theory of Games IV*, pages 47–85. Princeton University Press, 1959. Annals of Mathematics Studies 40.

[20] J. F. Banzhaf III. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19(2):317–343, winter 1965.

[21] J. Deegan Jr. and E. W. Packel. A new index of power for simple N-person games. *International Journal of Game Theory*, 7(2):113–123, 1979.

[22] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, New York, NY, USA, 1984. ACM.

[23] B. de Keijzer. A survey on the computation of power indices. Technical report, Delft University of Technology, 2009.

[24] D. Kleitman and M. Markowski. On dedekind's problem: The number of isotone boolean functions ii. In *Transactions of the American Mathematical Society*, volume 213, pages 373–390, 1975.

[25] B. Klinz and G. J. Woeginger. Faster algorithms for computing power indices in weighted voting games. *Mathematical Social Sciences*, 49:111–116, 2005.

[26] A. D. Korshunov. Monotone boolean functions. *Russian Mathematical Surveys*, 58(5(353)):198–162, 2003.

[27] M. W. Krentel. The complexity of optimization problems. In *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 69–76, New York, NY, USA, 1986. ACM.

[28] A. Laurelle and M. Widgren. Is the allocation of voting power among EU states fair? *Public Choice*, 94:317–339, 1998.

[29] D. Leech. Computation of power indices. Technical Report 664, Warwick Economic Research Papers, July 2002.

[30] D. Leech. Designing the voting system for the EU council of ministers. *Public Choice*, 113(3–4):437–464, December 2002.

[31] D. Leech. Power indices as an aid to institutional design: the generalised apportionment problem. In M. Holler, H. Kliemt, D. Schmidtchen, and M. Streit, editors, *Yearbook on New Political Economy*. Warwick Economic Research Papers, 2003. Number 648.

[32] A. Makhorin. GNU linear programming toolkit, 2004.

[33] I. Mann and L. S. Shapley. Values of large games, VI: Evaluating the electoral college exactly. Technical Report RM-3158-PR, The RAND Corporation, 1962.

[34] Y. Matsui and T. Matsui. A survey of algorithms for calculating power indices of weighted majority games. *J. Oper. Res. Soc. Japan*, 43:71–86, 2000.

[35] S. Muroga. *Threshold logic and its applications*. Wiley-Interscience, 1971.

[36] A. M. Odlyzko and L. B. Richmond. On the unimodularity of some partition polynomials. *European Journal of Combinatorics*, 3:69–84, 1982.

[37] U. M. Peled and B. Simeone. Polynomial-time algorithms for regular set-covering and threshold synthesis. *Discrete Applied Mathematics*, 12:57–69, 1985.

[38] B. Peleg and P. Sudh¨olter. *Introduction to the Theory of Cooperative Games*. Springer, 2003.

[39] L. S. Penrose. The elementary statistics of majority voting. *Journal of the Royal Statistical Society*, 109:53–57, 1946.

[40] K. Prasad and J. S. Kelly. NP-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19(1):1–9, 1990.

[41] W. H. Riker. *The Theory of Political Coalitions*. Greenwood Press, 1962.

[42] D. Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal of Applied Mathematics*, 17:1163–1170, 1969.

[43] L. S. Shapley. A value for N-person games. *Annals of Mathematics Study*, 28:307–317, 1953.

[44] L. S. Shapley and M. Shubik. A method of evaluating the the distribution of power in a committee system. *American Political Science Review*, 48(3):787–792, 1954.

[45] J. Simon. On some central problems in computational complexity. Technical report, Cornell University, Ithaca, NY, USA, 1975.

[46] N. A. Smith and N. W. Tromble. Sampling uniformly from the unit simplex. Technical report, John Hopkins University, 2004.

[47] E. Sperner. Ein satz über untermengen einer endlichen menge. *Mathematische Zeitschrift*, 27(1):544–548, December 1928.

[48] R. P. Stanley. Weyl groups, the hard Lefschetz theorem, and the Sperner property. *SIAM J. Algebraic Discrete Methods*, 1:168–184, 1980.

[49] M. Sutter. Fair allocation and re-weighting of votes and voting power in the EU before and after the next enlargement. *Journal of Theoretical Politics*, 12:433–449, 2000.

[50] A. D. Taylor and W. S. Zwicker. *Simple Games: Desirability Relations, Trading, Pseudoweightings*. Princeton University Press, 1999.

[51] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[52] V. Zankó. #P-completeness via many-one reductions. *International Journal of Foundations of Computer Science*, 2(1):76–82, 1991.

[53] Y. A. Zuev. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Math. Dokl.*, 39:512–513, 1989.

[54] J. Žunić. On encoding and enumerating threshold functions. *IEEE Transactions on Neural Networks*, 15(2):261–267, march 2004.

Appendix A

Some Preliminary Knowledge

Throughout this thesis, we make use of some results and ideas from mathematics and computer science that the reader might not be familiar with. In this appendix, we will define some notation and introduce some of the essential notions that are needed in order to understand the results. Particularly, we will introduce some order-theoretical concepts.

A.1 Order Theory

All of the following can be looked up in any introductory text on order theory, for example [7].

Definition 82 (Partially ordered set)**.** A *partially ordered set* or *poset* is a pair $(P, \preceq)$, where $\preceq$ is a binary relation over the set P satisfying: $\forall a, b, c \in P$:

- $a \preceq a$ (reflexivity)
- $a \preceq b \land b \preceq a \to a = b$ (antisymmetry)
- $a \preceq b \land b \preceq c \to a \preceq c$ (transitivity)

When $|P|$ is finite, $(P, \preceq)$ is called a *finite poset*.

Definition 83 (Least element)**.** The *least element of a subset* $S \subseteq P$ *of a partially ordered set* $(P, \preceq)$ is the $a \in S$ for which it holds that $\forall b \in S : a \preceq b$. The least element of S may not exist. The least element of S in the case that we take $S = P$ if referred to as the *least element of poset* $(P, \preceq)$. The least element of a poset may not exist.

Definition 84 (Cover)**.** We say that in a poset $(P, \preceq)$, an element $a \in P$ *covers* an element $b \in P$ if and only if $b \prec a$ and $\neg\exists c \in P : b \prec c \prec a$.[1]

Definition 85 (Graded poset & rank function)**.** A poset $(P, \preceq)$ is *graded* if and only if there exists a function $\rho : P \to \mathbb{Z}$ such that $\forall a, b \in P : a$ covers $b \to \rho(a) = \rho(b) + 1$. ρ is called the *rank function*.

[1]For any pair of elements $(a, b) \in P^2$, we have $a \prec b$ if and only if $a \preceq b$ and $a \neq b$.

Definition 86 (Hasse diagram)**.** A *Hasse diagram* of a finite poset $(P, \preceq)$ is a depiction of the graph $(P, \{(a,b) \mid (a,b) \in P^2 \wedge a \text{ covers } b\})$. It is considered a convenient way to visualize finite partially ordered sets.

Definition 87 ((Finite) Tree)**.** A poset $(P, \preceq)$ is a *finite tree* when $(P, \preceq)$ is finite and when the graph depicted by its Hasse diagram is a tree.

Definition 88 (Chain)**.** A *chain* in a poset $(P, \preceq)$ is a set $S \subseteq P$ such that for any $(a,b) \in S^2$ we have $a \preceq b$ or $b \preceq a$.

Definition 89 (Antichain)**.** An *antichain* in a poset $(P, \preceq)$ is a set $S \subseteq P$ such that for any $(a,b) \in S^2$ we have that neither $a \preceq b$ nor $b \preceq a$ holds.

A.2 Some notions from computer science

We assume that the reader is familiar with basic complexity theory and asymptotic analysis of algorithms. In this thesis, we use the following additional concepts that might be less well-known.

The O^*-notation for exponential-time algorithms

Definition 90 (O^*-notation)**.** We say that a function $f(x)$ defined on the real numbers is in $O^*(g(x))$ if and only if there is a polynomial p such that $f(x) \in O(g(x) \cdot p(x))$.

This notation is convenient for functions of superpolynomial growth, in order to make light of polynomial factors. Use of the O^*-notation is justified by the fact that the exponential factor of a function has much more influence on its growth than the polynomial factors.

The class #P

Some of the computational problems that we will encounter are in the complexity class $\#\mathsf{P}$, introduced in [51]:

Definition 91 ($\#\mathsf{P}$)**.** Let $\Sigma = \{0,1\}$. A function $f : \Sigma^* \to \mathbb{N}$ is in $\#\mathsf{P}$ iff there is a polynomial-time Turing machine M such that

$$f(x) = \{y \in \Sigma^* \mid M(x,y) \text{ accepts}\}.$$

Moreover, we require that the size of y is polynomial in x. Any $f \in \#\mathsf{P}$ can be seen as the problem of counting the number of accepting paths in the computation graph of a non-deterministic polynomial time turing machine.

With an *instance* of $f \in \#\mathsf{P}$ we mean an input $\in \Sigma^*$ for f.

A function f is #P-complete if $f \in \#\mathsf{P}$ and for every $g \in \#\mathsf{P}$ there exists a *Cook-reduction* from g to f.

Definition 92 (Cook reduction)**.** Given two problems Π_1 and Π_2, a *Cook reduction* from Π_1 to Π_2 is a Turing machine which solves instances of Π_1 in polynomial time and makes use of an oracle for Π_2.

In this thesis we will use the Cook-reduction definition of #P-completeness, although more stringent definitions of #P-completeness have been proposed in [52, 45, 27].

Output-polynomial time

Definition 93 (Output-polynomial time)**.** An algorithm computing a function $f(x)$ is said to run in *output-polynomial time* if its worst-case running time is bounded by a polynomial in $|x|$ *and* $|f(x)|$.

Often, when talking about algorithms, we would like to find algorithms that run in time polynomial in the input. However, in a lot of cases, we know in advance that the output of the algorithm will be exponentially sized, relative to the input.

For example, this occurs often in the case of *enumeration algorithms*: algorithms that find all solutions to a specific problem, or (more generally) algorithms that enumerate all members of a certain class of objects. Polynomial time algorithms do not exist for a lot of such problems, and in this case it is reasonable to ask for an output polynomial time algorithm, i.e., an algorithm that runs in an amount of time that is only polynomially larger than the amount of data that it outputs.

A.3 Miscellaneous

Stirling's approximation is an expression that is asymptotically equal to the factorial function.

Theorem 94 (Stirling's approximation)**.**

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n .$$

Definition 95 (Unit simplex)**.** For any $n \in \mathbb{N}$, the n *dimensional unit simplex* is defined as $\{(r_1, \ldots, r_n) \in \mathbb{R}^n \mid \sum_{i=1}^n r_i = 1\}$.