Solving Weighted Voting Game Design Problems Optimally: Representations, Synthesis, and Enumeration

Bart de Keijzer, Tomas B. Klos, and Yingqian Zhang

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2012-006-LIS
Publication	May 2012
Number of pages	59
Persistent paper URL	http://hdl.handle.net/1765/32170
Email address corresponding author	yqzhang@ese.eur.nl
Address	Erasmus Research Institute of Management (ERIM) RSM Erasmus University / Erasmus School of Economics Erasmus Universiteit Rotterdam P.O.Box 1738 3000 DR Rotterdam, The Netherlands Phone: + 31 10 408 1182 Fax: + 31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

Bibliographic data and classifications of all the ERIM reports are also available on the ERIM website: www.erim.eur.nl

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

ABSTRACT AND KEYWORDS	
Abstract	We study the inverse power index problem for weighted voting games: the problem of finding a weighted voting game in which the power of the players is as close as possible to a certain target distribution. Our goal is to find algorithms that solve this problem exactly. Thereto, we study various subclasses of simple games, and their associated representation methods. We survey algorithms and impossibility results for the synthesis problem, i.e., converting a representation of a simple game into another representation. We contribute to the synthesis problem by showing that it is impossible to compute in polynomial time the list of ceiling coalitions of a game from its list of roof coalitions, and vice versa. Then, we proceed by studying the problem of enumerating the set of weighted voting games. We present first a naive algorithm for this, running in doubly exponential time. Using our knowledge of the synthesis problem, we then improve on this naive algorithm, and we obtain an enumeration algorithm that runs in quadratic exponential time. Moreover, we show that this algorithm runs in output-polynomial time, making it the best possible enumeration algorithm up to a polynomial factor. Finally, we propose an exact anytime algorithm for the inverse power index problem that runs in exponential time. By the genericity of our approach, our algorithm can be used to find a weighted voting game that optimizes any exponential time computable function. We implement our algorithm for the case of the normalized Banzhaf index, and we perform experiments in order to study performance and error convergence.
Free Keywords	weighted voting games, inverse power index problem, synthesis problem, algorithms
Availability	
Classifications	

Solving Weighted Voting Game Design Problems Optimally: Representations, Synthesis, and Enumeration

Bart de Keijzer* Tomas B. Klos† Yingqian Zhang‡

Abstract

We study the inverse power index problem for weighted voting games: the problem of finding a weighted voting game in which the power of the players is as close as possible to a certain target distribution. Our goal is to find algorithms that solve this problem exactly. Thereto, we study various subclasses of simple games, and their associated representation methods. We survey algorithms and impossibility results for the synthesis problem, i.e., converting a representation of a simple game into another representation.

We contribute to the synthesis problem by showing that it is impossible to compute in polynomial time the list of ceiling coalitions of a game from its list of roof coalitions, and vice versa. Then, we proceed by studying the problem of enumerating the set of weighted voting games. We present first a naive algorithm for this, running in doubly exponential time. Using our knowledge of the synthesis problem, we then improve on this naive algorithm, and we obtain an enumeration algorithm that runs in quadratic exponential time. Moreover, we show that this algorithm runs in output-polynomial time, making it the best possible enumeration algorithm up to a polynomial factor.

Finally, we propose an exact anytime algorithm for the inverse power index problem that runs in exponential time. By the genericity of our approach, our algorithm can be used to find a weighted voting game that optimizes any exponential time computable function. We implement our algorithm for the case of the normalized Banzhaf index, and we perform experiments in order to study performance and error convergence.

Keywords: Weighted voting games, inverse power index problem, synthesis problem, algorithms

*Algorithms, Combinatorics and Optimization; Centrum Wiskunde & Informatica; The Netherlands; Email: keijzer@cwi.nl.

†Algorithmics; Delft University of Technology; The Netherlands; Email: T.B.Klos@tudelft.nl.

‡Department of Econometrics; Erasmus University Rotterdam; The Netherlands; Email: yqzhang@ese.eur.nl

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, a 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 on 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. Throughout the literature, various *power indices* have been proposed: ways to measure a player's influence (or *(a priori) power*) in a voting game. However, computing a power index turns out to be a challenge in many cases.

In this paper, instead of analyzing the power of each agent in a voting game, we investigate the problem that has been referred to as the "inverse problem" and the "generalized apportionment 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 target power index. The power index voting game design problem is an instantiation of a larger class of problems that we will call *voting game design problems*, in which the goal is to find a game G in a class of voting games such that G has a given set of target properties.

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 in the formation of these bodies, such an algorithm may be used to design a voting method that is as fair as possible. Only very little work is known that tries to solve this problem. The existing algorithms are local search methods that do not guarantee an optimal answer. Surprisingly, no algorithm is known for generating an optimal solution. Such an algorithm to solve the inverse problem exactly is the topic of this paper: We are interested in finding a game for which the power index of that game is the *closest possible* to a certain target power index (or one of those games if there are multiple).

It seems that the most straightforward approach to solve the inverse problem would be to simply enumerate *all possible* weighted voting games of n players, 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 target power index. This is precisely what we will do in this paper. Unfortunately, it turns out that enumerating all weighted voting games efficiently is not so straightforward.

The enumeration method that we present in this paper leads to a *generic exponential time exact anytime algorithm* for solving voting game design problems. We implemented our algorithm for the power index voting game design problem where our power index of choice is the *(normalized) Banzhaf index*: one of the two most widely used power indices. Using this implementation, we experimentally study the runtime and error convergence of this algorithm.

1.1 Contributions

This paper is based on the master's thesis of De Keijzer [11]; one of the authors of this manuscript. A shorter discussion of this work has also appeared [12]. We present and discuss the results of [11], and remove various redundancies, imprecisions, typos, and mistakes that were present in [11]. The results we present are thus as follows:

- We provide a general definition of voting game design problems, and show how the power index voting game design problem is one of these problems.
- 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 a new result: 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.[1]
- We present exact algorithms for solving power index voting game design problems: first, a doubly exponential one for the large class of monotonic simple games; and subsequently we show that it is possible to obtain a (singly) exponential algorithm for the important special case of weighted voting games. This can be regarded as the main result of this paper.
  - 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 practically any voting game design problem.

[1] The roof-representation is also known as the *shift-minimal winning coalition* representation [50]. Likewise, the ceiling representation is also known as the *shift-minimal losing coalition* representation.

  - 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. This result is of independent interest from a 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 runtime 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 of a *linear game*, 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 Related work

Although some specific variants of voting game design problems are mentioned sporadically in the literature, not many serious attempts to solve these problems are known to us. The voting game design problem that is usually studied is that of finding a weighted voting game, represented as a weight vector, for which the power index lies as close as possible to a given target power index. This specific version of the voting game design problem is sometimes referred to as *the inverse problem*, and is also the focus of this paper.

We know of only a few papers where the authors propose algorithms for the inverse problem. One of them is by Fatima et al. [18], where the authors present an algorithm for the inverse problem with the Shapley-Shubik index [45] as the power index of choice. 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 [17, 19] by the same authors. For updating the weights, the authors propose two different rules of which they prove that by applying them, the Shapley-Shubik index of each player cannot get worse. Hence, the proposed algorithm is an *anytime* algorithm: it can be terminated at any time, 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 algorithm is by Aziz et al. [5] for the inverse problem with the Banzhaf index as the power index of choice. The algorithm the authors present here resembles that of [18], in that the algorithm repeatedly updates the weight vector in order to 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 estimated 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 [9, 35, 10]. 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 in [18] for which the output may have rational weights). The updating is done by interpolating a best fit curve. This results in a rational 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.

For Aziz's approach, there is no approximation guarantee and the convergence rate is unknown, so it is not certain whether this method is *anytime* just like Fatima's algorithm. Moreover, not much is known about the time complexity and practical performance of this algorithm (one example is presented of this algorithm working on a specific input).

Leech proposes in [30, 33] 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 [31]), and to the board of governors of the International Monetary Fund.

As of writing, the most recent work on the voting game design problem that we know of, is by Kurz [28]. Kurz proposes an exact method for solving the weighted voting game design problem for the Shapley-Shubik index and the Banzhaf index, by using integer linear programming: The set of linear games is taken as the search space, and branch-and-bound techniques (along with various insights about the set of weighted voting games) are used in order to find in this set a weighted voting game with a power index closest to the target. The author does not give a runtime-analysis. However, experiments are performed that show that the algorithm works well for small numbers of players. Besides the algorithm, various precise conjectures are made about which instances have a large optimal error, i.e., which points in the n-dimensional unit simplex are farthest away from the closest power index among the power indices of the set of n-player weighted voting games.

Kurz moreover correctly points out that in the master's thesis of De Keijzer [11] (on which the present paper is based) the numbers of canonical weighted voting games for 6, 7, and 8 players are wrongly stated. After investigation on our part, it turned out that this is due to a bug in the first implementation of the algorithm. In this paper, we correct this mistake and report the numbers of canonical weighted voting games correctly, although these numbers are already

known by now due to the recent paper [27], also by Kurz (see below).

The problem of *enumerating* the set of weighted voting games on a fixed number of players is, as we will see, closely related to the approach that we take for solving the weighted voting game design problem. This enumeration problem has been studied before in a paper by Kurz [27], where the author uses integer programming techniques in order to enumerate all canonical weighted voting games on up to nine players. The author generates integer weighted representations for all of these games and classifies the games that do not have a unique minimum-sum integer weighted representation.

Krohn and Südholter [26] study enumeration of weighted voting games as well as linear games using a combination of order theoretic concepts and linear programming.

Enumeration of threshold functions is a topic closely related to enumeration of weighted voting games, and has been studied in [39, 52, 40].

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 $[0, 1]^n$. As a first step into solving this problem, they prove in [2] a specific result for the case of the Banzhaf index for monotonic simple games.

Also, some applied work has been done on the design of voting games. In two papers, one by Laruelle and Widgrén [29] 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 [5].

1.3 Outline

The paper is divided into seven sections.

Section 2 introduces the required preliminary knowledge and defines some novel concepts. In particular it introduces cooperative games, with an emphasis on simple games (since our paper deals exclusively with simple games). We will also explain the notion of a power index, because a great part of what motivates the results of this paper has to do with a problem related to power indices. We give a definition of one of the most popular power indices (the Banzhaf index), and we briefly discuss algorithms for computing them, as well as the computational complexity of this problem.

We define in Section 3 the main problem of interest that we attempt to solve in this paper: 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 that we call voting game design problems.

Before directly trying to solve the problem introduced in Section 3, we first discuss in Section 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 section, we make some statements about the cardinality of certain classes of simple games.

Some of the results given in Section 4 are necessary for Section 5, where we devise exact algorithms for the power index voting game design problem (our main problem of interest). We first explain a naive approach for the class of monotonic simple games, and after that, improve on this exponentially for the subclass 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 concepts of *roof* and *ceiling* coalitions.

After that, in Section 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 paper in Section 7, with a discussion and some ideas for future work.

2 Preliminaries

In this section, we will discuss some required preliminary definitions and results. Throughout this paper, we assume familiarity with big-O notation and analysis of algorithms. In some parts of this paper, some basic knowledge of computational complexity theory is assumed as well, although these parts are not crucial for understanding the main results presented. We will not cover these topics in this section.

We use some order-theoretic notions throughout various sections of the paper. These are given in the following definition.

Definition 1 ((Graded) poset, cover, rank function, least element)**.** A *poset* or *partially ordered set* is a set S equiped with a partial order $\preceq$, i.e., a pair $(S, \preceq)$. We say that $y \in S$ *covers* $x \in S$ in $(S, \preceq)$ when $x \preceq y$ and there is no $z \in S$ such that $x \preceq z \preceq y$. A poset $(S, \preceq)$ is *graded* when there exists a *rank function* $\rho : S \to \mathbb{N}$ such that for any pair $(x, y) \in S^2$ it is true that $\rho(y) = \rho(x) + 1$ whenever y covers x in the poset $(S, \preceq)$. A *least element* of a poset $(S, \preceq)$ is an element $x \in S$ such that $x \preceq y$ for all $y \in S$.

The remainder of this section on preliminaries will be devoted to the theory of cooperative simple games. A lot of the information in this section can be looked up in an introductory text on cooperative game theory, for example [43] or in Taylor and Zwicker's book on simple games [50]. We start with defining some essential terminology.

Definition 2 (Cooperative (simple) games, grand coalition, characteristic function, monotonicity)**.**

- A *cooperative game* is a pair (N, v), where N is a finite set of *players*; subsets of N are called *coalitions* and $v : 2^N \to \mathbb{R}_{\geq 0}$ is a function mapping coalitions to non-negative real numbers. Intuitively, v describes how much collective payoff a coalition of players can gain when they cooperate.

- N is called the *grand coalition*. v is called the *characteristic function* or *gain function*.

- A *simple game* is a cooperative game (N, v) where the codomain of v is restricted to $\{0, 1\}$. In this context, subsets of N are referred to as *winning coalitions* if $v(S) = 1$, and *losing coalitions* otherwise (i.e., if $v(S) = 0$).

- A cooperative game (N, v) is *monotonic* if and only if $v(S) \leq v(T)$ for all pairs of coalitions $(S, T) \in (2^N)^2$ that satisfy $S \subseteq T$.

Note that in a large body of literature, the additional assumption is made that $v(\emptyset)$ in a cooperative game. For the sake of stating the results of our paper elegantly, we do not make this assumption.

We will often use simply the word *game* to refer to a cooperative game. A cooperative game (N, v) will often be denoted by just v when it is clear what the set of players is. Later, we define various important additional classes of simple games. Since we will be working with these classes extensively, it is convenient to introduce the following notation in order to denote classes of games that are restricted to a fixed number of players n:

Definition 3. 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:

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

Throughout this paper, n will always be the symbol we use to denote the number of players in a cooperative game.

The monotonic simple games are the games that we are concerned with in this paper.

Definition 4 (The class of monotonic simple games)**.** We define $\mathcal{G}_{\mathsf{mon}}$ to be the class of all monotonic simple games.

Some of the definitions in the remainder of this section are taken or adapted from [4] 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 5 (Representations of simple games)**.** Suppose that (N, v) is a simple game. Let $W = \{S : S \in 2^N \wedge v(S) = 1\}$ and $L = \{S : S \in 2^N \wedge v(S) = 0\}$ be its sets of respectively *losing coalitions* and *winning coalitions*. Define $W_{\min} = \{S \in W : (\forall i \in S)v(S \setminus \{i\}) = 0\}$ and $L_{\max} = \{S \in L : (\forall i \in N \setminus S)v(S \cup \{i\}) = 1\}$ as their respective sets of *minimal winning coalitions* and *maximal losing coalitions*. We can describe a simple game in the following forms:

Winning coalition form (N, W) is called the *winning coalition form* of (N, v).

Losing coalition form (N, L) is called the *losing coalition form* of (N, v).

Minimal winning coalition form If (N, v) is monotonic, then $(N, W_{\min})$ is the *minimal winning coalition form* of (N, v). Observe that $W_{\min}$ fully describes v if and only if (N, v) is monotonic.

Maximal losing coalition form If (N, v) is monotonic, then $(N, L_{\max})$ is the *maximal losing coalition form* of (N, v). Observe that $L_{\max}$ fully describes v if and only if (N, v) is monotonic.

Weighted form If there exists a quota $q \in \mathbb{R}_{\geq 0}$ and a weight $w_i \in \mathbb{R}_{\geq 0}$ for each player $i \in N$, such that for each coalition $S \in 2^N$ it holds that $v(S) = 1 \Leftrightarrow \sum_{i \in S} w_i \geq q$, then the vector $w = (q, w_1, \ldots, w_n)$, also written as $[q; w_1, \ldots, w_n]$, is called a weighted form of (N, v). Observe that every game that has a weighted form is also monotonic.

Games that have a weighted form are of our main interest and have a special name:

Definition 6 (Weighted voting games)**.** If a monotonic simple game has a weighted form, then it is called a *weighted voting game*. The class of all weighted voting games is denoted by $\mathcal{G}_{\mathsf{wvg}}$.

It is well known that the class of weighted voting games is strictly contained in the class of monotonic simple games: examples of monotonic simple games that are not weighted are numerous and easily constructed. Later, in Section 4.1, we discuss the cardinalities of these classes with respect to n.

A weighted voting game is an important type of simple game because it has a 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, politics and stockholder companies. An important property of weighted voting games that we will use, is that a weighted representation of such a game is invariant to scaling:

Proposition 1. *Let $G \in \mathcal{G}_{\mathsf{wvg}}(n)$ be a weighted voting game, and let $\ell = [q; w_1, \ldots, w_n]$ be a weighted representation for G. For every $\lambda \in \mathbb{R}^+$, we have that $\ell' = [\lambda q; \lambda w_1, \ldots, \lambda w_n]$ 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 $w_\ell(C) \geq q$:

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

□

We will be using the following notational abuse in the remainder of this paper: Whenever we are discussing a weighted voting game G with players $N = \{1, \ldots, n\}$ and weighted form $[q; w_1, \ldots, w_n]$, we use $w(S)$ as a shorthand for $\sum_{i \in S} w_i$ for any subset S of N.

We next turn our attention to the topic of influence and power in monotonic simple games. For a monotonic simple game, it is possible to define a relation called the *desirability relation* on the players (see [50] and [15]):

Definition 7 (Desirability relation)**.** For a monotonic simple game (N, v), the *desirability relation* $\succeq_v$ is defined by:

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

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

In cases that it is clear which game is meant, we drop the subscript and write $\preceq, \prec, \succeq, \succ, \sim$ instead of $\preceq_v, \prec_v, \succeq_v, \succ_v, \sim_v$.

There exist other notions of desirability, for which different properties hold [15]. In the context of other desirability relations, the desirability relation that we have defined here is refered to as the *individual desirability relation*. Since this is the only desirability relation that we will use in this paper, 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 8 (Linear game)**.** A simple game (N, v) is a *linear game* if and only if it is monotonic, and in (N, v) no pair of players in N is incomparable with respect to $\preceq$. Thus, for a linear game (N, v), $\preceq$ is a total preorder on N. We denote the class of linear games by $\mathcal{G}_{\mathsf{lin}}$.

It is straightforward to see that all weighted voting games are linear: let (N, v) be a weighted voting game where $N = \{1, \ldots, n\}$, and let $[q; w_1, \ldots, w_n]$ be a weighted form of (N, v). Then it holds that $i \preceq j$ when $w_i \leq w_j$. Hence, every pair of players is comparable with respect to $\preceq$.

In fact, the following sequence of strict containments holds: $\mathcal{G}_{\mathsf{wvg}} \subset \mathcal{G}_{\mathsf{lin}} \subset \mathcal{G}_{\mathsf{mon}}$. This brings us to the definition of two special classes of games that will be convenient for use in subsequent sections.

Definition 9 (Canonical weighted voting games & canonical linear games)**.** A linear game (N, v) is a *canonical linear game* whenever $N = \{1, \ldots, n\}$ for some $n \in \mathbb{N}_{>0}$, and the desirability relation $\succeq$ satisfies $1 \succeq 2 \succeq \cdots \succeq n$. When G is also weighted, then G is a *canonical weighted voting game*. The class of canonical linear games is denoted by $\mathcal{G}_{\mathsf{clin}}$, and the class of canonical weighted voting games is denoted by $\mathcal{G}_{\mathsf{cwvg}}$.

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 10 (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 $i \in S$ and $i - 1 \notin S$ with $2 \leq i \leq n$ such that $S' = (S \setminus \{i\}) \cup \{i - 1\}$. A coalition $S' \subseteq N$ is a *left-shift* of S whenever for some $k > 1$ there exists 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 obtained when we replace in the above definition $i - 1$ with $i + 1$ and $i + 1$ with $i - 1$.

For example, coalition $\{1, 3, 5\}$ is a *direct* left-shift of coalition $\{1, 4, 5\}$, and coalition $\{1, 2, 5\}$ is a left-shift of $\{1, 4, 5\}$.

The notions of left-shift and right-shift make sense for canonical linear games and canonical weighted voting games: Because of the specific desirability order that holds in canonical linear games, a left-shift of a winning coalition is always winning in such a game, and a right-shift of a losing coalition is always losing in such a game. This allows us to represent a canonical linear game in one of the following two forms.

Definition 11 (Roof/ceiling coalition/form)**.** Let $(N = \{1, \ldots, n\}, v)$ be a canonical linear game. Also, let $W_{\min}$ be (N, v)'s list of minimal winning coalitions and let $L_{\max}$ be (N, v)'s list of maximal losing 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. The pair (N, W_{roof}) is called the *roof form* of G. A maximal losing coalition $S \in L_{\max}$ is a *ceiling coalition* whenever every left-shift of S is winning. Let W_{ceil} denote the set of all ceiling coalitions of G. The pair (N, W_{ceil}) is called the *ceiling form* of G.

The terminology ("roof" and "ceiling") is taken from [42], although they have also been called *shift-minimal winning coalitions* and *shift-maximal losing coalitions* [50].

Because we will be discussing simple games from a computational perspective, we next introduce the concept of *representation languages* for simple games.

2.1 Representation languages

We have introduced several ways of representing simple games: by the sets of winning and losing coalitions; by the sets of minimal winning coalitions and maximal losing coalitions; by the sets of roof coalitions and ceiling coalitions; and by their weighted representation.

We now make precise the notion of *representing a simple game* by turning 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.

Prior to defining these languages, we need a way of describing coalitions. Coalitions can be described using their *characteristic vector*.

Definition 12. 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

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

A characteristic vector of a coalition in a game of n players is described by n bits.

Definition 13 (Representation Languages)**.** We define the following *representation languages* to represent simple games.

- $\mathcal{L}_W$. Strings $\ell \in L_W$ are lists of characteristic vectors of coalitions. The string 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.
- The languages $\mathcal{L}_{W,\min}$, $\mathcal{L}_L$, $\mathcal{L}_{L,\max}$, $\mathcal{L}_{\mathsf{roof}}$, and $\mathcal{L}_{\mathsf{ceil}}$ are defined in the obvious analogous fashion.
- $\mathcal{L}_{\mathsf{weights}}$. Strings $\ell \in L_{\mathsf{weights}}$ are lists of numbers $\langle q, w_1, \ldots, w_n \rangle$. The string ℓ represents the simple game G if and only if G is a weighted voting game with weighted form $[q; w_1, \ldots, w_n]$.

We will use the following convention: For a representation language $\mathcal{L}$, we denote with $\mathcal{L}(n)$ the set of strings in the language $\mathcal{L}$ that represent games of 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 $\{1, \ldots, n\}$ that is represented by ℓ.

Definition 14. 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 $\mathcal{L}_{\mathsf{weights}}$.

2.2 Power indices

Power indices can be used to measure the amount of influence that a player has in a monotonic simple game. Power indices were originally 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 game. This is easy to see through the following trivial example weighted voting game:

$$[1000; 997, 1, 1, 1].$$

Here, each player is in only one winning coalition: the grand coalition. All players are required to be present in this coalition for it to be winning, and can therefore be said to have the same influence, despite the fact that there is a huge difference between the weights of the first vs. the other three players.

Many proposals have been put forward to answer the question of what constitutes a good definition of power in a voting game. These answers are in the form of *power indices*, which are mathematical formulations for values that try to describe the 'true' influence a player has in a weighted voting game. We refer the reader to [3] for an excellent WWW information resource on power indices.

Power indices try to measure a player's *a priori* power in a voting game. That is, they attempt to objectively measure the influence a player has on the outcome of a voting game, without having any statistical information on which coalitions are likely to form due to the preferences of the 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 we describe below, the assumption is that each coalition will form with equal probability.

While the need for power indices originally arose from studying weighted voting games, all of the power indices that have been devised up till now also make sense for (and are also well-defined for) simple games. 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 this paper we use the *normalized Banzhaf index* as our power index of choice. This power index is used in the experiments discussed in Section 6. However, we will see that for the theoretical part of our work, the particular choice of power index is irrelevant.

Definition 15 (normalized Banzhaf index & raw Banzhaf index)**.** For a monotonic simple game $(N = \{1, \ldots, n\}, v)$, the *normalized Banzhaf index* of (N, v) 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

$$\beta'_i = |\{S \subseteq N \setminus \{i\} : v(S) = 0 \wedge v(S \cup \{i\}) = 1\}|. \tag{1}$$

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

Note that the Banzhaf index of an n-player simple game is always a member of the unit simplex of $\mathbb{R}^n$, i.e., the set $\{x \in \mathbb{R}^n : \sum_{i=1}^{n} x_i = 1\}$.

The problem of computing power indices, and its associated computational complexity, has been widely studied (e.g., in [37, 36, 16, 7, 8, 1, 24, 9, 51, 6, 32, 30, 22]). For a survey of complexity results, exact algorithms and approximation algorithms for computing power indices, see [22]. In general, computing power indices is a hard task, and the case of the normalized Banzhaf index is no exception: Computation of the raw Banzhaf index is known to be #P-complete [44], and the fastest known exponential time algorihm for computing the Banzhaf index is due to Klinz and Woeginger [24]. It achieves a runtime in $O((\sqrt{2})^n \cdot n^2)$.

3 The problem statement

In this section, 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 (or set of requirements) as well as possible. We will focus on the problem of finding games in which the power index of the game is as close as possible to a given target power index.

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 $f : \mathcal{G} \to \mathbb{R}_{\geq 0}$, 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 16 ($(f, \mathcal{G}, \mathcal{L})$-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 $f : \mathcal{G} \to \mathbb{R}^+ \cup \{0\}$ 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 those where f is a function that returns the distance of a game's power index from a certain *target* power index.

Definition 17 ($(g, \mathcal{G}, \mathcal{L})$-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 $g : \mathcal{G} \to \mathbb{R}^n$ is a function that returns a type of power index (e.g., the normalized Banzhaf index) for games in $\mathcal{G}$. Then, the *$(g, \mathcal{G}, \mathcal{L})$-power index voting game design problem* (or *$(g, \mathcal{G}, \mathcal{L})$-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 $G \in \mathcal{G}$,

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

In words, in a $(g, \mathcal{G}, \mathcal{L})$-PVGD problem we must find a 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 and error function f. In this paper we measure the error by means of the Euclidean distance in $\mathbb{R}^n$ between the power index of the game and the target power index. We made this particular choice of f because from intuition it seems like a reasonable error function. In principle, we could also choose f differently. For example, we could take for f any other norm on $\mathbb{R}^n$. For our purpose, the precise choice of f does not really matter, 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, an instance of such a problem is then represented by only a vector $(p_1, \ldots, p_n)$, representing a target power index.

We will focus in this paper on the problem $(\beta, \mathcal{G}_{\mathsf{wvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD, i.e., the problems of finding a weighted voting game in weighted representation, that is as close as possible to a certain target (normalized) Banzhaf index.

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. In this section we will give an overview of transforming representations of simple games into each other. We call these problems *voting game synthesis problems*, inspired by the term *threshold synthesis* used in [42] for finding a weight vector for a so-called *threshold function*, to be defined later in this section.

In Section 4.1, we first find out what we can say about the cardinalities of various classes 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. This is an interesting question in its own right, but we also require it in order to analyze the algorithms for the voting game design problems that we will present in the next sections.

First we will discuss the number of monotonic simple games and linear games on n players. After that, we will also look at the number of weighted voting games on n players.

4.1.1 The number of monotonic simple games and linear games

Let us start off with the cardinality of the class of monotonic simple 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 monotonic simple game G such that the MWCs of G are precisely $W_{\min}$. We see therefore that the number of monotonic simple games on n players is equal to the number 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 $\subseteq$, or: A family of MWCs on the set of players $N = \{1, \ldots, n\}$ is an *antichain* in the poset $(2^N, \subseteq)$. Hence, the number 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 [13]. 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 rather quickly in n: as n gets larger, D_n increases 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 [42]:

Theorem 1. *For large enough n,*

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

Proof. First observe that $|\mathcal{G}_{\mathsf{clin}}(n)|$ is equal to the number of antichains in the poset $(2^N, \preceq_{\mathsf{ssrs}})$, where $\preceq_{\mathsf{ssrs}}$ is defined as follows: for two coalitions $S \subseteq N$ and $S' \subseteq N$, we have $S \preceq_{\mathsf{ssrs}} S'$ if and only if S is a superset of a right-shift of S'. It can be seen that $(2^N, \preceq_{\mathsf{ssrs}})$ is a graded poset, with the following rank function r:

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

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 $N \setminus S$ 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 [41] 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 also an antichain, there must be more than

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

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

4.1.2 The number of weighted voting games

To our knowledge, in the game theory literature there has not been any research on the number of Weighted Voting Games on n players. Fortunately there is a closely related field of research, called *threshold logic* (see for example [38]), that has some relevant results.

Definition 18 (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 $r = (r_0, r_1, \ldots r_n) \in \mathbb{R}^{n+1}$ such that $r_1x_1 + \cdots + r_nx_n \geq r_0$ if and only if $f(x_1, \ldots, x_n) = 1$. We say that r *realizes* f. We denote the set of threshold functions of n variables $\{x_1, \ldots, x_n\}$ by $\mathbf{LT}(n)$.[2]

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 $r_0, r_1, \ldots, r_n$ are allowed to be negative for threshold functions, whereas $q, w_1, \ldots, w_n$, must be non-negative in weighted voting games.

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

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

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

$$|\mathbf{LT}(n)| \geq 2^{n^2(1-\frac{10}{\log n})}. \tag{3}$$

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. Let $\mathbf{LT}^+(n)$ be the set of *non-negative threshold functions* of

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

variables $\{x_1, \ldots, x_n\}$: threshold functions $f \in \mathbf{LT}(n)$ for which there exists a *non-negative weight vector* r that realizes f It is then not hard to see that there is an obvious one-to-one correspondence between the games in $\mathcal{G}_{\mathsf{wvg}}(n)$ and the threshold functions in $\mathbf{LT}^+(n)$, so $|\mathcal{G}_{\mathsf{wvg}}(n)| = |\mathbf{LT}^+(n)|$. An easy upper bound then follows:

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

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

Corollary 2. *For large enough n, it holds that*

$$|\mathcal{G}_{\mathsf{wvg}}(n)| \leq 2^{n^2(1-\frac{10}{\log n})-n-1} \tag{4}$$

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, it follows that $|\mathbf{LT}^+(n)| \geq \frac{|\mathbf{LT}(n)|}{2^{n+1}}$, and thus also $|\mathcal{G}_{\mathsf{wvg}}(n)| \geq |\frac{\mathbf{LT}(n)}{2^{n+1}}|$. Now by using (3) 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 at least lower-bound k, then we would also be able to sharpen the upper bound on the number of weighted voting games.

Our next question is: 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

$$|\mathcal{G}_{\mathsf{cwvg}}(n)| \geq \frac{2^{n^2(1-\frac{10}{\log n})-n-1}}{n!} \tag{5}$$

for large enough 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 19 (Voting game synthesis (VGS) problem)**.** Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two representation languages for (possibly) distinct classes of simple games. 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 $\mathcal{L}_2$,

- 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 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 ($\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceil}}$);
- 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

For our approach to solving the power index voting game design problem for weighted voting games, which we will present in Section 5, it is of central importance 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 [42] by Peled and Simeone. In [42], the problem is stated in terms of *set-covering problems*. Because this algorithm is central to our approach 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 time complexity.

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

Definition 20 (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 21 (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 $N = \{1, \ldots, 5\}$, we have $\{1, 2, 3\} \preceq_{\mathsf{pr}} \{1, 3, 5\}$. The least element of $(2^N, \preceq_{\mathsf{pr}})$ is $\emptyset$ and the greatest element of $(2^N, \preceq_{\mathsf{pr}})$ is N.

Next, we introduce some operations that we can apply to coalitions. For this, the reader should recall definitions 10 and 12.

Definition 22 (fill-up, bottom right-shift, truncation, immediate successor)**.** Let N be the set of players $\{1, \ldots, n\}$ 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: $\mathsf{brs}(S) = S \cup \{b(S) + 1\} \setminus \{b(S)\}$ (undefined if $b(S) = n$).
- The *truncation* of S: $\mathsf{trunc}(S) = S \setminus \{a(S) + 1, \ldots n\}$.
- The *immediate successor* of S:

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

The immediate successor operation is named as such because it denotes 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 23 (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, 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 $w_1, \ldots, w_n$:

$$\begin{aligned} w_1\chi(1,S) + \cdots + w_n\chi(n,S) &\geq q, \forall S \in W_{\min} \\ w_1\chi(1,S) + \cdots + w_n\chi(n,S) &< q, \forall S \in L_{\max} \end{aligned} \tag{6}$$

   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: $[q; w_1, \ldots, w_n]$ 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 [4]. This algorithm decides whether a monotonic simple game represented as a listing of minimal winning coalitions is a linear game, and if so it outputs a strict desirability order[3]. From the strict desirability order, the required permutation directly follows.

The generation of the sorted list of shelters can be done 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 [21] for example. For the linear program of the fourth part of the algorithm we will have to show that its size is bounded by a polynomial in n and the number 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 $q \in \mathbb{R}_{>0}$ follows from Theorem 1.

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 bounded by a polynomial in the number 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 consider all coalitions in the order induced by the PR-lexi-order, and output those coalitions that are maximal losing coalitions. During

[3]With this, we mean that the algorithm outputs a list $\vec{P} = (P_1, \ldots, P_j)$ such that $\{P_1, \ldots, P_j\}$ is a partition of N, where the players of a set in this partition are all 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.

this process, we will be able to skip huge intervals of coalitions in order to achieve a polynomial run-time.

Algorithm 1 The Hop-Skip-and-Jump algorithm. A polynomial-time algorithm that outputs the set of maximal losing coalitions of a monotonic 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 this is a safe assumption.

```
1:  nextshelter := first shelter on the list. {Output the coalition N and stop if
    the list is empty.}
2:  currentcoalition := ∅ {Start with the least coalition, according to the PR-
    lexi-order.}
3:  loop
4:    while currentcoalition ≠ nextshelter \ {b(nextshelter)} do
5:      if n ∉ 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 ∉ 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
```

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

Theorem 2. *Algorithm 1 outputs only maximal losing coalitions.*

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 1. *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, currentcoalition is a bottom right-shift of a shelter, so 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, currentcoalition is a direct successor of a shelter s containing player n, by definition of the direct successor function, 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 currentcoalition is losing when the while-loop is entered. It suffices now to show that 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 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., currentcoalition $= l \cup \{i\}$. Suppose for contradiction that currentcoalition is winning; then $i \in$ nextshelter and $i-1 \in$ nextshelter. It must also be true that $l \subseteq$ nextshelter because otherwise l is a right-shift of nextshelter and therefore winning (the induction hypothesis states that l is losing). Therefore $l =$ nextshelter $\setminus \{b($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.* 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 currentcoalition is losing.

□

From the lemma above, it follows that at line 8, 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 currentcoalition at line 8 is obtained by a series of successive fill-ups starting from the empty coalition, and $n \in$ currentcoalition. This means that currentcoalition $= N$, so currentcoalition is 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, 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 currentcoalition will turn currentcoalition into a winning coalition, because currentcoalition would then become a superset of a left-shift of s. So 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 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 currentcoalition would make it a superset of a left-shift of s, and thus winning. So currentcoalition is maximal.

□

Theorem 3. *Algorithm 1 outputs all maximal losing coalitions.*

Proof. By Theorem 2 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* currentcoalition $= s$. In that case s is losing, following from Lemma 1.

Case 2: *There is a point when the execution of the algorithm has just passed line 8, such that* currentcoalition $\preceq_{\mathsf{pr}} s \preceq_{\mathsf{pr}}$ brs(trunc(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* currentcoalition $\preceq_{\mathsf{pr}} s \preceq_{\mathsf{pr}}$ brs(nextshelter). Here we have that s is either a right-shift of currentcoalition or a left-shift of a superset of nextshelter. In the former case, s is not a maximal losing coalition because it is a right-shift of currentcoalition, and currentcoalition is not a maximal

losing coalition because it is a strict subset of the bottom right-shift of 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 4. *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

Next, we 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, there is a polynomial-time algorithm that checks whether a monotonic simple game given as a list of minimal winning coalitions is linear, 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 completely symmetric.

What also follows now, is that the problems $(\mathcal{L}_{\mathsf{W},\min}, \mathcal{L}_{\mathsf{ceil}})$-VGS can be solved in polynomial time: we first check if the input list of minimal winning coalitions describes a linear game. If so, then the Hop-Skip-and-Jump algorithm of Section 4.3.1 is able to generate in polynomial time a list of maximal losing coalitions from the list of minimal winning coalitions. After that, we

filter from this output list the coalitions that are not ceiling coalitions. The problem $(\mathcal{L}_{\mathsf{L},\max}, \mathcal{L}_{\mathsf{roof}})$-VGS is also solvable in polynomial time by running a "symmetric" version of the Hop-Skip-and-Jump algorithm where we

- permute the players according to the permutation π 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;
- run a version of the Hop-Skip-and-Jump algorithm where losing coalitions are treated as winning coalitions and vice versa.

once the Hop-Skip-and-Jump algorithm is done, we have a list of coalitions. For each C in this list, the coalition $\{\pi^{-1}(i) : i \in C\}$ 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 $(\mathcal{L}_{\mathsf{ceil}}, \mathcal{L}_{\mathsf{roof}})$-VGS solvable in polynomial time? This turns out to not be the case. We will now give a family of examples of canonical linear games 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 $(\mathcal{L}_{\mathsf{roof}}, \mathcal{L}_{\mathsf{ceil}})$-VGS is not solvable in polynomial time.

Let us first define the following specific type of coalition.

Definition 24 ((k,i)-encoding coalition)**.** Let $N = \{1, \ldots, n\}$ be 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

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

For example, $S_{2,2} = \{1, 4, 6, 7\}$, and $S_{5,3} = \{1, 4, 6, 7, 9, 12\}$. We can then define canonical linear games in which the roof coalitions are (k,i)-encoding coalitions.

Definition 25 (i-bit roof game)**.** Let $N = \{1, \ldots, n\}$ be 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 $\{S_{0,i}, \ldots, S_{2^i-1,i}\}$.

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. First let us use the following definitions for convenience.

Definition 26 (Accepting roof set)**.** Let $G \in \mathcal{G}_{\mathsf{clin}}(n)$ be a canonical linear game on players $N = \{1, \ldots, n\}$. Let $C \subseteq N$ be a coalition, let a be a number such that $1 \leq a \leq |C|$, and let $D(C, a)$ be the ath most desirable player in C. The *accepting set of roofs of the ath most desirable player in C*, denoted $A(C, a)$, is the set consisting of those roof coalitions R for which either the ath most desirable player in R is greater than or equal to $D(c, a)$, or $|R| < a$.

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

Proposition 2. *In a canonical linear game, a coalition C is winning if and only if $\bigcap_{a=1}^{|C|} A(C, a) \neq \emptyset$.*

Proof. This lemma is in fact an equivalent statement of the fact that C is winning in a canonical linear game if and only if it is a superset of a left-shift of a roof: if $R \in \bigcap_{a=1}^{|C|} A(C, a)$ then it means that replacing the ath most desirable player in R by the ath most desirable player in C for all a,$1 \leq a \leq R$ would result in a left-shift of R that is a subset of C, so C must be winning.

Conversely, suppose C is winning. Then there must be a roof R that is a right-shift of a subset of C. By removing from C the players with a higher number than $D(C, |R|)$, we obtain a subset C' of C with $|R|$ players. By replacing the ath most desirable player of C by the ath most desirable player of R for $1 \leq a \leq R$, we obtain a right-shift of C that is R. Because in this last step we replaced each player in C' by a higher-numbered player, we get that $R \in \bigcap_{a=1}^{|R|} A(C, a)$. R is also in $\bigcap_{a=|R|+1}^{|C|} A(C, a)$ by definition. □

Using the notion of an accepting roof set, we can prove the following technical lemma. The reader should recall the definition of a *direct* left-shift (Definition 10).

Lemma 2. *Let C be a ceiling of $G_{i-\mathsf{bit}}$ with two or more distinct coalitions that are direct left-shifts of C, 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 $C_1 = C \cup \{p-1\} \setminus \{p\}$ is a direct left-shift of C. Also, let a be the number such that $p = D(C, a)$. Then $p = 2a$.*

Proof. Observe that for all b it holds that every roof R of $G_{i-\mathsf{bit}}$ has either $D(R, b) = 2b - 1$ or $D(R, b) = 2b$. 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 at least two distinct direct left-shifts, so there must be another player p', $p' \neq p$, such that $C_2 = C \cup \{p' - 1\} \setminus \{p'\}$ 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 $\bigcap_a A(C_2, a) = \emptyset$. We see that C_2 is losing, but C_2 is a direct left-shift of C, which is a ceiling, 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 $\bigcap_a A(C, a) = \emptyset$ because C is losing, and therefore $\bigcap_a A(C_1, a) = \emptyset$ so C_1 is losing. C_1 is also winning, because it is a left-shift of ceiling C. This is a contradiction, so $p \geq 2a$.

$p \geq 2a$ and $p \leq 2a$, so $p = 2a$. □

Lemma 3. *In $G_{i-\mathsf{bit}}$, 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 $C \cup \{q-1\} \setminus \{q\}$ 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 $G_{i-\mathsf{bit}}$:

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

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 $A(C, a) \cap A(C, a') = \emptyset$. But then we would be able to apply a direct left-shift on player p'' without turning C into a winning coalition, i.e., $C \cup \{p'' - 1\} \setminus \{p''\}$ 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 the above 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 exist no more than $O(n^5)$ coalitions with exactly two left-shifts, here 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 3. *The game $G_{i-\mathsf{bit}}$ (on $n = 4i$ players) has $O(n^5)$ ceilings.*

We can now conclude that $\{G_{i-\mathsf{bit}} : i \in \mathbb{N}\}$ is an infinite family of examples in which there are exponentially many more roofs than ceilings. Hence, finally we obtain:

Corollary 4. *there is no polynomial time algorithm for $(\mathcal{L}_{\mathsf{ceil}}, \mathcal{L}_{\mathsf{roof}})$-VGS and $(\mathcal{L}_{\mathsf{ceil}}, \mathcal{L}_{\mathsf{roof}})$-VGS.*

4.3.3 Other voting game synthesis problems & summary of complexity results for voting game synthesis

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 1 summarizes all of the results that we have discussed up to now.

First of all, Freixas et al. investigate in [20] the $(\mathcal{L}_1, \mathcal{L}_2)$-VGS problem for $\mathcal{L}_1$ and $\mathcal{L}_2 \in \{\mathcal{L}_W, \mathcal{L}_{W,\min}, \mathcal{L}_L, \mathcal{L}_{L,\max}\}$. Most of their results follow from the discussion above. One of their results that does not, is that $(\mathcal{L}_W, \mathcal{L}_L)$-VGS and $(\mathcal{L}_L, \mathcal{L}_W)$-VGS do not have a polynomial time algorithm. This holds because 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 [20], 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 monotonic simple games that have exponentially many more maximal losing coalitions than minimal winning coalitions. The Hop-Skip-and-Jump algorithm that we described above does actually solve $(\mathcal{L}_{W,\max}, \mathcal{L}_{L,\min})$-VGS in polynomial time, but only 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 number 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 $\binom{n}{\lfloor n/2 \rfloor}$ and $\binom{n}{\lfloor n/2 \rfloor - 1}$ such coalitions (by Sperner's theorem, see Theorem 5). By using Stirling's

approximation, we can see that both these expressions are exponential in n.

- When $\mathcal{L}_2 = \mathcal{L}_{\mathsf{roof}}$ it follows directly from the proof of Theorem 1 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.

That completes our study of the voting game synthesis problem. As said before, table 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:

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

$\mathcal{L}_2 \rightarrow$ $\mathcal{L}_1 \downarrow$	$\mathcal{L}_W$	$\mathcal{L}_{W,\min}$	$\mathcal{L}_L$	$\mathcal{L}_{L,\max}$	$\mathcal{L}_{\mathsf{roof}}$	$\mathcal{L}_{\mathsf{ceil}}$	$\mathcal{L}_{\mathsf{weights}}$
$\mathcal{L}_W$	-	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
$\mathcal{L}_{\mathsf{roof}}$	EXP	EXP	EXP	EXP	-	EXP	?
$\mathcal{L}_{\mathsf{ceil}}$	EXP	EXP	EXP	EXP	EXP	-	?
$\mathcal{L}_{\mathsf{weights}}$	EXP	EXP	EXP	EXP	EXP	?	-

5 Solving the power index voting game design problem

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 weighted voting game (this follows from Proposition 1). This makes it hard to derive an exact algorithm that is based on working with weighted representations alone, since there is no clear finite set of weight vectors that an algorithm can search through. Hence, it seems not that surprising that no algorithm has yet been developed that 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. What follows in this section, is a study of exact algorithms for some 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 monotonic simple 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 weighted voting games 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. From a mathematical point of view, this partial order is interesting in its own right.

Because we will be dealing with exponential algorithms, we make use of the O^*-notation: A function $f : \mathbb{R} \to \mathbb{R}$ is in $O^*(g)$ for some $g : \mathbb{R} \to \mathbb{R}$ if and only if there is a polynomial $p : \mathbb{R} \to \mathbb{R}$ such that $f \in O(g \cdot p)$. This essentially means that we make light of polynomial factors.

5.1 Monotonic simple game design

In this section we will consider the power index voting game design problem for the class of monotonic simple games $\mathcal{G}_{\mathsf{mon}}$. There are four representation languages that can be used for monotonic simple 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: $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_W)$-PVGD, $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_L)$-PVGD, $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD and $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{L,\max})$-PVGD. For g 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 monotonic simple games.

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

An exact algorithm that solves $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD or $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{L,\max})$-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 symmetric. An algorithm for the languages $\mathcal{L}_W$ and $\mathcal{L}_L$ can be obtained by applying the transformation algorithm discussed in the last section.

Algorithm 2 A straightforward algorithm for solving $(g, \mathcal{G}_{\mathsf{mon}}, \mathcal{L}_{W,\min})$-PVGD. The input is a target power index $\vec{p} = (p_1, \ldots, p_n)$. The output is an $\ell \in \mathcal{L}_{W,\min}$ such that $g(G_\ell)$ 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 L_{W,min}.}
2:  besterror := ∞ {besterror is the error of g(G_bestgame) from p, according to
    the sum-of-squared-errors measure.}
3:  for all ℓ ∈ L_{W,min} do
4:     Compute g(G_ℓ) = (g(G_ℓ, 1), ..., g(G_ℓ, n)).
5:     error := Σ_{i=1}^{n} (g(G_ℓ, i) − p_i)^2.
6:     if error < besterror then
7:        bestgame := ℓ
8:        besterror := error
9:     end if
10: end for
11: return bestgame
```

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 number of antichains we need to

enumerate is D_n, the nth Dedekind number. Because the sequence of Dedekind numbers (D_n) quickly grows very large, line 3 is what gives the algorithm a very high time complexity. The following bounds are known [23]:

$$2^{(1+c'\frac{\log n}{n})E_n} \geq D_n \geq 2^{(1+c2^{-n/2})E_n}, \tag{7}$$

where c' and c are constants and E_n is the size of the largest antichain on an n-set.[4] Sperner's theorem [47] tells us the following about E_n:

Theorem 5 (Sperner's theorem)**.**

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

From Sperner's theorem and Stirling's approximation, we get

$$E_n \in \Theta\left(\frac{2^n}{\sqrt{n}}\right). \tag{8}$$

We conclude that D_n is doubly exponential in n. Algorithm 2 therefore 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. The function h is an exponential function for all popular power indices (e.g., the Shapley-Shubik index and the Banzhaf index) ([4, 14, 44]).

Remark 1. *The following is an interesting related problem: Apart from the fact that D_n is very large, we do not even know of an output-polynomial time procedure to enumerate all antichains on a set of n elements: The simplest way to enumerate antichains would be to enumerate each possible family of coalitions, and check if that family is an antichain. Unfortunately, this method does not run in output-polynomial time. In total, there are 2^{2^n} families of coalitions. Substituting the tight bound of (8) into the upper bound of (7), we get*

$$D_n \leq 2^{(1+c'\frac{\log n}{n})k\frac{2^n}{\sqrt{n}}} \tag{9}$$

for some constants k and c'.

Exponentiating both sides of the above inequality by $\frac{\sqrt{n}}{(1+c'\frac{\log n}{n})k}$, we see that

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

This means that the number of families of subsets on an n-set (i.e., the right hand side of the above inequality) is super-polynomial in n relative to the Dedekind number. This enumeration algorithm does thus not run in output-polynomial time, even though the nth Dedekind number and the number of families of coalitions on a set of n elements, are both doubly-exponential. We leave this as an open problem.

[4] Korshunov devised an asymptotically equal expression [25]:

$$D_n \sim 2^{C(n)} e^{c(n)2^{-n/2}+n^2 2^{-n-5}-n2^{-n-4}}$$

with $C(n) = \binom{n}{\lfloor n/2 \rfloor}$ and $c(n) = \binom{n}{\lfloor n/2 \rfloor - 1}$. In [25], this expression is described as the number of monotonic boolean functions, which is equal to the nth Dedekind number.

5.2 Weighted voting game design

Having given a simple but very slow algorithm for the PVGD-problem for the very general class of monotonic simple games, we will now see that we can do much better if we restrict the problem to smaller classes of simple games. 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 monotonic simple 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 introduced.

As has been said in Section 1.2, the known literature on voting game design problems has focused on this specific variant, $(g, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-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 address 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 $(g, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem is to use Algorithm 2 as our basis, and check for each monotonic simple 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 Section 4.3.1. 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 monotonic simple games: in fact, we will see that there are far less weighted voting games than monotonic simple games.

The main problem we face for the $(g, \mathcal{G}_{\mathsf{vwg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem is the fact that 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 Proposition 1: 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 internally works with weighted representations of weighted voting games. By intuition

one would suspect that an exact algorithm for this problem will probably have to consider many different weighted voting games, but when doing so, it is hard for such an algorithm to find out whether or not it has already considered a game at some earlier point in time.

All of the local search methods that try to solve the PVGD-problem do use the weighted representation internally, but if we want to solve this problem exactly, we will 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 a unique representation as a listing of minimal winning coalitions, instead of an infinite number, as is the case for weighted representations.

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 class of *canonical* 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 $\subseteq_{\mathsf{MWC}}$ 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.

We define the $\subseteq_{\mathsf{MWC}}$-relation as follows.

Definition 27 ($\subseteq_{\mathsf{MWC}}$)**.** Let G_1 and G_2 be any two monotonic simple games. Let $W_{\min,1}$ and $W_{\min,2}$ be their respective sets of minimal winning coalitions. Then, we say that $G_1 \subseteq_{\mathsf{MWC}} G_2$ if and only if $W_{\min,1} \subseteq W_{\min,2}$.

The relation $\subseteq_{\mathsf{MWC}}$ on $\mathcal{G}_{\mathsf{cwvg}}(n)$ forms a poset with some interesting properties, as the next theorem tells us. This theorem is crucial for our enumeration algorithm.

Theorem 6. *For each n, $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ is a graded poset with rank function*

$$\begin{array}{rccc} \rho & : & \mathcal{G}_{\mathsf{cwvg}}(n) & \to & \mathbb{N} \\ & & G & \mapsto & |W_{\min}(G)|, \end{array}$$

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

Informally, the theorem says the following: *"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 6). By the properties of the $\subseteq$-relation, $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ 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 prove the following lemma constructively.

Lemma 4. *For every 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 $W_{\min} \setminus \{C\}$ is the set of minimal winning coalitions of G'.*

From Lemma 4, 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 is the least element of the poset.

To prove Lemma 4, we first prove the following two preliminary lemmas.

Lemma 5. *Let $G = (N = \{1, \ldots, n\}, v)$ be a weighted voting game, and let $\ell = [q; w_1, \ldots, w_n]$ be a weighted representation for G. For each player i there exists an $\epsilon > 0$ such that for all $\epsilon' < \epsilon$, the string $\ell' = [q; w_1, \ldots, w_i + \epsilon', \ldots, w_n]$ is also a weighted representation for G.*

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

Proof. The string ℓ is a weighted representation for G, so for each winning coalition D we have that $w_\ell(D) \geq q$, and for each losing coalition C we have that $w_\ell(C) < q$. If all coalitions containing player i are winning, then increasing player i's weight by whatever amount will not change the game: all coalitions containing player i will still be winning, and all coalitions not containing player i are unaffected by the increase in i's weight.

If, on the other hand, there exist one or more losing coalitions containing player i, then define L_i as the set of losing coalitions containing player i. Now consider (one of) the heaviest coalition(s) in L_i: 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 $0 < q - w_\ell(C)$. If we increase w_i in ℓ by a number strictly between 0 and $q - w_\ell(C)$ to obtain ℓ', then no losing coalition in G_ℓ becomes 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 6. *Let $G = (N = \{1, \ldots, n\}, v)$ be a weighted voting game. There exists a weighted representation $\ell \in \mathcal{L}_{\mathsf{weights}}$ for G such that for all $(C, C') \in (2^N)^2$ with $C \neq C'$, for which $v(C) = v(C') = 1$, it holds that $w_\ell(C) \neq w_\ell(C')$.*

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

Proof. Let $\ell = [q; w_1, \ldots, w_n]$ be a weighted representation for G for which there exists a $(C, C') \in (2^N)^2$ with $C \neq C'$ and $v(C) = v(C') = 1$, for which $w_\ell(C) = w_\ell(C')$. 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 coalition $C'' \in 2^N$ other than C 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 w.l.o.g. that there is an agent i in C but not in C'. By Lemma 5, there is an $\epsilon > 0$ such that $\ell' = [q; w_1, \ldots, w_i + \epsilon', \ldots, w_n]$ is a weighted representation for G for any $0 < \epsilon' < \epsilon$. The string 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 that we adjust i's weight in such a way that C's weight does not become equal to any other coalition's weight. This can indeed be done: Consider the set of winning coalitions W_i containing agent i, and let $D \in W_i$ be a coalition such that $C \neq D$ and for all $D' \in W_i \setminus \{C\}$, we have $w_\ell(D) < w_\ell(D')$. If D exists, we make sure that $0 < \epsilon' < \min\{w_\ell(D) - w_\ell(C), \epsilon\}$, and then $w_{\ell'}(C)$ is different from $w_{\ell'}(C'')$ for any $C'' \subseteq N$. $w_\ell(D) - w_\ell(C) > 0$, so this is possible. Otherwise, if D does not exist, then it suffices to simply take ϵ' strictly between 0 and ϵ. □

Using Lemma 6, we can prove Lemma 4, which establishes Theorem 6.

Proof (Lemma 4). Let $G = (\{1, \ldots, n\}, v)$ be a canonical weighted voting game. Let $W_{\min}$ be its set of minimal winning coalitions and let $\ell = [q; w_1, \ldots, w_n]$ be a weighted representation for which it holds that all winning coalitions have a different weight. By Lemma 6, 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 $W_{\min} \setminus \{C\}$ 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 $\forall C' \in W_{\min} : (C' \neq C \wedge i \in C) \rightarrow w_\ell(C') > w_\ell(C)$. Next, define ℓ' as $[q; w_1, \ldots, w_i - (w_\ell(C) - q), \ldots, w_n]$. Now $G_{\ell'} = G_\ell = G$ and $w_{\ell'}(C) = q$. Moreover, all coalitions in $W_{\min}$ that contain player i have a different weight under ℓ'.

We now 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 $C' \in W_{\min}$ be the second-lightest minimal winning coalition containing i. Obtain ℓ'' 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. No new minimal winning coalition is introduced in this process: suppose there would be such a new minimal winning coalition S, then S contains only players that are at least as desirable as i (the other players have weight 0). But then S would also be a minimal winning coalition in the original game $G_{\ell'}$, which is a contradiction.

We conclude that ℓ'' is a weighted representation for G', so $G' \in \mathcal{G}_{\mathsf{cwvg}}(n)$. □

□

In Figure 1, $(\mathcal{G}_{\mathsf{cwvg}}(4), \subseteq_{\mathsf{MWC}})$ is depicted graphically. Note that this is *not* precisely the Hasse diagram of the poset $(\mathcal{G}_{\mathsf{cwvg}}(4), \subseteq_{\mathsf{MWC}})$ (see the explanation

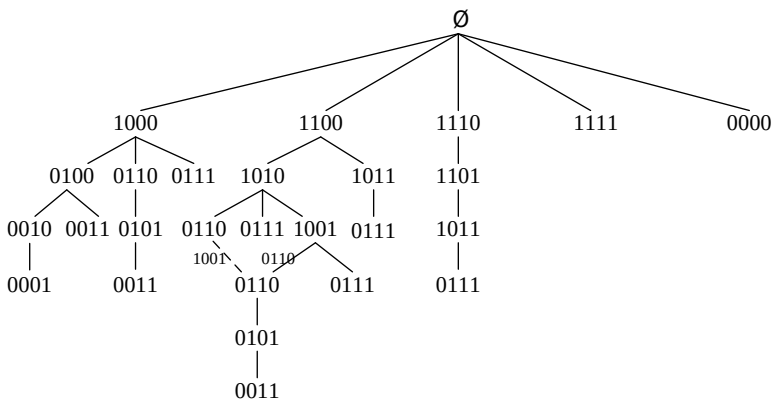

Figure 1: Graphical depiction of $(\mathcal{G}_{\mathsf{cwvg}}(4), \subseteq_{\mathsf{MWC}})$. Each node in this graph represents a canonical weighted voting game of four players. It should be read as follows: each node has the characteristic vector of a minimal winning coalition as a label. The set 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 V_n of vectors that is obtained by traversing the path from the top node to n along the solid edges. The top node corresponds to the canonical weighted voting game with zero minimal winning coalitions (i.e., every coalition loses). The actual Hasse diagram of this poset can be obtained by changing the label of each node n to V_n and including the solid edges as well as the dashed edge in the diagram.

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 $(\mathcal{G}_{\mathsf{cwvg}}(4), \subseteq_{\mathsf{MWC}}))$.

Next, we show that $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ 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 complicated.

Proposition 3. *For $n \geq 4$, $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ is not a tree.*

Proof. We will give an example of a game in $(\mathcal{G}_{\mathsf{cwvg}}(4), \subseteq_{\mathsf{MWC}})$ that covers multiple games.[5] A similar example for $n > 4$ is obtained by adding dummy players to the example that we give here.

Consider the following weighted representation of a canonical weighted voting game over players $\{1, 2, 3, 4\}$:

$$\ell = [4; 3, 2, 2, 1].$$

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

[5] In fact, from inspecting Figure 1 and the explanation given in its caption, it may already be rather obvious to the reader which example game we intend.

as follows:

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

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

$$\begin{aligned} \ell' &= [4; 3, 1, 1, 1] \\ \ell'' &= [2; 1, 1, 1, 0], \end{aligned}$$

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 $C_{\min,\ell'} = C_{\min,\ell} \setminus \{0110\}$ and $C_{\min,\ell''} = C_{\min,\ell} \setminus \{1001\}$. □

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 6 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 22.

Definition 28 (Right-truncation)**.** Let $S \subseteq N$ be a coalition on players $N = \{1, \ldots, n\}$. The *ith right-truncation* of S, denoted $\mathsf{rtrunc}(S, i)$, is defined as

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

where $P(S, i)$ is the ith least desirable player among the players in S.

In effect, the ith right truncation of a coalition S (for $i \leq |S|$) is the coalition that remains when the i least desirable players are removed from S.

Theorem 7. *For any n, let $(G, G') \in \mathcal{G}_{\mathsf{cwvg}}(n)^2$ be a pair of canonical weighted voting games such that G is covered by G' in $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$. 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 is covered by G', by definition there is a coalition $C' \notin W_{\min,G}$ such that $W_{\min,G'} = W_{\min,G} \cup \{C'\}$. Coalition C' can not be a superset of a coalition in $W_{\min,G}$, because then it would not be a *minimal* winning coalition in G'. Therefore, C' is a losing coalition in G, and thus 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 $C \in L_{\max,G}$. So C' is a subset of C, but not a right truncation: This would mean that in C', some player j from C is not present, while at least one less desirable player $k > j$ from C does remain in C'. Then there is a left-shift C'' of C' such that C'' is a subset of a coalition in $L_{\max,G}$: In C'', the less desirable player k can be replaced by player j removed from C. This means that C'' is a losing coalition in G, and is thus not a superset of any coalition in $W_{\min,G}$, and 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, and hence also a canonical linear game. By the fact that canonical linear games have the 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 7, it becomes apparent how to use $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ 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 Hop-Skip-and-Jump algorithm presented in Section 4.3.1.

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. 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 represents a weighted voting game. Again, testing whether a game is a weighted voting game is done by using the Hop-Skip-and-Jump algorithm. If a game turns out to be weighted, we can store it and output it.

There is one remaining problem with this approach: It outputs duplicate games. If $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$ were a tree, then this would not be the case, but by Proposition 3 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 can grow 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". 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 discard G, and otherwise we keep it. This way, it is certain that each canonical weighted voting game will be generated only once.

Algorithm 3 gives the pseudocode for this enumeration method. Correctness

Algorithm 3 An enumeration algorithm for the class of n player canonical weighted voting games. hopskipjump refers to the Hop-Skip-and-Jump algorithm, see Algorithm 1.

1: {games$[i]$ will be the list of canonical weighted voting games that have i minimal winning coalitions. The value of i can not exceed $\frac{n}{\lfloor n/2 \rfloor}$ by Theorem 5. The games are represented in language $\mathcal{L}_{W,\min}$. The game games$[0]$ is our starting point. First we output the n-player canonical weighted voting game with zero minimal winning coalitions.}
2: **Output** $[1; 0, \ldots, 0]$.
3: games$[0] := \{\emptyset\}$
4: **for** $i := 1$ to $\binom{n}{\lfloor n/2 \rfloor}$ **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** $j := 0$ to n **do**
10: **if** isweighted$(W_{\min} \cup$ rtrunc$(C, j))$ **then**
11: **if** $W_{\min} \cup$rtrunc(C, j) 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$(C, j))$.
13: **Append** $W_{\min} \cup$ rtrunc$(C, j))$ to games$[i]$.
14: **end if**
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: **end for**

of the algorithm follows from our discussion above. We will now analyze the time-complexity of the algorithm.

Theorem 8. *Algorithm 3 runs in $O^*(2^{n^2+2n})$ time.*

Proof. Lines 5 to 18 are executed at most once for every canonical weighted voting game. From Theorem 5 we know that any list of minimal winning coalitions has fewer than $\binom{n}{\lfloor n/2 \rfloor}$ elements. So by the runtime of the Hop-Skip-and-Jump algorithm, line 7 runs in time $O\left(n\binom{n}{\lfloor n/2 \rfloor}^2 + n^3\binom{n}{\lfloor n/2 \rfloor}\right) = O(n^2\sqrt{n}2^n)$. Within an iteration of the outer loop (line 4), lines 10 to 15 are executed at most $n\binom{n}{\lfloor n/2 \rfloor} = O(\sqrt{n}2^n)$ 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 15 is as follows.

- At line 10 we must solve a linear program, taking time $O\left(n^{4.5}\binom{n}{\lfloor n/2 \rfloor}\right) = O(n^4 2^n)$ using Karmarkar's interior point algorithm [21].
- At line 11, we must execute the duplicates-check. This consists of checking for at most $\binom{n}{\lfloor n/2 \rfloor}$ 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 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 Corollary 2 in the previous section), and of course $|\mathcal{G}_{\mathsf{cwvg}}(n)| < |\mathcal{G}_{\mathsf{wvg}}(n)|$, so lines 6 to 17 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 an amount of time that is only polynomially greater than the amount of data output. This implies that Algorithm 3 is essentially the fastest possible enumeration algorithm for canonical weighted voting games, up to a polynomial factor.

Theorem 9. *Algorithm 3 runs in output-polynomial time, i.e., a polynomial in the number of bits that Algorithm 3 outputs.*

Proof. Lines 5 to 18 are executed less than $|\mathcal{G}_{\mathsf{cwvg}}(n)|$ times. From 5, 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

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

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

Remark 2. *We can not give a very sharp bound on the space complexity of Algorithm 3, because we do not know anything about the maximum cardinality of an antichain in* $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$*. 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* $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$*. 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.

5.3 Improvements and optimizations

Algorithm 3 is in its current state not that suitable for solving the $(f, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem in practice. In this section we will make several improvements to the algorithm. This results in a version of the 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 $(\beta, \mathcal{G}_{\mathsf{cwvg}}, \mathcal{L}_{\mathsf{weights}})$-PVGD problem for small numbers of players.

Section 5.3.1 shows how we can make the system of linear inequalities (6) smaller. In Section 5.3.2, we will improve Theorem 7 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 note 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 weighted voting game

When finding a weight vector for a weighted voting game of which we obtained the minimal winning coalitions and maximal losing coalitions, we proposed in the previous section to do this by solving the system of inequalities (6). In [42] it is noted that we can make this system much more compact, as follows.

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, adding the inequalities $w_1 \geq \cdots \geq w_n$ to our system of inequalities (6) allows us to remove a lot of other inequalities from (6), because it now 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 (6) 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, and
2. any player in $N \setminus D$ 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 to (6).

5.3.2 A better way of finding new minimal winning coalitions

Theorem 7 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 10. *For any n, let $(G, G') \in \mathcal{G}_{\mathsf{wvg}}(n)^2$ be a pair of weighted voting games such that G is covered by G' in $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$. 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 $W_{\min,G}$ and $W_{\min,G'}$ be the sets of minimal winning coalitions of games G and G' respectively. Because G is covered by G', by definition there is a coalition $C \notin W_{\min,G}$ such that $W_{\min,G'} = W_{\min,G} \cup C$. By Theorem 7, C is a right-truncation of a coalition in $L_{\max,G}$. Suppose for contradiction that C is not a right-truncation of a ceiling in $L_{\mathsf{ceil},G}$. Then there is a ceiling $C' \in L_{\mathsf{ceil},G}$ such that C is a subset of a right-shift of C', and there is a left-shift C'' of C, $C'' \neq C$, such that C'' is also a subset of a right-shift of C'. Coalition C'' is not a superset of $W_{\min,G}$ because C' is losing in G, and C'' is not a superset of C either, because C'' is a left-shift of C and is unequal to C. So it follows that *C'' is a losing coalition in G'*.

But G' is a canonical weighted voting game, so the desirability relation $1 \succeq_D \cdots \succeq_D n$ is satisfied. Because C'' is a left-shift of C, and C is winning in G', it follows that *C'' is a winning coalition in G'*. This is a contradiction. □

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 an 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 fewer roof coalitions than minimal winning coalitions, and fewer 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. For this, we need the notion of a *prefix*.

Definition 29 (i-prefix, prefix)**.** Let S be a coalition on $N = \{1, \ldots, n\}$. The *i-prefix* of S is the set $S' = S \cap \{1, \ldots, x\}$ 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'.

The algorithm that we present is based on the following observation.

Theorem 11. *Let $G \in \mathcal{G}_{\mathsf{clin}}(n)$ be a canonical linear game on players $N = \{1, \ldots, n\}$, 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. *$S \cup \{a+1, \ldots, a+i\}$ is winning in G or a ceiling,*
2. *and $S \cup \{n-i+1, \ldots, n\}$ 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 $S \cup \{a+1, \ldots, a+|C|-|S|\}$ is either equal to C or a left-shift of C (hence winning), and $S \cup \{n-|C|+|S|+1, \ldots, n\}$ 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 $S \cup \{a+1, \ldots, a+1\}$ is winning or a ceiling, and $S \cup \{n-i+1, \ldots n\}$ is losing. There are two cases: $S \cup \{a+1, \ldots, a+i\}$ 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 $S \cup \{a+1, \ldots, a+i\}$ is winning. From the fact that $S \cup \{n-i+1, \ldots n\}$ 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. □

The algorithm we will give, uses Theorem 11 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 11 provides us with a fast method to test whether a coalition is a prefix of a ceiling.

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: $P_0 := \{\emptyset\}$ {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 (high level) pseudocode for the algorithm is given in Algorithm 4. The correctness of this algorithm is obvious. We will now show that it can be implemented to run in output-polynomial time.

Theorem 12. *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 11: 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 $O(n^3 \cdot |\ell| \cdot |\mathcal{C}|)$. □

6 Experiments

Before we turn to the results of some relatively large scale experiments, let us visualize the results for just $n = 3$ players, because we can easily depict these in two dimensions. Figure 2 shows the 3-player simplex, with the vertices labeled by the player numbers. Because we focus on canonical weighted voting games, only the shaded part of the simplex contains games. These games are represented as dark dots. There are four dots, but, as we shall see, there are ten 3-player games. Two of these are degenerate, namely the game with no winning coalitions, and the game in which the empty set is the minimal winning

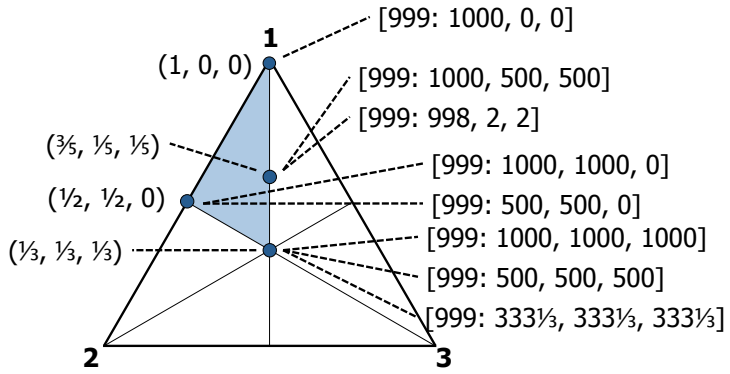

Figure 2: The games on three players, and their power indices.

coalition, so all coalitions are winning. Weighted representations for the other eight games are given on the right in the figure. There are only four distinct power indices corresponding to these games, they are indicated on the left.

In the remainder of this section, 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). There are various reasons for performing these experiments: First of all, we are interested in running our algorithm for some small choices of n to see at what point our algorithm becomes intractable. A second goal of these experiments is to obtain some interesting statistics about the class of canonical weighted voting games (e.g., the number of weighted voting games on n players). Thirdly, it we are interested in obtaining some statistics on the average optimal attainable error on a random instance, when we let the algorithm run to completion for small n. Lastly, 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. More precisely, we want to gain insight in the following:

- the practical time-performance of the algorithm (for small n);
- the average optimal attainable error on random instances (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 for fixed numbers of players, as a function of the number of minimal winning coalitions.

In Section 6.1 we give some important information about the implementation of our algorithm. Section 6.2 describes our experiments. Lastly, in Section 6.3 we present the results of the experiments.

6.1 Implementation details

We have implemented Algorithm 3 together with all of the optimization tricks described in Section 5.3. The programming language that we used is *C*.

Execution of the algorithm encompasses solving a large number of linear programs. For doing this, we make use of the *GNU Linear Programming Toolkit* [34]. This is an open-source C library.

As said in the introduction of this section, our 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. The algorithm we use for this is simply the naive brute-force approach.

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 2 (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. An instance is a target banzhaf index for a canonical weighted voting game, i.e., a point p in the unit simplex such that $p_i \geq p_j$ if $i < j$, for all i, j between 1 and n. Our instances therefore consist of samples of such vectors that were taken uniformly at random. These samples are generated according to the procedure described in [46].

The experiments are as follows:

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 in 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 number of canonical

weighted voting games on n players with m minimal winning coalitions.

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 17. The reason for using this specific error measure is because it has a nice geometric interpretation: it is the Euclidean distance between the target (input) vector and the closest point in the unit simplex that is a normalized Banzhaf index of a weighted voting game.

Experiment 4: For $n \in \{10, 15, 20\}$, we measure the error-convergence behaviour of the algorithm: the Euclidean 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 3. From the graph we see that for all four versions of the algorithm, there is relatively not much difference in the runtimes. This means that the inclusion of the Banzhaf index computation procedure does not add a significant amount of additional runtime. Nonetheless, one should not forget 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 4. 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 2, we state the exact numbers of canonical weighted voting games on n players as numbers, for $1 \leq n \leq 8$.

For Experiment 2, the results are displayed in Figure 5. 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 number of minimal winning coaltions relative to the maximum number of winning coalitions $\binom{n}{\lfloor n/2 \rfloor}$.

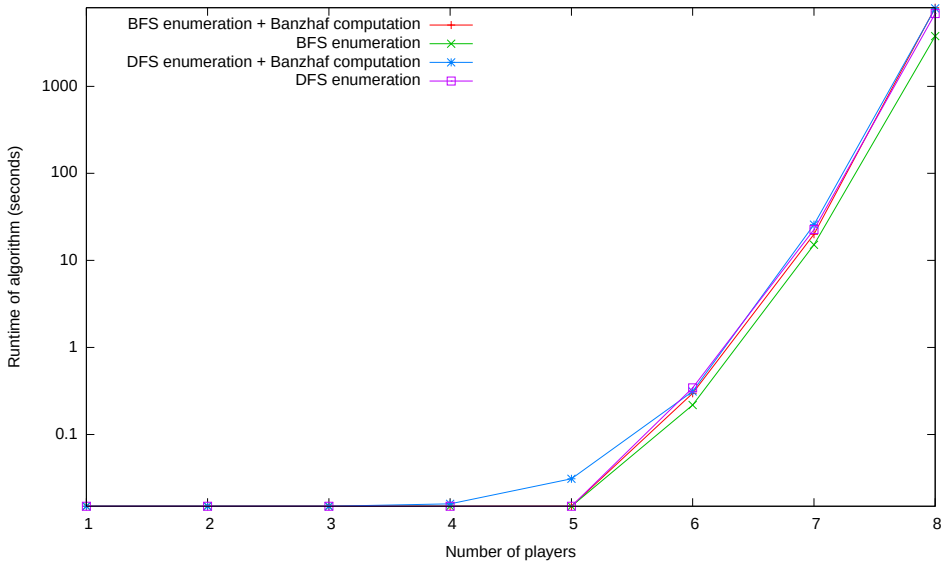

Figure 3: Runtimes of Algorithm 3 for 1 to 8 players, for both the breadth-first search and the depth-first search variant of the algorithm, both with and without the Banzhaf index computation procedure included.

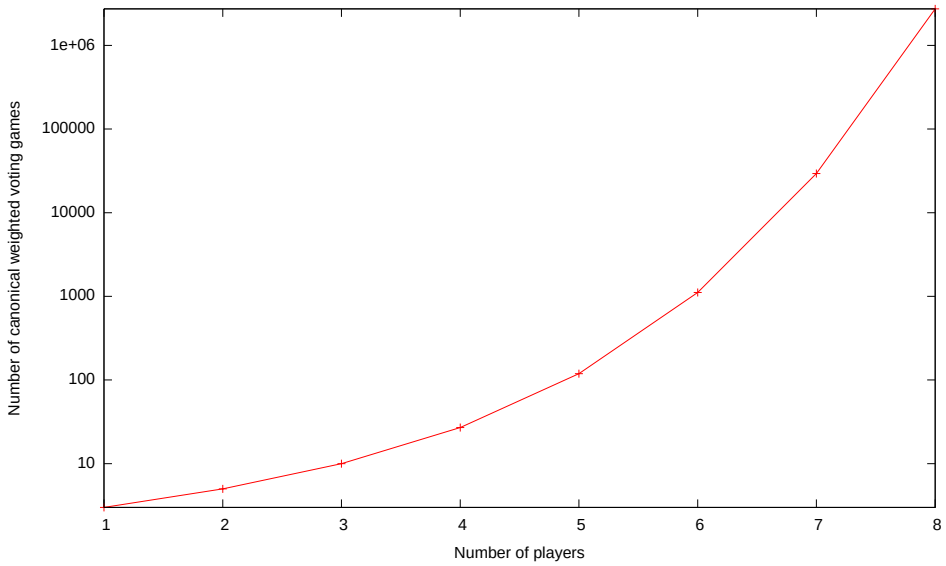

Figure 4: The number of canonical weighted voting games on n players, for $1 \leq n \leq 8$.

Table 2: 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	1113
7	29375
8	2730166

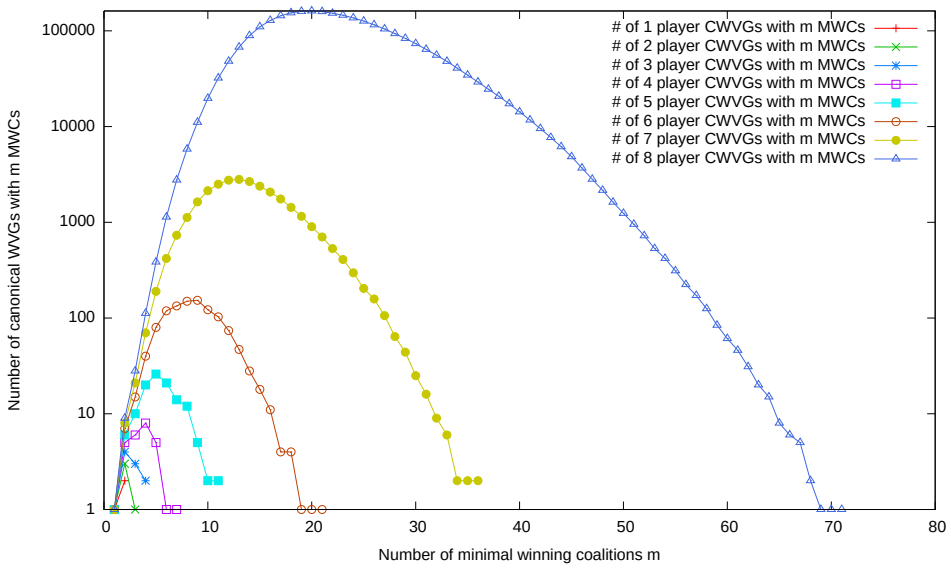

Figure 5: The number of canonical weighted voting games (y-axis) on n players, for $1 \leq n \leq 8$, with m minimal winning coalitions (x-axis).

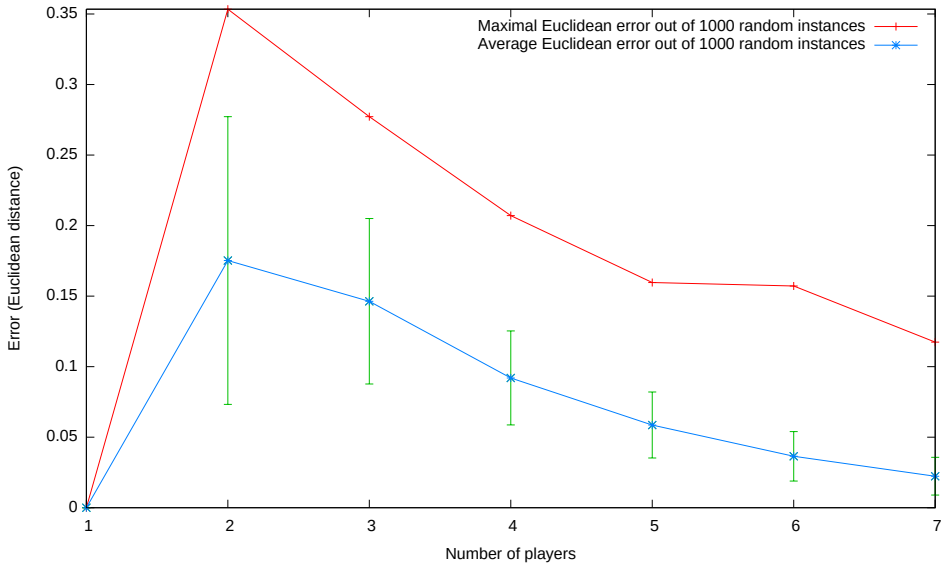

Figure 6: Optimal Euclidean error of 1000 random n player instances, for $1 \leq n \leq 7$. The error bars indicate one standard deviation.

The Euclidean errors computed in Experiment 3 are displayed in Figure 6. 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.

For Experiment 4, we see no possibility for a meaningful or interesting visualisation of its results. Experiment 4 confirms to us that this enumeration-approach of solving PVGD problems quickly becomes impractical as n gets larger. 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 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, it holds without exception that after the first second, the improvements are only tiny. The average euclidean errors obtained after letting the algorithm run for one minute are as follows:

- For $n = 10$, after one minute, the average euclidean 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 euclidean 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 euclidean 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 generated games are scattered more evenly across the unit simplex in the case of the breadth-first variant: We expect the depth-first variant to enumerate a lot of games for which the Banzhaf indices are close to each other, due to the cover relation of $(\mathcal{G}_{\mathsf{cwvg}}(n), \subseteq_{\mathsf{MWC}})$.

A final comment we would like to make is that when n gets larger, the output rate of the enumeration algorithm goes down. 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 time exponential 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 significant improvement in the performance of this algorithm by optimizing the code.

7 Conclusions & future work

In this paper, 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 developed 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 enumerates more games, and the quality of the solution will improve.

Also, due to the genericity of enumeration, we can use our algorithm not only to solve power index voting game design problems: we can use it to solve

any other voting game design problem as well. The only thing we have to adapt is the error-function of the algorithm (i.e., the part of the algorithm that checks the property in question for each of the games that the enumeration procedure outputs); the enumeration procedure does not need to be changed.

Finally, we implemented a simple, non-optimized version of the algorithm in order to do some experiments and obtain 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 small choices of n, and every m. We have seen that even for small n, it is already obvious from the experimental results that the number of weighted voting games grows quadratically on an exponential scale, precisely according to the known asymptotic bounds.

We measured the runtime of the algorithm, and observed that running the algorithm to completion becomes intractable at approximately $n = 10$ (on the computer that we performed the experiments with, we estimate that it takes a month to run the algorithm to completion for $n = 9$). Lastly, for larger values of n, our algorithm (or at least our current implementation) is of little use for practical purposes because the error does not converge as quickly as we would want to. We think that we can attain a significant speedup by optimizing the code, and by using better linear programming software.

Note that in most real-life examples, the number of players in a weighted voting game is rather small: usually 10 to 50 players are involved. For future work, the goal is to get this algorithm to yield good results within a reasonable amount of time when the number of players is somewhere in this range. We believe that there is still a lot of room for improving the proposed algorithm, and our current implementation of it.

We think that it will be interesting to study in more depth the partial order we introduced in this paper, both from a from a computational perspective and from a purely mathematical perspective. One possible prospect is the following. With regard to weighted voting game design problems, we suspect that it is possible to prune a lot of "areas" in this partial order: Careful analysis of the partial order and its properties might lead to results that allow us to construct an enumeration algorithm that *a priori* discards certain (hopefully large) subsets of weighted voting games.

We are moreover interested to see 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 if it is possible to use such a search method while still having an optimality guarantee or approximation guarantee on the quality of the solution. Lastly, we can also consider the ideas presented here as a postprocessing step to existing algorithms. In other words, it might be a good idea to first run the algorithm of [18] or [5] in order to obtain a good initial game. Subsequently, we can try to search through the "neighborhood" of the game to find improvements, according to the partial order introduced in this paper, .

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 of the problem. The runtime of our current algorithm implies that the problem is in EXPTIME for the case of weighted voting games, but it may be possible to characterize its computational complexity more precisely. We do not expect the problem to be complete for EXPTIME, but on the other hand, at the moment we do not have any ideas on how to prove hardness for this problem for any complexity class whatsoever. It seems a challenge to come up with a polynomial-time reduction from any known computational problem that is hard for any nontrivial complexity class. Also, on questions related to approximability of PVGD problems we currently do not have an answer.

References

[1] E. Algaba, J. M. Bilbao, J. R. Fernández García, and J. J. López. Computing power indices in weighted multiple majority games. *Mathematical Social Sciences*, 46:63–80, 2003.

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

[3] P. Antti. Voting power and power index website: a voting power WWW-resource including powerslave voting body analyser. WWW, april 2002. University of Turku, Finland. URL: `http://powerslave.val.utu.fi/` (last accessed on April 10, 2012).

[4] 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.

[5] 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.

[6] Y. Bachrach, V. Markakis, A. D. Procaccia, J. S. Rosenschein, and A. Saberi. Approximating power indices. In *Proceedings of The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 943–950, 2008.

[7] Y. Bachrach and J. S. Rosenschein. Computing the Banzhaf power index in network flow games. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7, New York, NY, USA, 2007. ACM.

[8] Y. Bachrach and J. S. Rosenschein. Power and stability in connectivity games. In *The Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portual, May 2008.

[9] 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.

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

[11] B. de Keijzer. On the design and synthesis of voting games : exact solutions for the inverse problem. Master's thesis, Delft University of Technology, 2009.

[12] B. de Keijzer, T. Klos, and Y. Zhang. Enumeration and exact design of weighted voting games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '10, pages 391–398, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[13] R. Dedekind. Über Zerlegungen von Zahlen durch ihre grössten gemeinsammen Teiler. *Gesammelte Werke*, 1:103–148, 1897.

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

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

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] B. de Keijzer. A survey on the computation of power indices. Technical report, Delft University of Technology, 2009. `http://www.st.ewi.tudelft.nl/~tomas/theses/DeKeijzerSurvey.pdf`.

[23] 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.

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

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

[26] I. Krohn and P. Sudhölter. Directed and weighted majority games. *Mathematical Methods of Operations Research*, 42(2):189–216, 1995.

[27] S. Kurz. On minimum sum representations for weighted voting games. *CoRR*, abs/1103.1445, 2011.

[28] S. Kurz. On the inverse power index problem. *Optimization*, 2012. To appear.

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

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

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

[32] D. Leech. Computing power indices for large voting games. *Management Science*, 49(6):831–838, June 2003.

[33] 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.

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

[35] 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.

[36] 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.

[37] Y. Matsui and T. Matsui. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1–2):305–310, 2001.

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

[39] S. Muroga, I. Toda, and M. Kondo. Majority functions of up to six variables. *Mathematics of Computation*, 60(80):459–472, October 1962.

[40] S. Muroga, T. Tsuboi, and C. R. Baugh. Enumeration of threshold functions of eight variables. *IEEE Transactions on Computers*, C-19(9):818–825, September 1970.

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

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

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

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

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

[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] T. Uno. Efficient computation of power indices for weighted majority games. Technical Report NII-2003-006E, National Institute of Informatics, 2003.

[52] R. O. Winder. Enumeration of seven-argument threshold functions. *IEEE Transactions on Electronic Computers*, EC-14(3):315–325, June 1965.

[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.