Computing the nucleolus of weighted voting games

Pasechnik, Dmitrii V.; Elkind, Edith

2009

Elkind, E., & Pasechnik, D. (2009). Computing the nucleolus of weighted voting games. Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA'09) (pp.327-335).

https://hdl.handle.net/10356/93815

Computing the nucleolus of weighted voting games

Edith Elkind * Dmitrii Pasechnik †

Abstract

Weighted voting games (WVG) are coalitional games in which an agent's contribution to a coalition is given by his *weight*, and a coalition wins if its total weight meets or exceeds a given quota. These games model decision-making in political bodies as well as collaboration and surplus division in multiagent domains. The computational complexity of various solution concepts for weighted voting games received a lot of attention in recent years. In particular, Elkind et al.(2007) studied the complexity of stability-related solution concepts in WVGs, namely, of the core, the least core, and the nucleolus. While they have completely characterized the algorithmic complexity of the core and the least core, for the nucleolus they have only provided an NP-hardness result. In this paper, we solve an open problem posed by Elkind et al. by showing that the nucleolus of WVGs, and, more generally, k-vector weighted voting games with fixed k, can be computed in pseudopolynomial time, i.e., there exists an algorithm that correctly computes the nucleolus and runs in time polynomial in the number of players n and the maximum weight W. In doing so, we propose a general framework for computing the nucleolus, which may be applicable to a wider of class of games.

*Intelligence, Agents, Multimedia group, School of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, United Kingdom, ee@ecs.soton.ac.uk

†Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore 637371, dima@ntu.edu.sg

1 Introduction

Both in human societies and in multi-agent systems, there are many situations where individual agents can achieve their goals more efficiently (or at all) by working together. This type of scenarios is studied by *coalitional game theory*, which provides tools to decide which teams of agents will form and how they will divide the resulting profit. In general, to describe a coalitional game, one has to specify the payoff available to every team, i.e., every possible subset of agents. The size of such representation is exponential in the number of agents, and therefore working with a game given in such form is computationally intensive. For this reason, a lot of research effort has been spent on identifying and studying classes of coalitional games that correspond to rich and practically interesting classes of problems and yet have a compact representation.

One such class of coalitional games is *weighted voting games*, in which an agent's contribution to a coalition is given by his *weight*, and a coalition has value 1 if its total weight meets or exceeds a given quota, and 0 otherwise. These games model decision-making in political bodies, where agents correspond to political parties and the weight of each party is the number of its supporters, as well as task allocation in multi-agent systems, where the weight of each agent is the amount of resources it brings to the table and the quota is the total amount of resources needed to execute a task.

An important issue in coalitional games is *surplus division*, i.e., distributing the value of the resulting coalition between its members in a manner that encourages cooperation. In particular, it may be desirable that all agents work together, i.e., form the *grand coalition*. In this case, a natural goal is to distribute the payoff of the grand coalition so that it remains *stable*, i.e., so as to minimize the incentive for groups of agents to deviate and form coalitions of their own. Formally, this intuition is captured by several related solution concepts, such as the core, the least core, and the nucleolus. Without going into the technical details of their definitions (see Section 3), the nucleolus is, in some sense, the most stable payoff allocation scheme, and as such it is particularly desirable when the stability of the grand coalition is important.

The stability-related solution concepts for WVGs have been studied from computational perspective in [5]. There, the authors show that while computing the core is easy, find-

ing the least core and the nucleolus is NP-hard. These computational hardness results rely on all weights being given in binary, which suggests that these problems may be easier for polynomially bounded weights. Indeed, paper [5] provides a pseudopolynomial time algorithm (i.e., an algorithm whose running time is polynomial in the number of players n and the maximal weight W) for the least core. However, an analogous question for the nucleolus has been left open.

In this paper, we answer this question in affirmative by presenting a pseudopolynomial time algorithm for computing the nucleolus.

THEOREM 1.1. *For a WVG specified by integer weights $w_1, \ldots, w_n$ and a quota q, there exists a procedure that computes its nucleolus in time polynomial in n and $W = \max_i w_i$.*

As in many practical scenarios (such as e.g., decision-making in political bodies) the weights are likely to be not too large, this provides a viable algorithmic solution to the problem of finding the nucleolus. Our approach relies on solving successive exponential-sized linear programs by constructing dynamic-programming based separation oracles, a technique that may prove useful in other applications.

A proof of Theorem 1.1 is presented in Section 4, after preliminaries in Section 3 and a discussion of related work in Section 2. The text rounds up by Section 5 that discusses conclusions and future work directions.

2 Related work

Another approach to payoff distribution in weighted voting games is based on *fairness*, i.e., dividing the payoff in a manner that is proportional to the agent's influence. The most popular solution concepts used in this context are the Shapley–Shubik power index [17] and the Banzhaf power index [1]. Both of these indices are known to be computationally hard for large weights [13, 4], yet efficiently computable for polynomially bounded weights [10].

The concept of the nucleolus was introduced by Schmeidler [15] in 1969. Paper [15] explains how the nucleolus arises naturally as "the most stable" payoff division scheme, and proves that the nucleolus is well-defined for any coalitional game and is unique. Kopelowitz [9] proposes to compute the nucleolus by solving a sequence of linear programs; we use this approach in our algorithm.

The computational complexity of the nucleolus has been studied for many classes of games, such as flow games [3, 14], cyclic permutation games [18], assignment games [12], matching games [8], and neighbor games [7], as well as several others. While some of these papers provide polynomial-time algorithms for computing the nucleolus, others contain NP-hardness results.

The work in this paper is inspired by [5], which shows that the least core and the nucleolus of a weighted voting game are NP-hard to compute. It also proves that the nucleolus cannot be approximated within any constant factor. On the positive side, it provides a pseudopolynomial time algorithm for computing the least core, i.e., an algorithm whose running time is polynomial in n and W (rather than in the game representation size $O(n \log W)$), as well as a fully polynomial time approximation scheme (FPTAS) for the least core. However, for the nucleolus, paper [5] contains no algorithmic results.

3 Preliminaries and Notation

A *coalitional game* $G = (I, \nu)$ is given by a set of agents $I = \{1, \ldots, n\}$ and a function $\nu : 2^I \to \mathbb{R}$ that maps any subset (coalition) of the agents to a real value. This value is the total utility these agents can guarantee to themselves when working together. A coalitional game is called *simple* if $\nu(S) \in \{0, 1\}$ for any coalition $S \subseteq I$. In a simple game, a coalition S is called *winning* if $\nu(S) = 1$ and *losing* otherwise.

A *weighted voting game* is a simple coalitional game G given by a set of agents $I = \{1, \ldots, n\}$, their non-negative *weights* $\mathbf{w} = (w_1, \ldots, w_n)$, and a *quota* q; we write $G = (I; \mathbf{w}; q)$. As the focus of this paper is computational complexity of such games, it is important to specify how the game is represented. In what follows, we assume that the weights and the quota are integers given in binary. This does not restrict the class of WVGs that we can work with, as any weighted voting game has such a representation [11].

For a coalition $S \subseteq I$, its value $\nu(S)$ is 1 (i.e., S is winning) if $\sum_{i \in S} w_i \geq q$; otherwise, $\nu(S) = 0$. Without loss of generality, we assume that the value of the grand coalition I is 1, that is, $\sum_{i \in I} w_i \geq q$. Also, we set $W = \max_{i \in I} w_i$.

For a coalitional game $G = (I, \nu)$, an *imputation* is a vector of non-negative numbers $\mathbf{p} = (p_1, \ldots, p_n)$, one for each agent in I, such that $\sum_{i \in I} p_i = \nu(I)$. We refer to p_i as the *payoff* of agent i. We write $p(S)$ to denote $\sum_{i \in S} p_i$. Similarly, $w(S)$ denotes $\sum_{i \in S} w_i$.

An important notion in coalitional games is that of stability: intuitively, a payoff vector should distribute the gains of the grand coalition in such a way that no group of agents has an incentive to deviate and form a coalition of their own. This intuition is captured by the notion of the core: the *core* of a game G is the set of all imputations $\mathbf{p}$ such that

$$p(S) \geq \nu(S) \text{ for all } S \subseteq I. \tag{3.1}$$

While the core is an appealing solution concept, it is very demanding: indeed, for many games of interest, the core is empty. In particular, it is well known that in simple games the core is empty unless there exists a veto player, i.e., a player that is present in all winning coalitions. Clearly, this is

not always the case in weighted voting games, and a weaker solution concept is needed.

We can relax the notion of the core by allowing a small error in the inequalities (3.1). This leads to the notion of ε-core: the *ε-core* of a game G is the set of all imputations $\mathbf{p}$ such that $p(S) \geq \nu(S) - \varepsilon$ for all $S \subseteq I$. Under an imputation in the ε-core, the *deficit* of any coalition S, i.e., the difference $\nu(S) - p(S)$ between its value and the payoff that it gets, is at most ε. Observe that if ε is large enough, e.g., $\varepsilon \geq 1$, then the ε-core is guaranteed to be non-empty. Therefore, a natural goal is to identify the smallest value of ε such that the ε-core is non-empty, i.e., to minimize the error introduced by relaxing the inequalities in (3.1). This is captured by the concept of the least core, defined as the smallest non-empty ε-core of the game. More formally, consider the set $\{\varepsilon \mid \varepsilon \leq 1, \varepsilon\text{-core of } G \text{ is non-empty}\}$. It is easy to see that this set is compact, so it has a minimal element ε_1. The *least core* of G is its ε_1-core. The imputations in least core distribute the payoff in a way that minimizes the incentive to deviate: under any $\mathbf{p}$ in the least core, no coalition can gain more than ε_1 by deviating, and for any $\varepsilon' < \varepsilon_1$, there is no way to distribute the payoffs so that the deficit of every coalition is at most ε'. However, while the least core minimizes the worst-case deficit, it does not attempt to minimize the *number* of coalitions that experience the worst deficit, i.e., ε_1, nor does it try to minimize the second-worst deficit, etc. The *nucleolus* is a refinement of the least core that takes into account these higher-order effects.

Recall that the deficit of a coalition S under an imputation $\mathbf{p}$ is given by $d(\mathbf{p}, S) = \nu(S) - p(S)$. The *deficit vector* of $\mathbf{p}$ is the vector $\mathbf{d}(\mathbf{p}) = (d(\mathbf{p}, S_1), \ldots, d(\mathbf{p}, S_{2^n}))$, where $S_1, \ldots, S_{2^n}$ is a list of all subsets of I ordered so that $d(\mathbf{p}, S_1) \geq d(\mathbf{p}, S_2) \geq \cdots \geq d(\mathbf{p}, S_{2^n})$. In other words, the deficit vector lists the deficits of all coalitions from the largest to the smallest (which may be negative). The *nucleolus* is an imputation $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$ that satisfies $\mathbf{d}(\boldsymbol{\eta}) \leq_{\text{lex}} \mathbf{d}(\mathbf{x})$ for any other imputation $\mathbf{x}$, where $\leq_{\text{lex}}$ is the lexicographic order. It is known [15] that the nucleolus is well-defined (i.e., an imputation with a lexicographically minimal deficit vector always exists) and is unique.

4 Algorithm

The description of our algorithm is structured as follows. We use the idea of [9], which explains how to compute the nucleolus by solving a sequence of (exponential-size) linear programs. In Section 4.1, we present the approach of [9], and argue that it correctly computes the nucleolus. This material is not new, and is presented here for completeness. In Section 4.2, we show how to design separation oracles for the linear programs in this sequence so as to solve them by the ellipsoid method. While a naive implementation of these separation oracles would require storing exponentially many constraints, we show how to replace explicit enumeration of these constraints with a counting subroutine, while preserving the correctness of the algorithm. The arguments in Sections 4.1 and 4.2 apply to *any* coalitional game rather than just weighted voting games.

In Section 4.3, we show that for weighted voting games with polynomially-bounded weights the counting subroutine used by the algorithm of Section 4.2 can be efficiently implemented. Finally, in Section 4.4 we show how to modify this subroutine to efficiently identify a violated constraint if a given candidate solution is infeasible. The results in Sections 4.3 and 4.4 are specific to weighted voting games with polynomially bounded weights.

4.1 Computing the nucleolus by solving successive linear programs As argued in [9], the nucleolus can be computed by solving at most n successive linear programs. The first linear program $\mathcal{LP}^1$ contains the inequality $p(S) \geq \nu(S) - \varepsilon$ for each coalition $S \subseteq I$, and attempts to minimize ε subject to these inequalities, i.e., it computes a payoff in the least core as well as the value ε^1 of the least core. Given a (*relative*) *interior optimizer* $(\mathbf{p}^1, \varepsilon^1)$ for $\mathcal{LP}^1$ (i.e., an optimal solution that minimizes the number of tight constraints), let Σ^1 be the set of all inequalities in $\mathcal{LP}^1$ that have been made tight by $\mathbf{p}^1$ (we will abuse notation and use Σ^1 to refer both to these inequalities and the corresponding coalitions). We construct the second linear program $\mathcal{LP}^2$ by replacing all inequalities in Σ^1 with equations of the form $p(S) = \nu(S) - \varepsilon^1$, and try to minimize ε subject to this new set of constraints. This results in $\varepsilon^2 < \varepsilon^1$ and a payoff vector $\mathbf{p}^2$ that satisfies $\mathbf{p}^2(S) = \nu(S) - \varepsilon^1$ for all $S \in \Sigma^1$, $\mathbf{p}^2(S) \geq \nu(S) - \varepsilon^2$ for all $S \notin \Sigma^1$. We repeat this process until the payoffs to all coalitions are determined, i.e., the solution space of the current linear program consists of a single point. It has been shown [9] that this will happen after at most n iterations: indeed, each iteration reduces the dimension of the solution space by at least 1.

More formally, the sequence of linear programs $(\mathcal{LP}^1, \ldots, \mathcal{LP}^n)$ is defined as follows. The first linear program $\mathcal{LP}^1$ is given by

(4.2)
$$\min_{(\mathbf{p},\varepsilon)} \varepsilon \text{ subject to } \begin{cases} \sum_{i \in I} p_i = 1, p_i \geq 0 \text{ for all } i = 1, \ldots, n \\ \sum_{i \in S} p_i \geq \nu(S) - \varepsilon \text{ for all } S \subseteq I. \end{cases}$$

Let $(\mathbf{p}^1, \varepsilon^1)$ be an interior optimizer to this linear program. Let Σ^1 be the set of tight constraints for $(\mathbf{p}^1, \varepsilon^1)$ (and, by a slight abuse of notation, the coalitions that correspond to them), i.e., for any $S \in \Sigma^1$ we have $p^1(S) = 1 - \varepsilon^1$.

Now, suppose that we have defined the first $j-1$ linear programs $\mathcal{LP}^1, \ldots, \mathcal{LP}^{j-1}$. For $k = 1, \ldots, j-1$, let $(\mathbf{p}^k, \varepsilon^k)$ be an interior optimizer for $\mathcal{LP}^k$ and let $\Sigma^k = \{S \mid$

$p^k(S) = \nu(S) - \varepsilon^k\}$. Then the jth linear program $\mathcal{LP}^j$ is given by

(4.3)
$$\min_{(\mathbf{p},\varepsilon)} \varepsilon \text{ subject to } \begin{cases} \sum_{i\in I} p_i = 1, p_i \geq 0 \text{ for all } i = 1,\dots,n \\ \sum_{i\in S} p_i = \nu(S) - \varepsilon^1 \quad \text{for all } S \in \Sigma^1 \\ \qquad \dots \\ \sum_{i\in S} p_i = \nu(S) - \varepsilon^{j-1} \text{ for all } S \in \Sigma^{j-1} \\ \sum_{i\in S} p_i \geq \nu(S) - \varepsilon \text{ for all } S \notin \cup_{k=1}^{j-1}\Sigma^k. \end{cases}$$

Fix the minimal value of t such that there is no interior solution to $\mathcal{LP}^t$. It is not hard to see that the (unique) solution to $\mathcal{LP}^t$ is indeed the nucleolus. Indeed, the nucleolus is a payoff vector that produces the lexicographically maximal deficit vector. This means that it:

(i) minimizes ε^1 such that all coalitions receive at least $1 - \varepsilon^1$;

(ii) given (i), minimizes the number of coalitions that receive $1 - \varepsilon^1$;

(iii) given (i) and (ii), minimizes ε^2 such that all coalitions except for those receiving $1 - \varepsilon^1$ receive at least $1 - \varepsilon^2$;

(iv) given (i), (ii) and (iii), minimizes the number of coalitions that receive $1 - \varepsilon^2$, etc.

Our sequence of linear programs finds a payoff vector that satisfies all these conditions; in particular, (ii) and (iv) (and analogous conditions at subsequent steps) are satisfied, since at each step we choose an interior optimizer for the corresponding linear program. The only issue that we have to address is that our procedure selects an *arbitrary* interior optimizer to the current linear program in order to construct the set Σ^j. Conceivably, this may have an impact on the final solution: if two interior optimizers to $\mathcal{LP}^j$ lead to two different sets Σ^j, they may also result in different values of ε^{j+1}, so one would have to worry about choosing the *right* interior optimizer. Fortunately, this is not the case, as shown by the following lemma.

LEMMA 4.1. *Any two interior optimizers $(\mathbf{p},\varepsilon)$ and $(\mathbf{q},\varepsilon)$ for the linear program $\mathcal{LP}^j$ have the same set of tight constraints, i.e., the set Σ^j is independent of the choice of the interior optimizer.*

Proof. First note that the set of all interior optimizers for $\mathcal{LP}^j$ is convex. Now, suppose that $\mathbf{p}$ and $\mathbf{q}$ are two interior optimizers for $\mathcal{LP}^j$, but have different sets of tight constraints. Then, by convexity, any convex combination $\alpha\mathbf{p} + (1-\alpha)\mathbf{q}$ of $\mathbf{p}$ and $\mathbf{q}$ is also an interior optimizer for $\mathcal{LP}^j$. However, the set of constraints that are tight for $\alpha\mathbf{p} + (1-\alpha)\mathbf{q}$ is the intersection of the corresponding sets for $\mathbf{p}$ and $\mathbf{q}$, i.e., $\alpha\mathbf{p} + (1-\alpha)\mathbf{q}$ has strictly fewer tight constraints than $\mathbf{p}$ or $\mathbf{q}$, a contradiction with $\mathbf{p}$ and $\mathbf{q}$ being interior optimizers for $\mathcal{LP}^j$.

We conclude that when this algorithm terminates, the output is indeed the nucleolus. Next, we discuss how to solve each of the linear programs $\mathcal{LP}^j$, $j = 1,\dots,t$.

4.2 Solving the linear programs $\mathcal{LP}^1,\dots,\mathcal{LP}^t$ It is well-known (see e.g. [16, 6]) that a linear program can be solved in polynomial time by the ellipsoid method as long as it has a polynomial-time *separation oracle*, i.e., an algorithm that, given a candidate feasible solution, either confirms that it is feasible or outputs a violated constraint. Moreover, the ellipsoid method can also be used to find an interior optimizer (rather than an arbitrary optimal solution) in polynomial time [6, Thm. 6.5.5], as well as to decide whether one exists [6, Thm. 6.5.6]. We will now construct a polynomial-time separation oracle for jth linear program $\mathcal{LP}^j$ in our sequence.

It is easy to construct the part responsible for checking equations of $\mathcal{LP}^j$ in (4.3), assuming that we already have an oracle for the $(j-1)$st program $\mathcal{LP}^{j-1}$. Indeed, the latter oracle can be easily modified to also check whether the equation $\varepsilon = \varepsilon^{j-1}$ holds, thus providing an oracle for the optimal face of $\mathcal{LP}^{j-1}$. Then by [6, Thm. 6.5.5] we can compute a basis of the optimal face (which consists of at most n equations) in polynomial time. The separation oracle can then reject a candidate solution $(\mathbf{p},\varepsilon)$ if $\mathbf{p}$ violates one of those basis equations.

Dealing with the inequalities of $\mathcal{LP}^j$ is more complicated. A naive separation oracle would have to explicitly list the sets $\Sigma^1,\dots,\Sigma^{j-1}$, which may be superpolynomial in size. Alternatively, one can treat this part of the oracle as a 0-1 integer linear feasibility problem, with 0-1 variables x_i encoding a set $S \notin \cup_{k=1}^{j-1}\Sigma^k$ that provides a separating inequality for the oracle input $(\mathbf{p},\varepsilon)$. Namely, suppose that we have verified that $\mathbf{p}$ satisfies all the equations in $\mathcal{LP}^j$ (as described above). Then, given an interior optimizer $(\mathbf{p}^{j-1},\varepsilon^{j-1})$ for $\mathcal{LP}^{j-1}$, the values $x_1,\dots,x_n$ can be obtained as a solution to the following inequalities:

$$\sum_i p_i^{j-1} x_i > 1 - \varepsilon^{j-1}, \tag{4.4}$$

$$\sum_i p_i x_i < 1 - \varepsilon, \tag{4.5}$$

$$\sum_i w_i x_i \geq q. \tag{4.6}$$

The problem with this approach is that for arbitrary rational $\mathbf{p}$ and $\mathbf{p}^{j-1}$ this system of inequalities is at least as hard as KNAPSACK, which is NP-complete. Moreover, as $\mathbf{p}$

and $\mathbf{p}^{j-1}$ are produced by the ellipsoid method, there is no guarantee that their bitsizes are small enough to use a (pseudo)polynomial-time algorithm for KNAPSACK. The only hope is to replace at least one of (4.4) and (4.5) by something "tame".

We will now present a more sophisticated approach to identifying a violated constraint. In a way, it can be seen as replacing checking (4.4) with counting. Our construction proceeds by induction: to construct a separation oracle for $\mathcal{LP}^j$, we assume that we have constructed an oracle for $\mathcal{LP}^{j-1}$, and are given the sizes of sets $\Sigma^1, \ldots, \Sigma^{j-1}$ as well as the sequence $(\varepsilon^1, \ldots, \varepsilon^{j-1})$.

By construction, any optimal solution $(\mathbf{p}, \varepsilon)$ to $\mathcal{LP}^j$ satisfies $\varepsilon < \varepsilon^{j-1}$, so we can add the constraint $\varepsilon \leq \varepsilon^{j-1}$ to $\mathcal{LP}^j$ without changing the set of solutions. From now on, we will assume that $\mathcal{LP}^j$ includes this constraint. Our separation oracle will first check whether a given candidate solution $(\mathbf{p}, \varepsilon)$ satisfies $\varepsilon \leq \varepsilon^{j-1}$, as well as constraints $p_i \geq 0$ for all $i = 1, \ldots, n$ and $p(I) = 1$, and reject $(\mathbf{p}, \varepsilon)$ and output a violated constraint if this is not the case. Therefore, in what follows we assume that $(\mathbf{p}, \varepsilon)$ satisfies all these easy-to-identify constraints.

Now, a candidate solution $(\mathbf{p}, \varepsilon)$ is feasible for $\mathcal{LP}^j$ if $p(S) = \nu(S) - \varepsilon^t$ for $S \in \Sigma^t$, $t = 1, \ldots, j-1$, and $p(S) \geq \nu(S) - \varepsilon$ for all $S \notin \cup_{t=1}^{j-1} \Sigma^t$. Recall that the deficit of a coalition $S \subseteq I$ under a payoff vector $\mathbf{p}$ is given by $\nu(S) - p(S)$. Suppose that we have a procedure $\mathcal{P}(\mathbf{p}, \varepsilon)$ that, given a candidate solution $(\mathbf{p}, \varepsilon)$, can efficiently compute the top j distinct deficits under $\mathbf{p}$, i.e.,

$$
\begin{aligned}
m^1 &= \max\{d(S) \mid S \subseteq I\} \\
m^2 &= \max\{d(S) \mid S \subseteq I, d(S) \neq m^1\} \\
&\ldots \\
m^j &= \max\{d(S) \mid S \subseteq I, d(S) \neq m^1, \ldots, m^{j-1}\}
\end{aligned}
$$

as well as the numbers $n^1, \ldots, n^j$ of coalitions that have deficits of $m^1, \ldots, m^j$, respectively:

$$
n^k = |\{S \mid S \subseteq I, d(S) = m^k\}|, \qquad k = 1, \ldots, j.
$$

Suppose also that we are given the values $\varepsilon^1, \ldots, \varepsilon^{j-1}$ and the sizes s^t of the sets Σ^t, $t = 1, \ldots, j-1$.

Now, our algorithm works as follows. Given a candidate solution $(\mathbf{p}, \varepsilon)$, it runs $\mathcal{P}(\mathbf{p}, \varepsilon)$ to obtain $m^t, n^t, t = 1, \ldots, j$. If $\varepsilon < \varepsilon^{j-1}$, it then checks whether

(a) $m^t = \varepsilon^t$ and $n^t = s^t$ for all $t = 1, \ldots, j-1$

(b) $m^j \leq \varepsilon$.

If $\varepsilon = \varepsilon^{j-1}$, it simply checks whether $m^t = \varepsilon^t$ for $t = 1, \ldots, j-1$ and $n^t = s^t$ for all $t = 1, \ldots, j-2$ [1]. If these conditions are satisfied, the algorithm answers that $(\mathbf{p}, \varepsilon)$ is indeed a feasible solution, and otherwise it identifies and outputs a violated constraint (for details of this step, see Section 4.4). We will now show that this algorithm implements a separation oracle for $\mathcal{LP}^j$ correctly and efficiently.

THEOREM 4.1. *Given the values ε^t, s^t, $t = 1, \ldots, j-1$, and a procedure $\mathcal{P}(\mathbf{p}, \varepsilon)$ that computes m^t, n^t, $t = 1, \ldots, j$, in polynomial time, our algorithm correctly decides whether a given pair $(\mathbf{p}, \varepsilon)$ is feasible for $\mathcal{LP}^j$ and runs in polynomial time.*

Proof. We start by proving an auxiliary lemma.

LEMMA 4.2. *For any vector $\mathbf{p}$ and any $t \leq j-1$ such that $m^s = \varepsilon^s$, $n^s = s^s$ for all $s \leq t$, the coalitions with deficit ε^s under $\mathbf{p}$ are exactly the ones in Σ^s.*

Proof. The proof is by induction on s. For $s = 1$, we have that the largest deficit of any coalition under $\mathbf{p}$ is ε^1, and there are exactly s^1 coalitions with this deficit. Hence, $(\mathbf{p}, \varepsilon^1)$ is an interior optimizer for $\mathcal{LP}^1$, and therefore the lemma follows by Lemma 4.1. Now, suppose that the lemma has been proven for $s-1$. By the induction hypothesis, $\mathbf{p}$ satisfies all constraints in $\Sigma^1, \ldots, \Sigma^{s-1}$. Also, under $\mathbf{p}$ there are at most $s^1 + \cdots + s^{s-1}$ coalitions whose deficit exceeds ε^s, so for all coalitions not in $\cup_{r=1}^{s-1} \Sigma^r$ their deficit is at most ε^s. Finally, there are exactly s^s coalitions whose deficit is exactly ε^s. Hence, $(\mathbf{p}, \varepsilon^s)$ is an interior optimizer for $\mathcal{LP}^s$, and therefore the lemma follows by Lemma 4.1.

To prove the theorem, let us first consider the case $\varepsilon < \varepsilon^{j-1}$. Suppose that $(\mathbf{p}, \varepsilon)$ satisfies (a) and (b). By using Lemma 4.2 with $t = j-1$, we conclude that $(\mathbf{p}, \varepsilon)$ satisfies all equations in $\mathcal{LP}^j$. Now, under $\mathbf{p}$, m^j is the largest deficit of a coalition not in $\cup_{t=1}^{j-1} \Sigma^t$. If this deficit is at most ε, then the pair $(\mathbf{p}, \varepsilon)$ is a feasible solution to $\mathcal{LP}^j$.

Conversely, suppose that (a) or (b) is violated. If $(\mathbf{p}, \varepsilon)$ satisfies (a) but violates (b), by using Lemma 4.2 with $t = j-1$ we conclude that, under $\mathbf{p}$ the coalitions in Σ^t have deficit ε^t for $t = 1, \ldots, j-1$, but the deficit of some coalition not in $\cup_{t=1}^{j-1} \Sigma^t$ exceeds ε. Hence, this coalition corresponds to a violated constraint. Now, suppose that (a) does not hold, and let s be the smallest index for which $m^s \neq \varepsilon^s$ or $n^s \neq s^s$. By using Lemma 4.2 with $t = s-1$, we conclude that for $r = 1, \ldots, s-1$ the coalitions in Σ^r have deficit ε^r. However, either the sth distinct deficit under $\mathbf{p}$ is not ε^s, in which case $\mathbf{p}$ violates a constraint in $2^I \setminus \cup_{r=1}^{s-1} \Sigma^r$, or under $\mathbf{p}$ there are more than s^s coalitions with deficit ε^s (note that by construction it cannot be the case that $n^s < s^s$). In the latter case, there is a coalition in $2^I \setminus \cup_{r=1}^{s} \Sigma^r$ whose deficit exceeds ε^s, thus violating the corresponding constraint. The case $\varepsilon = \varepsilon^{j-1}$ is similar. In this case for a candidate solution $(\mathbf{p}, \varepsilon)$ to be feasible, it is not required that there are exactly s^{j-1} coalitions with deficit ε^{j-1}. Hence, the algorithm only

[1] Alternatively, if $\varepsilon = \varepsilon^j$, one can verify whether $(\mathbf{p}, \varepsilon)$ is a feasible solution to the previous linear program $\mathcal{LP}^{j-1}$.

has to decide if for all $t = 1, \ldots, j-2$, the coalitions with deficit ε^t under $\mathbf{p}$ are exactly the ones in Σ^t, and all other coalitions get at least ε^{j-1}. Showing that our algorithm checks this correctly can be done similarly to the previous case.

The bound on the running time is obvious from the description of the algorithm.

To provide the value $s^j = |\Sigma^j|$ for the subsequent linear programs $\mathcal{LP}^{j+1}, \ldots, \mathcal{LP}^n$, we need to find an interior optimizer for $\mathcal{LP}^j$. Thm. 6.5.5 in [6] explains how to do this given a separation oracle for the optimal face, i.e., the set of all optimizers of $\mathcal{LP}^j$. Observe that such an oracle can be obtained by a slight modification of the oracle described above. Indeed, the optimal face is the set of solutions to the linear feasibility problem given by the constraints in $\mathcal{LP}^j$ together with the constraint $\varepsilon = \varepsilon^j$. The modified oracle first checks the latter constraint, reports the violation (and the corresponding inequality) if it happens, and otherwise continues as the original oracle. Clearly, the modified oracle runs in polynomial time whenever the original one does. Hence, we can compute s^j in polynomial time by computing an interior solution $(\mathbf{p}, \varepsilon)$ to $\mathcal{LP}^j$ according to [6, Thm. 6.5.5], running $\mathcal{P}(\mathbf{p}, \varepsilon)$ to find n^j, and setting $s^j = n^j$.

4.3 Implementing the counting We will now show how to implement the counting procedure $\mathcal{P}(\mathbf{p}, \varepsilon)$ used in Section 4.2 for WVGs. The running time of our procedure is polynomial in the number of players n and the maximum weight W.

Our approach is based on dynamic programming. Fix a WVG $(I; \mathbf{w}; q)$, a payoff vector $\mathbf{p}$, and $j \leq n$. For all $k = 1, \ldots, n$, $w = 1, \ldots, nW$, let $X^1_{k,w}, \ldots, X^j_{k,w}$ be the bottom j distinct payoffs to coalitions in $\{1, \ldots, k\}$ of weight w, i.e., define

$$\begin{aligned}
X^1_{k,w} &= \min\{p(S) \mid \quad S \subseteq \{1, \ldots, k\}, w(S) = w\} \\
X^2_{k,w} &= \min\{p(S) \mid \quad S \subseteq \{1, \ldots, k\}, w(S) = w, \\
&\qquad\qquad\qquad\qquad p(S) \neq X^1_{k,w}\} \\
&\cdots \\
X^j_{k,w} &= \min\{p(S) \mid \quad S \subseteq \{1, \ldots, k\}, w(S) = w, \\
&\qquad\qquad\qquad\qquad p(S) \neq X^1_{k,w}, \ldots, X^{j-1}_{k,w}\},
\end{aligned}$$

and let $Y^1_{k,w}, \ldots, Y^j_{k,w}$ be the numbers of coalitions that get these payoffs, i.e. set

$$Y^t_{k,w} = |\{S \mid S \subseteq \{1, \ldots, k\}, w(S) = w, p(S) = X^t_{k,w}\}|$$

for $t = 1, \ldots, j$. These quantities can be computed inductively for $k = 1, \ldots, n$ as follows.

For $k = 1$, we have $X^1_{1,w} = p_1$ if $w = w_1$ and $+\infty$ otherwise, $Y^1_{1,w} = 1$ if $w = w_1$ and 0 otherwise, and $X^t_{1,w} = +\infty$, $Y^t_{1,w} = 0$ for $t = 2, \ldots, j$.

Now, suppose that we have computed $X^1_{k-1,w}, \ldots, X^j_{k-1,w}, Y^1_{k-1,w}, \ldots, Y^j_{k-1,w}$ for all $w = 1, \ldots, nW$. Consider $S \subseteq \{1, \ldots, k\}$ receiving one of the bottom j distinct payoffs to subsets of $\{1, \ldots, k\}$ of weight w, i.e., $p(S) \in \{X^1_{k,w}, \ldots, X^j_{k,w}\}$. Then either

(1) $S \subseteq \{1, \ldots, k-1\}$, in which case S must be among the coalitions that receive one of the bottom j distinct payoffs to subsets of $\{1, \ldots, k-1\}$ of weight w, i.e., we have $p(S) \in \{X^1_{k-1,w}, \ldots, X^j_{k-1,w}\}$, or

(2) $k \in S$, in which case $S \setminus \{k\}$ must be among the coalitions that receive one of the bottom j distinct payoffs to subsets of $\{1, \ldots, k-1\}$ of weight $w - w_k$, i.e., we have $p(S \setminus \{k\}) \in \{X^1_{k-1,w-w_k}, \ldots, X^j_{k-1,w-w_k}\}$.

Consider the multi-set $\mathcal{S}_{k,w} = \{X^1_{k-1,w}, \ldots, X^j_{k-1,w}, p_k + X^1_{k-1,w-w_k}, \ldots, p_k + X^j_{k-1,w-w_k}\}$. By the argument above, we have

$$\begin{aligned}
X^1_{k,w} &= \min\{x \mid x \in \mathcal{S}_{k,w}\} \\
X^2_{k,w} &= \min\{x \mid x \in \mathcal{S}_{k,w}, x \neq X^1_{k,w}\} \\
&\cdots \\
X^j_{k,w} &= \min\{x \mid x \in \mathcal{S}_{k,w}, x \neq X^1_{k,w}, \ldots, X^{j-1}_{k,w}\}.
\end{aligned}$$

The number of coalitions that receive the payoff $X^t_{k,w}$, i.e., $Y^t_{k,w}$, $t = 1, \ldots, j$, depends on how many times $X^t_{k,w}$ appears in $\mathcal{S}_{k,w}$. If it only appears once, then there is only one source of sets that receive a payoff of $X^t_{k,w}$, i.e., we set $Y^t_{k,w} = Y^s_{k-1,w}$ if $X^t_{k,w}$ appears as $X^s_{k-1,w}$ for some $s = 1, \ldots, j$, and we set $Y^t_{k,w} = Y^s_{k-1,w-w_k}$ if $X^t_{k,w}$ appears as $X^s_{k-1,w-w_k} + p_k$ for some $s = 1, \ldots, j$. On the other hand, if $X^t_{k,w}$ appears twice in $\mathcal{S}_{k,w}$ (first time as $X^s_{k-1,w}$ and second time as $p_k + X^{s'}_{k-1,w-w_k}$ for some $s, s' = 1, \ldots, j$), we have to add up the corresponding counts, i.e., we set $Y^t_{k,w} = Y^s_{k-1,w} + Y^{s'}_{k-1,w-w_k}$.

After all $X^1_{n,w}, \ldots, X^j_{n,w}, Y^1_{n,w}, \ldots, Y^j_{n,w}$ have been evaluated, it is not hard to compute m^t, n^t, $t = 1, \ldots, j$. Indeed, the top j deficits appear in the multi-set

$$\mathcal{S} = \{I_w - X^1_{n,w}, \ldots, I_w - X^j_{n,w} \mid w = 1, \ldots, nW\},$$

where $I_w = 1$ if $w \geq q$ and $I_w = 0$ if $w < q$ (recall that q is the quota of the game, i.e., $\nu(S) = 1$ if and only if $w(S) \geq q$). Hence, we can set

$$\begin{aligned}
m^1 &= \max\{x \mid x \in \mathcal{S}\} \\
m^2 &= \max\{x \mid x \in \mathcal{S}, x \neq m^1\} \\
&\cdots \\
m^j &= \max\{x \mid x \in \mathcal{S}, x \neq m^1, \ldots, m^{j-1}\}.
\end{aligned}$$

The procedure for computing n^t, $t = 1, \ldots, j$, is similar to that of computing $Y^s_{k,w}$: we have to check how many times m^t appears in $\mathcal{S}$ and add the corresponding counts.

In the next subsection, we will show how to find a violated inequality if $(\mathbf{p}, \varepsilon)$ is not a feasible solution to $\mathcal{LP}^j$.

4.4 Identifying a violated constraint Consider $\mathcal{LP}^j$ and a candidate solution $(\mathbf{p}, \varepsilon)$. Suppose that the algorithm described in the previous subsection has decided that $(\mathbf{p}, \varepsilon)$ is not a feasible solution to $\mathcal{LP}^j$. This can happen in three possible ways.

(a) $m^s = \varepsilon^s$, $n^s = s^s$ for $s = 1, \ldots, \ell - 1$, but $m^\ell \neq \varepsilon^\ell$ for an $\ell < j$.

(b) $m^s = \varepsilon^s$, $n^s = s^s$ for $s = 1, \ldots, \ell - 1$, $m^\ell = \varepsilon^\ell$, but $n^\ell \neq s^\ell$ for an $\ell < j$.

(c) $m^s = \varepsilon^s$, $n^s = s^s$ for $s = 1, \ldots, j - 1$, but $m^j > \varepsilon$.

In cases (a) and (b), there is a violated equation in (4.3), while in (c) there are none (but there is a violated inequality). Thus (a) and (b) can be handled using the ideas discussed in the beginning of Section 4.2. Indeed, as argued there, we can efficiently compute the basis of the optimal face of the feasible set of $\mathcal{LP}^{j-1}$ using the ellipsoid method. One can then easily check if a candidate solution violates one of the equations in the basis (recall that there are at most n of them), and, if this is the case, report one that is violated. Hence, we only need to show how to identify a violated constraint in case (c). However, for completeness, we present here a purely counting-based algorithm for each of the cases.

In case (a), let $(\hat{\mathbf{p}}, \varepsilon^\ell)$ be an interior optimizer for $\mathcal{LP}^\ell$. Under $\hat{\mathbf{p}}$, the deficit of any coalition in $2^I \setminus \cup_{s=1}^{\ell-1}\Sigma^s$ is at most ε^ℓ. On the other hand, under $\mathbf{p}$, there are n^ℓ coalitions in $2^I \setminus \cup_{s=1}^{\ell-1}\Sigma^s$ whose deficit is $m^\ell > \varepsilon^\ell$. Each of these coalitions corresponds to a violated constraint: indeed, if such a coalition is in Σ^s, $s \geq \ell$, then $\mathcal{LP}^j$ requires that its deficit is $\varepsilon^s \leq \varepsilon^\ell < m^\ell$, and if it is in $2^I \setminus \cup_{s=1}^{j-1}\Sigma^s$, then $\mathcal{LP}^j$ requires that its deficit is at most $\varepsilon \leq \varepsilon^\ell < m^\ell$, Hence, it suffices to identify a coalition whose deficit under $\mathbf{p}$ is m^ℓ. To this end, we can modify the dynamic program for $\mathbf{p}$ as follows. Together with every variable $X^t_{k,w}$, $t = 1, \ldots, j$, $k = 1, \ldots, n$, $w = 1, \ldots, nW$, we will use an auxiliary variable $Z^t_{k,w}$ which stores a coalition whose payoff under $\mathbf{p}$ is equal to $X^t_{k,w}$. The values of $Z^t_{k,w}$ can be easily computed by induction: if $X^t_{k,w} = X^s_{k-1,w}$ for some $s = 1, \ldots, j$ then $Z^t_{k,w} = Z^s_{k-1,w}$, and if $X^t_{k,w} = p_k + X^s_{k-1,w-w_k}$ for some $s = 1, \ldots, j$ then $Z^t_{k,w} = Z^s_{k-1,w} \cup \{k\}$ (if $X^t_{k,w} = X^s_{k-1,w} = p_k + X^{s'}_{k-1,w-w_k}$, we can set $Z^t_{k,w}$ to either of these values). Now, there exist some t and w such that $w \geq q$ and $1 - X^t_{n,w} = m^\ell$ or $w < q$ and $-X^t_{n,w} = m^\ell$; such t and w can be found by scanning all $X^t_{n,w}$. The corresponding set $Z^t_{n,w}$ has deficit m^ℓ under $\mathbf{p}$, and therefore corresponds to a violated constraint.

In case (b), let $(\mathbf{p}, \varepsilon)$ be an interior optimizer to $\mathcal{LP}^{j-1}$. There exists a coalition whose deficit under $\mathbf{p}$ is ε^ℓ, but whose deficit under $\hat{\mathbf{p}}$ is strictly less than ε^ℓ. To find such a coalition, run $\mathcal{P}(\hat{\mathbf{p}}, \varepsilon)$ in order to compute the corresponding values $\hat{X}^t_{k,w}$, $\hat{Y}^t_{k,w}$ $t = 1, \ldots, j-1$, $k = 1, \ldots, n$, $w = 1, \ldots, nW$. Define Z_w as follows: if there exists some $t \in \{1, \ldots, j-1\}$ such that $X^t_{n,w} = I_w - \varepsilon^\ell$, set $Z_w = Y^t_{n,w}$; otherwise, set $Z_w = 0$. $\hat{Z}_w$ is defined similarly: if there exists an $s \in \{1, \ldots, j-1\}$ such that $\hat{X}^s_{n,w} = I_w - \varepsilon^\ell$, set $\hat{Z}_w = \hat{Y}^s_{n,w}$; otherwise, set $\hat{Z}_w = 0$. The variables Z_w and $\hat{Z}_w$ count the number of coalitions that have total weight w and have deficit ε^ℓ under $\mathbf{p}$ and $\hat{\mathbf{p}}$, respectively. We have $n^\ell = \sum_{w=1,\ldots,nW} Z_w$, $s^\ell = \sum_{w=1,\ldots,nW} \hat{Z}_w$. As $n^\ell > s^\ell$, there exists a weight w such that $Z_w > \hat{Z}_w$. Set $q = I_w - \varepsilon^\ell$, i.e., q is the total payment received by the coalitions counted by Z_w and $\hat{Z}_w$. Now, we have $Z_w = Z^n_w + Z^{-n}_w$, where Z^n_w is the number of coalitions of weight w that include n, have weight w and receive total payment q, and Z^{-n}_w is the number of coalitions of weight w that do not include n, have weight w and receive total payment ε^ℓ; $\hat{Z}^n_w$ and $\hat{Z}^{-n}_w$ can be defined similarly. We can easily compute these quantities: for example, Z^n_w is the number of subsets of $\{1, \ldots, n-1\}$ that have weight $w - w_n$ and receive total payment $q - p_n$, i.e. $Z^n_w = Y^t_{n-1,w-w_n}$ if there exists a $t \in \{1, \ldots, j-1\}$ such that $X^t_{n-1,w-w_n} = q - p_n$, and $Z^{-n}_w = 0$ otherwise. It follows immediately that $Z^n_w > \hat{Z}^n_w$ or $Z^{-n}_w > \hat{Z}^{-n}_w$ (or both), and we can easily verify which of these cases holds. In the former case, we can conclude that the number of coalitions in $\{1, \ldots, n-1\}$ that have weight $w - w_n$ and are paid $q - p_n$ under $\mathbf{p}$ exceeds the number of coalitions in $\{1, \ldots, n-1\}$ that have weight $w - w_n$ and are paid $q - \hat{p}_n$ under $\hat{\mathbf{p}}$. In the latter case, we can conclude that the number of coalitions in $\{1, \ldots, n-1\}$ that have weight w and are paid q under $\mathbf{p}$ exceeds the number of coalitions in $\{1, \ldots, n-1\}$ that have weight w and are paid q under $\hat{\mathbf{p}}$. Continuing in the same manner for $n-1, \ldots, 1$, we can identify a coalition that is paid q under $\mathbf{p}$, but not under $\hat{p}$.

Case (c), i.e. $m^j > \varepsilon$, is similar to (a) and can be handled in the same manner.

5 Conclusions and future work

In this paper, we proposed a new technique for computing the nucleolus of coalitional games. Namely, we have shown that, when constructing the separation oracle for the jth linear program $\mathcal{LP}^j$, instead of storing the sets of tight constraints for the linear programs $\mathcal{LP}^t$, $t = 1, \ldots, j-1$, it suffices to store the sizes of these sets as well as the top $j-1$ deficits of an interior optimizer $(\mathbf{p}^{j-1}, \varepsilon)$ to $\mathcal{LP}^{j-1}$. A feasibility of a candidate solution $(\mathbf{p}, \varepsilon)$ to $\mathcal{LP}^j$ can then be verified, roughly, by computing the top j deficits for $\mathbf{p}$ as well as the number of coalitions that have these deficits, and comparing these values to their pre-computed counterparts for $(\mathbf{p}^{j-1}, \varepsilon)$.

We then demonstrated the usefulness of this tech-

nique by showing that for weighted voting games with polynomially-bounded weights both the top j deficits and the number of coalitions that have these deficits can be efficiently computed using dynamic programming. This allows us to implement the separation oracles for our linear programs in pseudopolynomial time. Combining this with the ellipsoid algorithm results in a pseudopolynomial time algorithm for the nucleolus of weighted voting games, thus solving an open problem posed by [5]. Furthermore, the general technique put forward in this paper effectively reduces the computation of the nucleolus to solving a natural combinatorial problem for the underlying game. Namely, we can state the following meta-theorem:

THEOREM 5.1. *Given a coalitional game G, suppose that we can, for any payoff vector $\mathbf{p}$, identify the top n distinct deficits under $\mathbf{p}$ as well as the number of coalitions that have these deficits in polynomial time. Then we can compute the nucleolus of G in polynomial time.*

We believe that this framework can be useful for computing the nucleolus in other classes of games. Indeed, by stripping away most of the game-theoretic terminology, we may be able to find the nucleolus by applying existing results in combinatorics and discrete mathematics in a black-box fashion.

In the context of weighted voting games, our assumption that the weights are polynomially bounded (or, equivalently, given in unary) is essential, as [5] shows that the nucleolus is NP-hard to compute for WVGs with weights given in binary. Moreover, in many practical scenarios the agents' weights cannot be too large (e.g., polynomial functions of n), in which case the running time of our algorithm is polynomial.

By a slight modification of our algorithm, we can obtain a pseudopolynomial time algorithm for computing the nucleolus in *k-vector weighted voting games* for constant k. Informally speaking, these are games given by the intersection of k weighted voting games, i.e., a coalition is considered to be winning if it wins in each of the underlying games. There are some interesting games that can be represented as k-vector weighted voting games for small values of k (i.e., $k = 2$ or $k = 3$), but not as weighted voting games, most notably, voting in the European Union [2]. Hence, this extension of our algorithm enables us to compute the nucleolus in some real-life scenarios. The overall structure of our algorithm remains the same. However, the dynamic program has to be modified to keep track of several weight systems simultaneously.

Another natural way to address the problem of computing the nucleolus is by focusing on approximate solutions. Indeed, [5] proposes a fully polynomial time approximation scheme (FPTAS) for several least-core related problems. It would be natural to expect a similar result to hold for the nucleolus. Unfortunately, this approach is ruled out by [5], which shows that it is NP-hard to decide whether the nucleolus payoff of any particular player is 0, and therefore approximating the nucleolus payoffs up to any constant factor is NP-hard. Nevertheless, one can attempt to find an *additive* approximation to the nucleolus, i.e., for a given error bound $\delta > 0$, find a vector $\mathbf{x}$ such that $|\eta_i - x_i| \leq \delta$ for $i = 1, \ldots, n$. This can be useful in situations when the agents' weights cannot be assumed to be polynomially bounded with respect to n, e.g., in the multiagent settings where the weights correspond to agents' resources. We are currently investigating several approaches to designing additive approximation algorithms for the nucleolus.

References

[1] J. F. Banzhaf. Weighted voting doesn't work: a mathematical analysis. *Rutgers Law Review*, 19:317–343, 1965.

[2] J. M. Bilbao, J. R. Fernández, N. Jiminéz, and J. J. López. Voting power in the European Union enlargement. *European Journal of Operational Research*, 143:181–196, 2002.

[3] X. Deng, Q. Fang, and X. Sun. Finding nucleolus of flow game. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 124–131, New York, 2006. ACM.

[4] X. Deng and C. H. Papadimitriou. On the complexit of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, 1994.

[5] E. Elkind, L. A. Goldberg, P. W. Goldberg, and M. Wooldridge. Computational complexity of weighted threshold games. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, volume 1, pages 718–723, Menlo Park, California, 2007. AAAI Press. ISBN 978-1-57735-323-2.

[6] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 1993.

[7] H. Hamers, F. Klijn, T. Solymosi, S. Tijs, and D. Vermeulen. On the nucleolus of neighbor games. *European J. Oper. Res.*, 146(1):1–18, 2003.

[8] W. Kern and D. Paulusma. Matching games: the least core and the nucleolus. *Math. Oper. Res.*, 28(2):294–308, 2003.

[9] A. Kopelowitz. Computation of the kernels of simple games and the nucleolus of n-person games. Preprint RM 37, 1967. Research Program in Game Theory and Mathematical Economics.

[10] T. Matsui and Y. Matsui. A survey of algorithms for calculating power indices of weighted majority games. *J. Oper. Res. Soc. Japan*, 43(1):71–86, 2000. New trends in mathematical programming (Kyoto, 1998).

[11] S. Muroga. *Threshold Logic and its Applications*. John Wiley & Sons, 1971.

[12] M. Núñez. A note on the nucleolus and the kernel of the assignment game. *Internat. J. Game Theory*, 33(1):55–65, 2004.

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

[14] H. Reijnierse, J. Potters, A. Biswas. The nucleolus of balanced simple flow networks. *Games and Economic Behavior*, 54 (1), 205–225, 2006.

[15] D. Schmeidler. The nucleolus of a characteristic function game. *SIAM J. Appl. Math.*, 17:1163–1170, 1969.

[16] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons Ltd., Chichester, 1986. A Wiley-Interscience Publication.

[17] L. S. Shapley and M. Shubik. A method for evaluating the distribution of power in a committee system. In *The Shapley value*, pages 41–48. Cambridge Univ. Press, Cambridge, 1988.

[18] T. Solymosi, T. E. S. Raghavan, and S. Tijs. Computing the nucleolus of cyclic permutation games. *European J. Oper. Res.*, 162(1):270–280, 2005.