Finding the Nucleolus of Large Cooperative Games

Tri-Dung Nguyen* Lyn Thomas†

June 29, 2014

Abstract

The nucleolus is one of the most important solution concepts in cooperative game theory as a result of its attractive properties - it always exists, is unique, and is always in the core (if the core is non-empty). However, computing the nucleolus is very challenging because it involves the lexicographical minimization of an exponentially large number of excess values. We present a method for computing the nucleolus of large games, i.e. those with more than 50 players, using nested linear programs (LP). Although different variations of the nested LP formulation have been documented in the literature, they have not been used for large games because of the large size and number of LPs involved. In addition, subtle issues such as how to deal with multiple optimal solutions and with tight constraint sets need to be resolved in each LP in order to formulate and solve the subsequent ones. Unfortunately, this technical issue has been largely overlooked in the literature. We treat these issues rigorously and provide a new nested LP formulation that is smaller in terms of the number of large LPs and their sizes. We provide numerical tests for several games, including the general flow games, the coalitional skill games and the weighted voting games, with up to 100 players.

Keywords: Nucleolus; cooperative game; multi-level programming; payoff distribution; constraint generation; lexicographical minimization.

1 Introduction and Literature Review

The nucleolus is one of the most important solution concepts for cooperative games with transferable utilities. It represents a way to distribute the reward (or cost) among the players involved in a way that lexicographically minimizes the excess values (i.e. dissatisfaction levels) of all coalitions. The nucleolus was introduced in 1969 by Schmeidler [33] as a solution concept with attractive properties - it always exists, it is unique, and it lies in the core (if the core is non-empty). We review concepts in cooperative game theory and their mathematical definitions in Section 2.1. The nucleolus concept has been used in many different applications. For example, in the banking industry, groups of banks enter into an agreement for their customers to use ATM machines owned by any bank in the same group. The

*Mathematics and Management School, University of Southampton, Southampton SO17 1BJ, United Kingdom, Email: T.D.Nguyen@soton.ac.uk. Tel.: +44 (0)23 8059 7759 Fax: +44 (0)23 8059 5147

†Southampton Management School, University of Southampton, Southampton SO17 1BJ, United Kingdom, T.D.Nguyen@soton.ac.uk.

nucleolus is then used to suggest how the cost of installing and maintaining those ATM machines can be shared among the banks (see Gow and Thomas [15]). It has also been applied to insurance premium setting (Lemaire [25]) and to network cost sharing (Deng et al. [8], Granot and Huberman [16], Granot and Maschler [17]), among many other applications.

Despite the desirable properties that the nucleolus has, its computation is, however, very challenging because the process involves the lexicographical minimization of 2^n excess values, where n is the number of players. Kohlberg [21] provides a necessary and sufficient condition for an imputation to be the nucleolus. However, the criterion requires forming the coalition array of all 2^n possible coalitions and then making sure 2^n linear inequalities are satisfied. The analytical form of the nucleolus is only available for games with three players Leng and Parlar [26]. There are a small number of games whose nucleoli can be computed in polynomial time. These include the connected games in Solymosi and Raghavan [35], the neighbor games in Hamers et al. [18], the cyclic permutation games in Solymosi et al. [36], and the flow games with unit capacities in Potters et al. [30], Deng et al. [8], Kern and Paulusma [20]. It has been shown that finding the nucleolus is NP-hard for many games such as the utility games with non-unit capacities in Deng et al. [8] and the weighted voting games in Elkind et al. [11]. In fact, finding the core and the least core is NP-hard in supermodular games [34] and inventory centralization games [7].

Kopelowitz [23] suggests using nested linear programming (LP) to compute the kernel of a game. This encouraged a number of researchers to compute the nucleolus using linear programming. For example, Kohlberg [22] presents a single LP with $\mathrm{O}\big(2^n!\big)$ constraints. The number of constraints in the LP formulation of Owen [28] is reduced to $\mathrm{O}\big(4^n\big)$ but the coefficients get larger. The nucleolus can also be found by solving a sequence of LPs. However, the number of LPs involved is exponentially large (i.e. $\mathrm{O}\big(4^n\big)$ in Maschler et al. [27] and $\mathrm{O}\big(2^n\big)$ in Sankaran [32]). Potters et al. [31] present another formulation that involves solving $(n-1)$ linear programs with, at most, (2^n+n-1) rows and (2^n-1) columns. The authors also develop a prolonged Simplex method for solving these large LPs and conduct numerical experiments for games with 10 players. Derks and Kuipers [9] improve the implementation of the prolonged Simplex method in [31] and provide numerical results for games with 20 players. Göthe-Lundgren et al. [14] attempted to apply a constraint generation framework to find the nucleoli of basic vehicle routing games. However, some of results are incorrect as has been pointed out by Chardaire [6]. The issue of having multiple optimal solutions in each LP was not considered in [14], but we are able to deal with that in Section 2.4. Fromen [13] uses Gaussian elimination to improve the formulation of Sankaran [32] and to reduce the number of LPs in the implementation. Nevertheless, the LPs are still extremely large when the number of players gets larger and existing methods become intractable when n exceeds 20. In the nested LPs formulation, subsequent LPs are formed based on the optimal solutions of the previous LPs and the tight inequalities. One needs to be very careful if there are multiple optimal solutions to these LPs and if there are multiple coalitions with the same worst excess values. The paper provides a rigorous treatment of the nested LPs formulation which deals with these issues.

For each payoff distribution, there are 2^n excess values that correspond to 2^n possible coalitions. The nucleolus is the payoff distribution that lexicographically minimizes its excess values. Thus finding

the nucleolus effectively is only possible if one can find the worst coalition(s) for a given imputation efficiently. Faigle et al. [12] show that the nucleolus can be found in polynomial time if finding the worst coalition for a given imputation, i.e. the separation problem, can be done in polynomial time. Their method is based on the ellipsoid algorithm which, although theoretically having a polynomial runtime, does not perform well in practice. Our paper bridges this gap and presents a practical numerical procedure for computing the nucleolus. We test our algorithm with the coalitional skill games and the weighted voting game.

The key contributions of our work include:

- We present a nested LPs formulation for computing the nucleoli of cooperative games. Although the idea of using a nested LPs framework has been around for more than 40 years ago (Kopelowitz [23]) with various reformulations having been proposed, these methods face several issues that will be described in detail in Section 2.4. The most critical issue among these is how to *handle multiple optimal solutions in each of the large LPs.* Dealing with multiple optimal solutions is often needed in multi-level programming and is often very challenging. We provide a concrete method for dealing with these issues in Sections 3.2-3.5.

- The size of our nested LPs formulation is smaller than other nested LPs formulation described in the literature as can be seen in Table 1. The number of LPs to be solved in our method is smaller than that in Maschler et al. [27] and Sankaran [32] while the number of columns in each LP is smaller than that in Potters et al. [31]. These features are results of our special way of handling tight coalitions and finding the minimal tight sets (the key idea in Theorem 1 and the main results in Theorem 2).

Algorithms	***# LPs and their sizes***
Kohlberg [22] in 1972	One LP with $\mathrm{O}(2^n!)$ constraints
Owen [28] in 1974	One LP with $\mathrm{O}(4^n)$ constraints but the coefficients get large
Maschler et al. [27] in 1979	$\mathrm{O}(4^n)$ LPs, each with with $\mathrm{O}(2^n)$ rows and $n+1$ columns
Sankaran [32] in 1991	$\mathrm{O}(2^n)$ LPs, each with $\mathrm{O}(2^n)$ rows and $n+1$ columns
Potters et al. [31] in 1996	$n-1$ LPs, each with (2^n+n-1) rows and (2^n-1) columns
Our method (2012)	$n-1$ LPs, each with (2^n+n-1) rows and $(n+1)$ columns

Table 1: Comparison between our method and the literature.

- We provide numerical computation for large games with up to 100 players in the weighted voting games [1, 5] and up to 75 players in the coalitional skill games [2, 3]. This is a significant improvement compared to the literature where numerical results are shown for computing the nucleoli of games with at most 20 players.

In addition to these key contributions, we also apply a constraint generation framework for solving the large LPs. This gives hope to solving very large LPs as we don't have to rely on the simplex method

to solve large scale LPs when $n \geq 25$. The constraint generation algorithm is not new and has been applied successfully in many areas including finding the solutions of cooperative games (e.g. Caprara and Letchford [4]). Applying it to solving nested LPs though creates some challenging problems in keeping track of multiple discrete and continuous optimal solutions so that subsequent LPs can be formulated. Our approach is appropriate for large games and for combinatorial games where it is costly to calculate the characteristic values for all the possible coalitions. For example, in the flow games proposed by Kalai and Zemel [19], the value of a coalition is the maximum flow that can be sent through the subnetwork using edges in the coalition only. In this case, it is time consuming to calculate all the 2^n possible values. Instead, we can incorporate the maximum flow problem into the constraint generation problem that is then solved only if needed. We also demonstrate how this can be done in other games such as the voting games and the coalitional skills games.

The structure of this paper is as follows: Section 2.1 provides the list of notations used throughout the paper and a review of important solution concepts in cooperative game theory such as the core, the least core, and the nucleolus. We present the idea behind the nested LPs and their formulation in Sections 2.2 and 2.3 with an illustrative example. The focus of this paper starts from Section 2.4 where the subtle issues relating to the nested LPs formulation are discussed in detail. The main contribution of this paper lies in addressing these issues in Section 3. We present the framework for finding the nucleolus in Section 3.1 and the idea of finding optimal solutions with minimal tight sets in Section 3.3. Our algorithm requires solving at most $(n-1)$ large LPs, each having $(n+1)$ columns and (2^n+n-1) rows as shown in Section 3.4. Various numerical experiments are presented in Section 5 and conclusion is drawn in Section 6.

2 Finding the Nucleolus using Nested LPs

2.1 Review of Solution Concepts in Cooperative Game Theory and Notation

Let n be the number of players and let $\mathcal{N} = \{1, 2, \ldots, n\}$ be the set of all the players. A *coalition* $\mathcal{S}$ is a subset of the players, i.e. $\mathcal{S} \subset \mathcal{N}$. Let $\mathcal{C} \equiv 2^{\mathcal{N}}$ be the set of all the possible coalitions. The *characteristic function* $v : 2^{\mathcal{N}} \mapsto \mathbb{R}$ maps each coalition to a real number with $v(\mathcal{S})$ representing the payoff that coalition $\mathcal{S}$ is guaranteed to obtain if all players in $\mathcal{S}$ collaborate no matter what the other players do. A solution of the game $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ is a way to distribute the reward among the players, with x_i being the share for player i. Let us denote $\boldsymbol{x}(\mathcal{S}) = \sum_{i \in \mathcal{S}} x_i$. For each imputation $\boldsymbol{x}$, the *excess value* of a coalition $\mathcal{S}$ is defined as $e(\mathcal{S}, \boldsymbol{x}) = v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})$ which can be viewed as the level of dissatisfaction the players in coalition $\mathcal{S}$ feel over the proposed solution $\boldsymbol{x}$. Solution concepts for cooperative games include:

- An *imputation* is a solution $\boldsymbol{x}$ that satisfies $\sum_{i \in \mathcal{N}} x_i = v(\mathcal{N})$ and $x_i \geq v(i), \forall i \in \mathcal{N}$.
- The *core* of the game is the set of all imputations $\boldsymbol{x}$ such that $e(\mathcal{S}, \boldsymbol{x}) \leq 0, \ \forall \mathcal{S} \subset \mathcal{N}$.
- The ϵ-*core* is defined as the set of all imputations $\boldsymbol{x}$ such that $e(\mathcal{S}, \boldsymbol{x}) \leq \epsilon, \forall \mathcal{S} \subset \mathcal{N}$.

- *Least core*: The least core is the non-empty ϵ-core with ϵ being the smallest value.
- *Nucleolus*: For any imputation $\boldsymbol{x}$, let $\Theta(\boldsymbol{x}) = (\Theta_1(\boldsymbol{x}), \Theta_2(\boldsymbol{x}), \ldots, \Theta_{2^n}(\boldsymbol{x}))$ be the vector of all the 2^n excess values at $\boldsymbol{x}$ sorted in the decreasing order, i.e. $\Theta_i(\boldsymbol{x}) \geq \Theta_j(\boldsymbol{x})$ if $1 \leq i < j \leq 2^n$. Let us denote $\Theta(\boldsymbol{x}) <_L \Theta(\boldsymbol{y})$ if $\exists\ r$ such that $\Theta_i(\boldsymbol{x}) = \Theta_i(\boldsymbol{y}), \forall 1 \leq i < r$ and $\Theta_r(\boldsymbol{x}) < \Theta_r(\boldsymbol{y})$. Then $\boldsymbol{x}$ is the *nucleolus* if, $\Theta(\boldsymbol{x}) <_L \Theta(\boldsymbol{y}),\ \forall \boldsymbol{y} \neq \boldsymbol{x}$.

This section provides only a brief review of cooperative game theory. We refer the interested readers to Peleg and Sudhölter [29], Chalkiadakis et al. [5] for more thorough introduction of the topic.

2.2 The Idea behind Using Nested LPs

Consider the following minimax problem for finding the least core: $\min\limits_{\boldsymbol{x} \in \mathcal{I}} \left(\max\limits_{\mathcal{S} \subset \mathcal{N}} \{v(\mathcal{S}) - \boldsymbol{x}(S)\} \right)$. This problem finds an imputation $\boldsymbol{x}$ such that the worst excess value among all coalitions, i.e. $\max\limits_{\mathcal{S} \subset \mathcal{N}} \{v(\mathcal{S}) - \boldsymbol{x}(S)\}$, is minimized. By definition, the nucleolus must be a solution of this problem. Furthermore, let $\epsilon = \max\limits_{\mathcal{S} \subset \mathcal{N}} \{v(\mathcal{S}) - \boldsymbol{x}(S)\}$, then the problem can be reformulated as an LP as follows:

$$\min_{\boldsymbol{x} \in \mathcal{I}, \epsilon} \quad \{\epsilon \mid \epsilon + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \subset \mathcal{N}\}.$$

Before going into further details on the algorithm, we need to define the concept of **tight set** as it is directly related to the definition of lexicographical minimization and will be crucial in formulating the nested LPs.

Definition 1. *For any given $(\boldsymbol{x}, \epsilon)$, let us denote $\mathcal{T}(\boldsymbol{x})$ as the corresponding **tight set** of all those coalitions $\mathcal{S}$ such that the constraints $\epsilon + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S})$ is tight, i.e. $\epsilon + \boldsymbol{x}(\mathcal{S}) = v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{T}(\boldsymbol{x})$.*

The reformulated LP problem is not easy to solve because there is an exponentially large number of constraints. However, assume its optimal solution is $(\boldsymbol{x}^*, \epsilon^*)$. Then the lexicographical ordering of all the excess values of imputation $\boldsymbol{x}^*$ will have the form: $\{\epsilon^*, \epsilon^*, \ldots, \epsilon^*, \sigma, \ldots\}$, where the first $|\mathcal{T}(\boldsymbol{x}^*)|$ elements of the sequence are equal to ϵ^* and the following elements are at most σ, which is smaller than ϵ^*.

Solving this problem can provide us with the worst excess value that the nucleolus will produce. However, the LP problem can have multiple optimal solutions and we are not guaranteed that the optimal solution produced is the nucleolus. In addition, two distinct optimal solutions might have different tight sets and the tight sets might have different sizes. We only know that the nucleolus must correspond to an optimal solution $\boldsymbol{x}^*$ with the smallest tight set $\mathcal{T}(\boldsymbol{x}^*)$ (here, 'smallest' is in terms of size). Elkind and Pasechnik [10] show that the tight set with the smallest size is unique, i.e. if both $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$ produce the sets $\mathcal{T}(\boldsymbol{x}^*)$ and $\mathcal{T}(\boldsymbol{y}^*)$ which are smallest in sizes, then the two sets must be identical, i.e. $\mathcal{T}(\boldsymbol{x}^*) \equiv \mathcal{T}(\boldsymbol{y}^*)$ (this result can be proved by exploiting the linearity property of the problem). This means that choosing any optimal solution $\boldsymbol{x}^*$ with the smallest tight set would lead to the same minimal tight set $\mathcal{T}^*$ which will be used to formulate the subsequent LP. We will demonstrate how to find the minimal tight set $\mathcal{T}^*$ in Sections 3.2 and 3.3. In order to find the nucleolus among all

the imputations $\boldsymbol{x}^*$ with the same smallest set $\mathcal{T}^*$, we must then aim to minimize σ and repeat this procedure until x^* is unique. This can be done by solving another LP:

$$\min_{\boldsymbol{x}\in\mathcal{I},\sigma} \quad \{\sigma \mid \epsilon^* + \boldsymbol{x}(\mathcal{S}) = v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{T}^*,\ \sigma + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{C}\backslash\mathcal{T}^*\}.$$

The first set of constraints ensures that only candidate imputations with the first $|\mathcal{T}^*|$ worst excess values equal to ϵ^* (but not larger) are considered. The second set of constraints and the objective function aim to minimize the worst excess values among all the remaining coalitions. Solving this LP will produce for us another set of coalitions $\mathcal{T}^{*'}$ with the excess σ^*. If we keep doing this, we will reach the point where the optimal solution $\boldsymbol{x}^*$ is unique; that is, $\boldsymbol{x}^*$ is the nucleolus of the game.

2.3 Nested LPs Formulation for Finding the Nucleolus

The problem of finding the nucleolus can be formulated as nested LPs (a sequence of $\mathbf{LP_1}$, $\mathbf{LP_2}$, ...) as follows:

$$\mathbf{LP_1} := \min_{\boldsymbol{x}\in\mathcal{I},\epsilon_1} \quad \{\epsilon_1 \mid \epsilon_1 + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \subset \mathcal{N}\}.$$

Let $(\boldsymbol{x}_1, \epsilon_1^*)$ be an optimal solution. Notice that ϵ_1^* should be unique. However, it is possible to have multiple optimal solutions $\boldsymbol{x}_1$. Each $\boldsymbol{x}_1$ will correspond to a set $\mathcal{T}_1(\boldsymbol{x}_1)$ of all the coalitions $\mathcal{S}$ for which the inequality constraint $\epsilon_1^* + \boldsymbol{x}_1(\mathcal{S}) \geq v(\mathcal{S})$ is tight. Among all the optimal solutions and their corresponding tight sets, let $\mathcal{T}_1^*$ be the set with the smallest size. Suppose for now that we are able to solve $\mathbf{LP_1}$ and produce an optimal solution $(\boldsymbol{x}_1^*, \epsilon_1^*)$ with the minimal tight set, i.e. $\mathcal{T}_1(\boldsymbol{x}_1^*) \equiv \mathcal{T}_1^*$. We then solve the following LP:

$$\mathbf{LP_2} := \min_{\boldsymbol{x}\in\mathcal{I},\epsilon_2} \quad \{\epsilon_2 \mid \epsilon_1^* + \boldsymbol{x}(\mathcal{S}) = v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{T}_1^*,\ \epsilon_2 + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{C} \setminus \mathcal{T}_1^*\}.$$

For each $k \geq 2$, suppose $\mathbf{LP_k}$ produces $(\boldsymbol{x}_k^*, \epsilon_k^*, \mathcal{T}_k^*)$. Then $\mathbf{LP_{k+1}}$ is formulated as follows:

$$\begin{aligned}
\mathbf{LP_{k+1}} := \quad & \min_{\boldsymbol{x}\in\mathcal{I},\epsilon_{k+1}} \quad \epsilon_{k+1}, \\
& s.t. \quad \epsilon_r^* + \boldsymbol{x}(\mathcal{S}) = v(\mathcal{S}), \qquad \forall \mathcal{S} \in \mathcal{T}_r^*, \ \ \forall r \in \{1, .., k\}, && \text{(1a)} \\
& \qquad\quad \epsilon_{k+1} + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}), \qquad \forall \mathcal{S} \in \mathcal{C}\backslash\mathcal{H}_k^*, . && \text{(1b)}
\end{aligned}$$

where $\mathcal{H}_k^* = \cup_{r\in\{1,..,k\}}\mathcal{T}_r^*$. We repeat this process until $\mathbf{LP_k}$ produces a unique imputation $\boldsymbol{x}^*$. That imputation is the nucleolus. To demonstrate the nested LPs formulation, we consider the following simple three-player cooperative game example with the characteristic function:

$$v(\{1\}) = 1,\ v(\{2\}) = 2,\ v(\{3\}) = 5,\ v(\{1,2\}) = 6,\ v(\{1,3\}) = 7,\ v(\{2,3\}) = 8,\ v(\{1,2,3\}) = 12.$$

The set of all imputations is: $\mathcal{I} = \{(x_1, x_2, x_3) \ : \ x_1 + x_2 + x_3 = 12,\ x_1 \geq 1,\ x_2 \geq 2,\ x_3 \geq 5\}$, and is shown in the shaded area (the largest triangle) in Figure 1. The core of the game is:

$$\mathcal{C} = \{(x_1, x_2, x_3) : \quad x_1 + x_2 + x_3 = 12,\ x_1 \geq 1,\ x_2 \geq 2,\ x_3 \geq 5,\ x_1 + x_2 \geq 6,\ x_1 + x_3 \geq 7,\ x_2 + x_3 \geq 8\},$$

and is shown in the shaded trapezoid in Figure 1. $\mathbf{LP_1}$ is formulated as:

$$\min_{\boldsymbol{x},\epsilon} \quad \{\epsilon \mid x_1+\epsilon \geq 1,\ x_2+\epsilon \geq 2,\ x_3+\epsilon \geq 5,\ x_1+x_2+\epsilon \geq 6,\ x_1+x_3+\epsilon \geq 7,\ x_2+x_3+\epsilon \geq 8,$$
$$x_1+x_2+x_3+\epsilon \geq 12,\ \epsilon \geq 0,\ x_1+x_2+x_3 = 12,\ x_1 \geq 1,\ x_2 \geq 2,\ x_3 \geq 5\}.$$

The optimal value of $\mathbf{LP_1}$ is $\epsilon_1^* = 0$ and the set of all the optimal solutions (the least core solutions) is exactly the core. Solving $\mathbf{LP_1}$ will produce for us a least core solution. However, depending on the solver we use, we might end up with one of the extreme points (e.g. if we use the simplex method) or a relative interior point (e.g. if we use an interior point method). The tight sets that correspond to these optimal solutions are:

$$\mathcal{T}_1(\boldsymbol{x}) = \begin{cases} \{\{\emptyset\};\{1,2,3\};\{3\};\{2,3\}\} & \text{if } \boldsymbol{x} = (4,3,5), \\ \{\{\emptyset\};\{1,2,3\};\{2\};\{1,2\};\{2,3\}\} & \text{if } \boldsymbol{x} = (4,2,6), \\ \{\{\emptyset\};\{1,2,3\};\{1\};\{1,2\};\{1,3\}\} & \text{if } \boldsymbol{x} = (1,5,6), \\ \{\{\emptyset\};\{1,2,3\};\{3\};\{1,3\}\} & \text{if } \boldsymbol{x} = (2,5,5), \\ \{\{\emptyset\};\{1,2,3\};\{2,3\}\} & \text{if } \boldsymbol{x} = \alpha(4,3,5)+(1-\alpha)(4,2,6),\ \text{with } 0<\alpha<1, \\ \{\{\emptyset\};\{1,2,3\};\{1,2\}\} & \text{if } \boldsymbol{x} = \alpha(4,2,6)+(1-\alpha)(1,5,6),\ \text{with } 0<\alpha<1, \\ \{\{\emptyset\};\{1,2,3\};\{1,3\}\} & \text{if } \boldsymbol{x} = \alpha(1,5,6)+(1-\alpha)(2,5,5),\ \text{with } 0<\alpha<1, \\ \{\{\emptyset\};\{1,2,3\};\{3\}\} & \text{if } \boldsymbol{x} = \alpha(2,5,5)+(1-\alpha)(4,3,5),\ \text{with } 0<\alpha<1, \\ \{\{\emptyset\};\{1,2,3\}\} & \text{if } \boldsymbol{x} \in int(conv((4,3,5);(4,2,6);(1,5,6);(2,5,5))). \end{cases}$$

In this case, the minimal tight set is $\mathcal{T}_1^* = \{\{\emptyset\};\{1,2,3\}\}$ when $\boldsymbol{x}$ belongs to the interior of the trapezoid or the interior of the line segment $\alpha(2,5,5)+(1-\alpha)(4,3,5)$ with $0<\alpha<1$. The problem of how to find an optimal solution that corresponds to the minimal tight set will be dealt with in Section 3.3. However, suppose for now that we are able to obtain this minimal tight set. Then we use $\mathcal{T}_1^*$ to formulate $\mathbf{LP_2}$ as follows:

$$\min_{\boldsymbol{x},\epsilon} \quad \{\epsilon \mid x_1+x_2+x_3 = 12,\ x_1+\epsilon \geq 1,\ x_2+\epsilon \geq 2,\ x_3+\epsilon \geq 5,$$
$$x_1+x_2+\epsilon \geq 6,\ x_1+x_3+\epsilon \geq 7,\ x_2+x_3+\epsilon \geq 8,\ x_1 \geq 1,\ x_2 \geq 2,\ x_3 \geq 5\}.$$

The optimal value is $\epsilon_2^* = -0.5$ and the optimal solutions of $\mathbf{LP_2}$ are those points in the line segment connecting $\boldsymbol{x} = (2, 4.5, 5.5)$ and $\boldsymbol{y} = (3.5, 3, 5.5)$, i.e. those points with the form $\alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}$ where $0 \leq \alpha \leq 1$. Solving $\mathbf{LP_2}$ will produce a solution that belongs to this line segment. From the minimal tight set $\mathcal{T}_2^* = \{\{3\};\{1,2\}\}$ we can formulate and solve $\mathbf{LP_3}$ and obtain the optimal value of $\epsilon_3^* = -1.25$ and a unique solution of $\boldsymbol{x}^* = (2.75, 3.75, 5.5)$ with the tight set $T_3^* = \{\{1,3\};\{2,3\}\}$. This is the nucleolus and we can stop the algorithm.

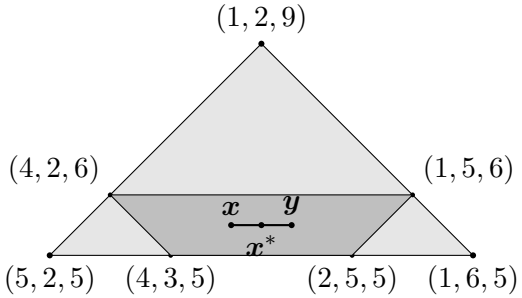

Figure 1: Example of a cooperative game with three players. The largest triangle is the domain of the imputations. The shaded trapezoid is the domain of the core, which coincides with the least core for this particular game, i.e. the optimal solutions of $\mathbf{LP_1}$. The line segment between $\boldsymbol{x}$ and $\boldsymbol{y}$ is the domain of all the optimal solutions of $\mathbf{LP_2}$. The nucleolus is $\boldsymbol{x}^*$, which is the unique solution of $\mathbf{LP_3}$.

2.4 Issues with the Nested LPs Formulation

From the simple example presented in Section 2.3, we have some observations on the properties of $\mathbf{LP_k}$. First of all, ϵ_k^* is always unique but there might be multiple optimal solutions $\boldsymbol{x}_k$. Second, among all the optimal solutions $\boldsymbol{x}_k$, there might be more than one solution whose tight set is smallest in size. However, all these optimal solutions share the same unique tight set. It is also interesting to notice that, based on the construction of the nested LPs, any optimal solution $\boldsymbol{x}_{k+1}$ of $\mathbf{LP_{k+1}}$ is also an optimal solution of $\mathbf{LP_k}$, $\mathbf{LP_{k-1}}$,...$\mathbf{LP_1}$. In addition, $\boldsymbol{x}_{k+1}$ produces the smallest tight sets in all these k LPs. The nested LPs formulation provides us with a general procedure for finding the nucleolus. However, there are many practical issues that need to be addressed:

Issue 1: *For a given imputation $\boldsymbol{x}$, how can we find the set $\mathcal{T}_k(\boldsymbol{x})$ of all the coalitions that produce the worst excess values? This issue is equivalent to the problem of finding multiple discrete solutions of the problem:* $\max_{\mathcal{S}\subset\mathcal{N}} \{v(\mathcal{S}) - \boldsymbol{x}(S)\}$.

Issue 2: $\mathbf{LP_k}$ *might have multiple optimal imputations $\boldsymbol{x}$, each of which corresponds to a tight set $\mathcal{T}_k(\boldsymbol{x})$. How can we find an optimal solution $\boldsymbol{x}$ with the smallest $|\mathcal{T}_k(\boldsymbol{x})|$?*

Issue 3: *The nested LPs presented might require solving as many LPs as there are distinguishing level ϵ_k^* and this could be exponentially large for some games. How can we modify it to bound the number of LPs to at most $(n-1)$?*

Issue 4: *Related to the second issue with multiple tight coalitions, the problem becomes even more challenging when the tight sets have exponential sizes (this occurs in many games such as the weighted voting games). If this is the case, how can we formulate and solve the subsequent LPs given that we might not have the resources to compute the entire set $\mathcal{T}_k^*$?*

As far as we have understood, although the nested LPs formulation was formulated by Kopelowitz [23] in 1967 and various reformulations have been presented, very limited numerical tests have been performed. In addition, all of these experiments are only for small games, i.e. Potters et al. [31] provide numerical tests for games with at most 10 players, Derks and Kuipers [9] and Fromen [13] improve

existing methods to provide numerical tests for games with no more than 20 players. We presume that there are two main reasons behind this. First of all, the LPs has 2^n constraints and hence are difficult to solve in practice. *The second and more critical reason lies on the difficulty of handling the multiple optimal solutions in each LP* (Issue 2).

Potters et al. [31] and Derks and Kuipers [9] recognize these four aforementioned issues and attempt to resolve them using a prolonged simplex method. However, their method does not scale well because of the large LPs involved. Issue 3 has been recognized in the literature and the conventional stopping condition is when the tight constraints produce a unique solution. The number of LPs to be solved is bounded by the number of players in [9, 31] thanks to their special way of avoiding having to solve redundant LPs. Issue 4 has been recognized by Elkind and Pasechnik [10]. Under these situations, their ellipsoid method requires the assumption that the size of $\mathcal{T}_k^*$ is known. Our paper deals issue 4 without this assumption. This is done by replacing $\mathcal{T}_k^*$ with a representative set with size at most n. This not only helps to resolve issue 4, but also makes the LPs formulation smaller, and thus easier to store and solve.

3 Computational Approach for Solving the Nested LPs

The nested LPs formulation presented in the literature is usually in the form of Model (1) described in Section 2.3. We will first reformulate these LPs slightly. Let $(\boldsymbol{x}_k^*, \epsilon_k^*)$ be an optimal solution from $\mathbf{LP_k}$ that has the minimal tight set $\mathcal{T}_k(\boldsymbol{x}_k^*) = \mathcal{T}_k^*$. Let $(\boldsymbol{x}, \epsilon_{k+1})$ be an optimal solution of $\mathbf{LP_{k+1}}$. Then equality constraints (1a) in $\mathbf{LP_{k+1}}$ requires $\epsilon_r^* + \boldsymbol{x}(\mathcal{S}) = v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{T}_r^*,\ \forall r \in \{1, .., k\}$. Since $(\boldsymbol{x}_k^*, \epsilon_r^*, \mathcal{T}_r^*)$ is an optimal solution from $\mathbf{LP_r}$, we also have $\epsilon_r^* + \boldsymbol{x}_k^*(\mathcal{S}) = v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{T}_r^*,\ \forall r \in \{1, .., k\}$. Subtracting one equality from the other, we obtain $(\boldsymbol{x} - \boldsymbol{x}_k^*)(\mathcal{S}) = 0,\ \forall \mathcal{S} \in \mathcal{T}_r^*,\ \forall r \in \{1, .., k\}$. Thus, $(\boldsymbol{x} - \boldsymbol{x}_k^*)(\mathcal{S}) = 0,\ \forall \mathcal{S} \in \mathcal{H}_k^*$, i.e. we restrict $\boldsymbol{x} \in \left\{\boldsymbol{x}_k^* + null(\mathcal{H}_k^*)\right\}$, where $null(\mathcal{H}_k^*)$ denotes the null space of all the coalition vectors in $\mathcal{H}_k^*$. From this, $\mathbf{LP_{k+1}}$ can be reformulated as:

$$
\begin{aligned}
\mathbf{LP_{k+1}} := \quad & \min_{\boldsymbol{x} \in \mathcal{I}, \epsilon_{k+1}} && \epsilon_{k+1}, \\
& s.t. && (\boldsymbol{x} - \boldsymbol{x}_k^*)(\mathcal{S}) = 0, \qquad \forall \mathcal{S} \in \mathcal{H}_k^*, && \text{(2a)} \\
& && \epsilon_{k+1} + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}), \qquad \forall \mathcal{S} \in \mathcal{C} \backslash \mathcal{H}_k^*. && \text{(2b)}
\end{aligned}
$$

This reformulation provides us with a more elegant form. The storage required in representing the formulation is also smaller, i.e. only needs $\mathcal{H}_k^*$ from previous LPs instead of $(\epsilon_r^*, \mathcal{T}_r^*)$ for all $r \in \{1, .., k\}$. This formulation also helps when dealing with large tight sets described as the fifth issue in Section 2.4. In this case, constraint (2a) can be expressed using $rank(\mathcal{H}_k^*)$ number of equalities which is always smaller than n. Here, $rank(\mathcal{H}_k^*)$ denotes the number of linearly independent vectors in $\mathcal{H}_k^*$. Notice that the nested LPs formulation shown requires solving as many large LPs as there are distinct values of ϵ_k^*. It is possible to resolve this using the ideas in Potters et al. [31] and Derks and Kuipers [9] to avoid solving redundant LPs. Here, the set of constraints in (2b) are replaced by $\epsilon_{k+1} + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}), \forall \mathcal{S} \notin span(\mathcal{H}_k^*)$, where $span(\mathcal{H}_k^*)$ denotes the linear span of vectors in $\mathcal{H}_k^*$. Hence the rank of $\mathcal{H}_k^*$ keeps increasing after solving every LP. Details of this will be presented in Section 3.4.

3.1 Algorithm for Finding the Nucleolus

Algorithm 1 presents the complete algorithm for finding the nucleolus. The algorithm iteratively solves $\mathbf{LP_k}$ to obtain an optimal solution (i.e. Step 2), find its representative set in Step 3, and find an optimal solution with the minimal representative set in Step 4. This process is repeated at most n times until the minimal tight set has full rank. A demonstration of how Algorithm 1 works on a small flow game is presented in the numerical results in Section 5.3.

Algorithm 1: Algorithm for finding the nucleolus $\boldsymbol{x}^*$ of cooperative game $(\mathcal{N}, v)$

1. Initialization: Set the initial tight set $\mathcal{H}_0^*$ to include only the identity vector $\boldsymbol{e}$, i.e. $\mathcal{H}_0^* = \{\boldsymbol{e}\}$;
for *iteration* $k = 1...n$ **do**
  2. Solve $\mathbf{LP_k}$ and produce an optimal solution $(\boldsymbol{x}^*, \epsilon_k^*)$;
  3. Find the representative set $\mathcal{R}(\boldsymbol{x}^*)$ of $\mathcal{T}(\boldsymbol{x}^*)$;
  4. Find improving optimal solution $\boldsymbol{x}^*$ with $\mathcal{R}(\boldsymbol{x}^*)$ having the smallest rank (i.e. minimal representing set);
  if $\mathcal{R}(\boldsymbol{x}^*)$ *has full rank* **then**
    5. Output $\boldsymbol{x}^*$ as the nucleolus and stop the algorithm;
  end
end

In Step 2, we need to solve $\mathbf{LP_k}$ efficiently. This task is feasible for any cooperative games with the number of players less than around 20 since linear programming solvers such as the Simplex method can handle these LP efficiently. For larger games, we need to make an assumption that the characteristic function has some special forms that allows us to exploit and solve $\mathbf{LP_k}$. This assumption is quite reasonable in the quest for finding the nucleolus because, without it, even solving for the core or the least core has already been difficult. We demonstrate this possibility through some combinatorial games to which the well-known constraint generation (CG) approach can be applied to solve $\mathbf{LP_k}$.

The CG algorithm (sometimes referred to as the delayed constraint generation or cutting plane method) has been successfully applied to problems involving an exponentially large number of constraints and it could potentially be used for finding the least core. The idea of the CG algorithm is to start with a restricted LP problem and then to check whether the optimal solution of the restricted problem satisfies all the constraints in the original problem. If that is the case, then the solution to the restricted problem is also an optimal solution of the original problem. Otherwise, we have identified a violating constraint and that can be added to the restricted problem. It is noted, however, that applying the CG algorithm to find the nucleolus is not straightforward since dealing with multiple optimal solutions and the tight sets, i.e. the first and the second issues presented in Section 2.4, becomes even more challenging to resolve within the CG framework. The CG algorithm often returns only a small subset of all the tight constraints because many of these constraints are relaxed in the restricted problem. How can we assess the size of the tight set of the optimal solution obtained against that of other optimal solutions? Fortunately, these issues are resolvable in the CG framework and are presented in Sections

3.2 and 3.3. We provide some further details about the CG algorithm in Appendix B.

We found many references in the literature that include only Step 2 when presenting the algorithm for finding the nucleolus. This seems to create a common perception within the research community that the nucleolus can be found by solving a nested LPs (without concerning about which optimal solution in each LP to choose). As we have shown in the simple example in Section 2.3 and the list of issues presented in Section 2.4, choosing the right optimal solution with the minimal tight set is crutially needed for formulating subsequent LPs and hence Steps 3-4 described in Algorithm 1 are needed. In fact, handling multiple optimal solutions is the major difficulty in computing the nucleolus and we can only realize this through doing the actual numerical computation. The focus of the paper is to resolve issues 1-4 listed in Section 2.4 which will be presented in the Sections 3.2-3.5.

The algorithm for finding the nucleolus presented in this paper involves solving several sub-problems. These sub-problems are referred to throughout the paper by a list of acronyms that are included in Appendix A.

3.2 Dealing with Issue 1: Finding Tight Coalitions

When solving $\mathbf{LP_k}$, we stop at the point when the relaxed LP produces the same result as the **CG** does, i.e. when the lower bound (from the relaxed LP) is equal to the upper bound (from the **CG**). Let $(\boldsymbol{x}, \epsilon_k^*)$ be an optimal solution found. Here, we use the notation $(\boldsymbol{x}, \epsilon_k^*)$ instead of $(\boldsymbol{x}_k^*, \epsilon_k^*)$ since there could be multiple optimal solutions $\boldsymbol{x}$ and what we obtained might not have the minimal tight set. The relaxed LP will have a number of constraints, $\epsilon_k + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S})$, that are tight. These correspond to coalitions with the worst excess values. These tight coalitions are often only a subset of $\mathcal{T}_k(\boldsymbol{x})$ and we need to find the remaining tight constraints that have not been in the relaxed LP. This problem is equivalent to finding the remaining coalitions $\mathcal{S}$ that solve $v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S}) = \epsilon_k^*$, which is also equivalent to solving the **CG** problem, $\max_{\mathcal{S}\in\mathcal{C}\setminus\mathcal{F}} \{v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})\}$, where $\mathcal{F}$ is the set of tight constraints that we have identified so far. We notice that each coalition $\mathcal{S}$ can be represented as an indicator vector $\boldsymbol{z} = \{z_1, z_2, \ldots, z_n\}$ where $z_i = 1$ means player i is in the coalition $\mathcal{S}$ and $z_i = 0$ means otherwise. Then, for each imputation $\boldsymbol{x}$, we have $\boldsymbol{x}(\mathcal{S}) = \boldsymbol{x}^t\boldsymbol{z}$, which is a linear function of $\boldsymbol{z}$. Suppose also that $v(\mathcal{S})$ can be expressed as a function of $\boldsymbol{z}$, i.e. $v(\mathcal{S}) \equiv v(\boldsymbol{z})$. Let $\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^K\}$ be the set of tight coalitions found so far. To find other tight coalitions, we need to solve the following problem: $\max\limits_{\boldsymbol{z}\in\{0,1\}^n} \{v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} \mid \boldsymbol{z} \notin \{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^K\}\}$. The constraint $\boldsymbol{z} \neq \boldsymbol{z}^j$ is equivalent to eliminating previous solution $\boldsymbol{z}^j$ from the feasible space. This can be done by generating a cut to separate this point from the remaining as follows. We notice that $\boldsymbol{z} \neq \boldsymbol{z}^j$ is equivalent to $(2\boldsymbol{z} - \boldsymbol{e}) \neq (2\boldsymbol{z}^j - \boldsymbol{e})$ where $\boldsymbol{e}$ is the identity vector. Notice that both $(2\boldsymbol{z} - \boldsymbol{e})$ and $(2\boldsymbol{z}^j - \boldsymbol{e})$ have elements equal to -1 or 1 and hence they are different if and only if $(2\boldsymbol{z}^j - \boldsymbol{e})^t(2\boldsymbol{z} - \boldsymbol{e}) \leq n - 2$. This is equivalent to $(2\boldsymbol{z}^j - \boldsymbol{e})^t\boldsymbol{z} \leq \boldsymbol{e}^t\boldsymbol{z}^j - 1$, which is a linear constraint on $\boldsymbol{z}$ (notice that $\boldsymbol{z}^j$ is given). The **CG** problem becomes:

$$\max_{\boldsymbol{z}\notin\mathcal{H}_{k-1}^*} \quad \{v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} \mid (2\boldsymbol{z}^j - \boldsymbol{e})^t\boldsymbol{z} \leq \boldsymbol{e}^t\boldsymbol{z}^j - 1, \ \ \forall j = \{1, .., K\}\}.$$

Notice that when solving the constraint generation problem at $\mathbf{LP_k}$, we need to find the worst coalition $\boldsymbol{z} \notin \mathcal{H}_{k-1}^*$. Instead of adding an individual cut for each coalition $\mathcal{S} \in \mathcal{H}_{k-1}^*$, a single 'aggregated

cut' can be generated as follows: $v(\boldsymbol{z})-\boldsymbol{x}^*_{k-1}(\boldsymbol{z}) < \epsilon^*_{k-1}$. This follows since $\mathcal{H}^*_{k-1}$ is the set of all coalitions whose excess values are at least ϵ^*_{k-1}. Depending on the problem structure and the form of function $v(\boldsymbol{z})$, the problem might be easy or difficult to solve. For example, if $v(\boldsymbol{z})$ can be transformed into a linear function of $\boldsymbol{z}$ as we will show later in Section 4.2 for the case of the weighted voting game, **CG** becomes a mixed integer linear programming problem. If $v(\boldsymbol{z})$ is quadratic, the problem is a quadratic assignment problem. From this, we show that the weighted voting game can be solved in reasonable time for games with up to 100 players.

3.3 Dealing with Issue 2: Finding Imputation Solutions $\boldsymbol{x}^*$ with Smallest Tight Set

Let $(\boldsymbol{x}^*_k, \epsilon^*_k)$ be an optimal solution of $\mathbf{LP_k}$ with the smallest size set $\mathcal{T}^*_k$. When solving $\mathbf{LP_k}$ using a standard LP solver or the constraint generation method, suppose we obtain an optimal solution $(\boldsymbol{x}, \epsilon^*_k)$ whose tight set $\mathcal{T}_k(\boldsymbol{x})$ might be larger in size compared to $\mathcal{T}^*_k$. If this is the case, we cannot take $(\boldsymbol{x}, \epsilon^*_k, \mathcal{T}_k(\boldsymbol{x}))$ to construct the subsequent LPs since this would not lead to a minimum lexicographical ordering of excess values. However, from this solution, we can find an improved imputation with smaller tight set as will be described in Theorem 1. Before presenting this theorem, we first describe some additional notation. Let $\mathcal{H}_k(\boldsymbol{x}) = \mathcal{H}^*_{k-1} \cup \mathcal{T}_k(\boldsymbol{x})$ be the set of all tight coalitions that have been identified so far. Notice that $\mathcal{H}_k(\boldsymbol{x}) \neq \mathcal{H}^*_k$ unless $\mathcal{T}_k(\boldsymbol{x}) \equiv \mathcal{T}^*_k$. Let σ be the $(k+1)^{th}$ excess value that corresponds to $\boldsymbol{x}$, i.e., $\sigma = \max_{\mathcal{S}\notin\mathcal{H}_k(\boldsymbol{x})}\{v(\mathcal{S})-\boldsymbol{x}(\mathcal{S})\}$. Then we have: $v(\mathcal{S})-\boldsymbol{x}(\mathcal{S}) \leq \sigma < \epsilon^*_k$, $\forall\, \mathcal{S} \in \mathcal{C}\backslash\mathcal{H}^*_k$. Let $\delta = \frac{1}{n+1}(\epsilon^*_k - \sigma) > 0$. We have the following theorem:

Theorem 1. *Let $(\boldsymbol{x}, \epsilon^*_k)$ be any optimal solution of $\mathbf{LP_k}$ with the tight set $\mathcal{T}_k(\boldsymbol{x})$, then the following LP:*

$$
\begin{aligned}
\mathbf{FBOS} := \min_{\boldsymbol{y}\in\mathcal{I}} \quad & \sum_{\mathcal{S}\in\mathcal{T}_k(\boldsymbol{x})} [v(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})], \\
s.t. \quad & (\boldsymbol{y} - \boldsymbol{x}^*_{k-1})(\mathcal{S}) = 0, \qquad \forall \mathcal{S} \in \mathcal{H}^*_{k-1}, && \text{(3a)} \\
& (\boldsymbol{y} - \boldsymbol{x})(\mathcal{S}) \geq 0, \qquad \forall \mathcal{S} \in \mathcal{T}_k(\boldsymbol{x}), && \text{(3b)} \\
& |y_i - x_i| \leq \delta, \ \forall\ i = 1, .., n, && \text{(3c)}
\end{aligned}
$$

always has an optimal solution $\boldsymbol{y}^*$. In addition, we can conclude that:

- *$(\boldsymbol{x}, \epsilon^*_k)$ is an optimal solution of $\mathbf{LP_k}$ with the minimal tight set if the optimal objective value of* **FBOS** *is equal to $|\mathcal{T}_k(\boldsymbol{x})|\epsilon^*_k$, and*
- *$(\boldsymbol{y}^*, \epsilon^*_k)$ is an optimal solution of $\mathbf{LP_k}$ with a smaller tight set compared to $\boldsymbol{x}$ otherwise.*

Proof. The main idea in this theorem is to observe that, if $|\mathcal{T}_k(\boldsymbol{x})| > |\mathcal{T}^*_k|$, then there must exist another imputation $\boldsymbol{y}$ that has smaller excess value than $\boldsymbol{x}$ on at least one coalition $\mathcal{S} \in \mathcal{T}_k(\boldsymbol{x})$, i.e. $e(\mathcal{S}, \boldsymbol{y}) < e(\mathcal{S}, \boldsymbol{x})$. This is equivalent to saying that $\boldsymbol{x}$ is not an optimizer of the following LP:

$$
\min_{\boldsymbol{y}\in\mathcal{I}} \quad \sum_{\mathcal{S}\in\mathcal{T}_k(\boldsymbol{x})} [v(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})],
$$

However, the optimizer of this LP does not guarantee to be better than $\boldsymbol{x}$ on all coalitions, especially those coalitions $S \notin \mathcal{T}_k(\boldsymbol{x})$. The additional constraints in **FBOS** are included for this purpose. The first constraint, $(\boldsymbol{y} - \boldsymbol{x}^*_{k-1})(\mathcal{S}) = 0, \forall \mathcal{S} \in \mathcal{H}^*_{k-1}$, guarantees that $\boldsymbol{y}$ does not change the worst $|\mathcal{H}^*_{k-1}|$ excess values that have been identified by $\mathbf{LP_1}, \ldots, \mathbf{LP_{k-1}}$. The second constraint, $(\boldsymbol{y}-\boldsymbol{x})(\mathcal{S}) \geq 0, \forall \mathcal{S} \in \mathcal{T}_k(\boldsymbol{x})$, guarantees that the excess vector $\Phi(\boldsymbol{y})$ is at least as good as $\Phi(\boldsymbol{x})$ for all elements from coalitions $\mathcal{S} \in \mathcal{T}_k(\boldsymbol{x})$. The last constraint, $|y_i - x_i| \leq \delta,\ \forall\ i = 1, .., n$, restricts $\boldsymbol{y}$ to a polyhedron (a box) that contains $\boldsymbol{x}$ in its interior. From the choice of the box size δ, we can guarantee that the excess values $e(\mathcal{S}, \boldsymbol{y})$ are smaller than ϵ^*_k for all the remaining coalitions $\mathcal{S} \in \mathcal{C} \backslash \mathcal{H}_k(\boldsymbol{x})$ as shown below.

$$\begin{aligned}(v(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})) &= (v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})) + (\boldsymbol{x}(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})) \\ &\leq \sigma + \sum_{i=1}^{n} |y_i - x_i| \leq \sigma + \frac{n}{n+1}(\epsilon^*_k - \sigma) < \sigma + \epsilon^*_k - \sigma = \epsilon^*_k.\end{aligned}$$

Combining all the three constraints, we guarantee that any feasible solution of **FBOS** isat least as good as $\boldsymbol{x}$ lexicographically with regards to the first k level of worst excess levels, i.e. up to $\mathbf{LP_k}$. Since the problem **FBOS** is bounded with at least one feasible solution $\boldsymbol{x}$, it must have an optimal solution $\boldsymbol{y}^*$. If the optimal objective value is smaller than $|\mathcal{T}_k(\boldsymbol{x})|\epsilon^*_k$, then there must be at least one coalition $\mathcal{S} \in \mathcal{T}_k(\boldsymbol{x})$ such that $v(\mathcal{S}) - \boldsymbol{y}^*(\mathcal{S}) < \epsilon^*_k$. Hence $\boldsymbol{y}^*$ is better than $\boldsymbol{x}$ lexicographically. To complete the proof of this theorem, we need to prove that, if $\boldsymbol{x}$ does not have a minimal tight set, i.e. $|\mathcal{T}_k(\boldsymbol{x})| > |\mathcal{T}^*_k|$, then the optimal objective value must be smaller than $|\mathcal{T}_k(\boldsymbol{x})|\epsilon^*_k$. Indeed, we can construct a feasible solution of **FBOS** with an objective value equal to $|\mathcal{T}_k(\boldsymbol{x})|\epsilon^*_k$.

We notice a result from Lemma 1 in Elkind and Pasechnik [10] which shows that the minimal tight set $\mathcal{T}^*_k$ is unique. In the proof of this lemma, the authors also show that if $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are two optimal solutions of $\mathbf{LP_k}$ with the corresponding tight sets $\mathcal{T}(\boldsymbol{x}^1)$ and $\mathcal{T}(\boldsymbol{x}^2)$, then any point $\boldsymbol{\omega} = \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{y}$ with $0 < \alpha < 1$ will be an optimal solution with a tight set equal to $\mathcal{T}(\boldsymbol{x}^1) \cap \mathcal{T}(\boldsymbol{x}^2)$ (the proof is based on the linearity of the problem). From this result, we can see that any point $\boldsymbol{y} = \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{x}^*_k$ with $0 < \alpha \leq 1$ would have the smallest tight set. These properties are demonstrated in Figure 2.

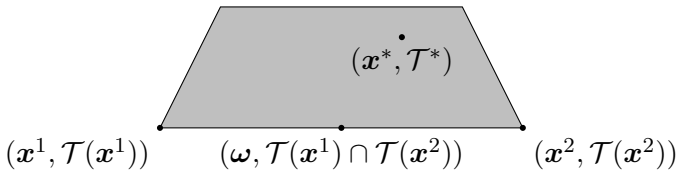

Figure 2: Domain of optimal solutions of $\mathbf{LP_k}$ and their associated tight sets. The domain is convex and all optimal solutions in the relative interior of the domain share the same tight set which is also smallest in size. $\boldsymbol{x}^1$ and $\boldsymbol{x}^2$ are two extreme points of the domain with the corresponding tight set $\mathcal{T}(\boldsymbol{x}^1)$ and $\mathcal{T}(\boldsymbol{x}^2)$. Any point $\boldsymbol{\omega} = \alpha\boldsymbol{x}^1 + (1-\alpha)\boldsymbol{x}^2$ with $0 < \alpha < 1$ will have a tight set equal to $\mathcal{T}(\boldsymbol{x}^1) \cap \mathcal{T}(\boldsymbol{x}^2)$. If the domain contains a unique point, then that point is the nucleolus.

Going back to our theorem, we notice that $\boldsymbol{x}^*_k$ satisfies constraints (3a-3b) of **FBOS**. Thus, if we choose a small enough α, we can obtain an optimal solution $\boldsymbol{y} = \alpha\boldsymbol{x} + (1-\alpha)\boldsymbol{x}^*_k$ that satisfies constraints

(3c). By linearity, $\boldsymbol{y}$ also satisfies constraints (3a-3b) and hence $\boldsymbol{y}$ is a feasible solution of **FBOS**. In addition, $\boldsymbol{y}$ has the smallest tight set since $\mathcal{T}_k(\boldsymbol{y}) = \mathcal{T}_k(\boldsymbol{x}) \cap \mathcal{T}_k(\boldsymbol{x}_k^*) = \mathcal{T}_k(\boldsymbol{x}_k^*)$. Thus, $\boldsymbol{y}$ is an optimal solution of **FBOS** and with the minimal tight set. The proof of the theorem is complete. □

The geometric view of Theorem 1 is illustrated in Figure 3. Let the cone $\mathcal{K}$ be defined as the intersec-

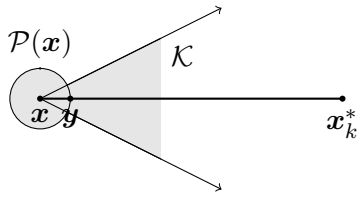

Figure 3: Figure demonstrating the result from Theorem 1. Any $\boldsymbol{y}$ lying in the interior of the line segment connecting $\boldsymbol{x}$ and $\boldsymbol{x}_k^*$ has the minimal tight set w.r.t $\mathbf{LP_k}$. Cone $\mathcal{K}$ is the set of all imputations that are at least as good as $\boldsymbol{x}$ when comparing excess values lexicographically on all coalitions belonging to $\mathcal{H}_{k-1}^* \cup \mathcal{T}_k(\boldsymbol{x})$. Imputations within the domain $\mathcal{P}(\boldsymbol{x})$ would have excess values smaller than ϵ_k^* for all those coalitions not belonging to $\mathcal{H}_{k-1}^* \cup \mathcal{T}_k(\boldsymbol{x})$. Thus, all imputations belonging to $\mathcal{K} \cap \mathcal{P}(\boldsymbol{x})$ would be at least as good as $\boldsymbol{x}$ lexicographically w.r.t $\mathbf{LP_k}$.

tion of the set $\mathcal{I}$ of all imputations, the affine subspace $\{\boldsymbol{x}_{k-1} + null(\mathcal{H}_{k-1}^*)\}$ that ensures the worst excess values from previous LPs do not change, and the dual cone $\mathcal{C}^*(\mathcal{T}_k(\boldsymbol{x})) \equiv \{\boldsymbol{y} \in \mathbb{R}^n \mid \langle \boldsymbol{y}, \boldsymbol{\sigma} \rangle \geq 0,\ \forall \boldsymbol{\sigma} \in \mathcal{T}_k(\boldsymbol{x})\}$ to ensure that $\boldsymbol{y}$ is at least as good as $\boldsymbol{x}$ when considering coalitions within $\mathcal{T}_k(\boldsymbol{x})$, i.e.,

$$\mathcal{K} = \mathcal{I} \cap \{\boldsymbol{x}_{k-1} + null(\mathcal{H}_{k-1}^*)\} \cap \mathcal{C}^*(\mathcal{T}_k(\boldsymbol{x})).$$

Then $\mathcal{K}$ contains $\boldsymbol{x}_k^*$. Here, $null(\mathcal{H}_{k-1}^*)$ denotes the null space of vectors in $\mathcal{H}_{k-1}^*$. Let $\mathcal{P}(\boldsymbol{x})$ be any domain that contains $\boldsymbol{x}$ in its interior. The idea is that once the domain $\mathcal{P}(\boldsymbol{x})$ is small enough, the excess values of all coalitions not belonging to the tight set $\mathcal{H}_k(\boldsymbol{x})$ do not change much to exceed ϵ_k^*. The intersection of $\mathcal{P}(\boldsymbol{x})$ and $\mathcal{K}$ provides us with imputations that are at least as good as $\boldsymbol{x}$ lexicographically with respect to the first k worst excess levels.

Remark: Notice that **FBOS** is an LP with n decision variables and with $|\mathcal{H}_k(\boldsymbol{x})| + 1$ equality constraints and $(3n + |\mathcal{T}_k(\boldsymbol{x})|)$ inequality constraints[1], and can be solved very efficiently. Notice also that the **FBOS** algorithm imposes the constraint of $\boldsymbol{y} \in \mathcal{P}(\boldsymbol{x})$ for a small domain $\mathcal{P}(\boldsymbol{x})$ to provide a theoretical guarantee. However, once we have found an improved solution $\boldsymbol{y}$, we should extrapolate it from $\boldsymbol{x}$ to obtain a new improved solution that is as close to the center of the domain of all the improved optimal solutions as possible. This can be done by performing a line search on all $\hat{\boldsymbol{y}} = \alpha * \boldsymbol{y} - (\alpha - 1)\boldsymbol{x}$ for the maximum α_{max} such that $\hat{\boldsymbol{y}}$ is still an improved optimal solution. Once we have obtained α_{max}, we can then choose the improved optimal solution as $\bar{\boldsymbol{y}} = \alpha_{max}/2 * \boldsymbol{y} - (\alpha_{max}/2 - 1)\boldsymbol{x}$. This extrapolation step achieves two objectives: First of all, it provides us an improved optimal solution that is far enough from $\boldsymbol{x}$ to avoid numerical error. Second, $\bar{\boldsymbol{y}}$ is more likely to be closer to the nucleolus and hence can speed up our algorithm.

[1]These include n inequality constraints $\boldsymbol{y} \in \mathcal{I}$ for individual rationality, $|\mathcal{T}_k(\boldsymbol{x})|$ inequalities from 3b, and $2n$ inequality constraints from 3c.

3.4 Dealing with Issue 3: Bounding the Number of LPs to be Solved by $(n-1)$

We notice that if we solve the nested LP formulation described directly, we might end up solving as many large LPs as there are distinct levels of ϵ_k^* before obtaining a full rank $\mathcal{H}_k^*$. However, in practice, we do not have to solve all of these LPs since the equality constraint $(\boldsymbol{x}-\boldsymbol{x}_k^*)(\mathcal{S})=0,\ \forall \mathcal{S}\in\mathcal{H}_{k-1}^*$ will make most of them redundant, i.e. any feasible solutions will result in the same optimal objective value due to the cost vector lies in the span of $\mathcal{H}_{k-1}^*$. In fact, we can use the same approach as in [31, 9] to prove the following results:

Theorem 2. *The total number of large LPs that we need to solve is at most $n-1$.*

Proof. We notice that once the equality constraints in $\mathbf{LP_k}$ are enforced, we have $(\boldsymbol{x}-\boldsymbol{x}_{k-1}^*)(\mathcal{S})=0, \forall \mathcal{S}\in span(\mathcal{H}_{k-1}^*)$ and hence $v(\mathcal{S})-\boldsymbol{x}(\mathcal{S})=v(\mathcal{S})-\boldsymbol{x}_{k-1}^*(\mathcal{S}), \forall \mathcal{S}\in span(\mathcal{H}_{k-1}^*)$. This means these excess values are constant and there is no reason to minimize them. Therefore, $\mathbf{LP_k}$ should only try to minimize the largest among non-constant excess values. In other words, we want to solve:

$$\begin{aligned}
\widehat{\mathbf{LP_k}} := \min_{\boldsymbol{x}\in\mathcal{I},\epsilon_k} \quad & \epsilon_k, \\
s.t. \quad & (\boldsymbol{x}-\boldsymbol{x}_{k-1}^*)(\mathcal{S})=0, \qquad \forall \mathcal{S}\in\mathcal{H}_{k-1}^*, && \text{(4a)}\\
& \epsilon_k+\boldsymbol{x}(\mathcal{S})\geq v(\mathcal{S}), \qquad \forall \mathcal{S}\notin span(\mathcal{H}_{k-1}^*). && \text{(4b)}
\end{aligned}$$

Notice that we have changed $\mathcal{S}\in\mathcal{C}\backslash\mathcal{H}_k^*$ to $\mathcal{S}\notin span(\mathcal{H}_k^*)$ as has been explained above. After solving $\widehat{\mathbf{LP_k}}$, we solve the small LPs in Theorem 1 to obtain tight coalitions $\mathcal{T}_k^*$. Notice that $\mathcal{T}_k^*$ are not in the span of $\mathcal{H}_{k-1}^*$ due to constraint (4b). Therefore, the rank of $\mathcal{H}_k^*$ is always greater than that of $\mathcal{H}_{k-1}^*$ (unless $\mathcal{H}_{k-1}^*$ has already had full rank). This means $rank(\mathcal{H}_k^*)\geq rank(\mathcal{H}_{k-1}^*)+1\geq\ldots\geq rank(\mathcal{H}_0^*)+k$. Notice also that the constraint $\boldsymbol{x}(\mathcal{N})=v(\mathcal{N})$ is also considered as a part of $\mathcal{H}_0$. This results in $rank(\mathcal{H}_k^*)\geq k+1$. Therefore, after solving at most $(n-1)$ LPs, we obtain $rank(\mathcal{H}_{n-1}^*)=n$, at which point the unique solution obtained is the nucleolus. □

The LP formulation $\widehat{\mathbf{LP_k}}$ now looks like that in Derks and Kuipers [9] but notice that the key difference is in our way of solving to obtain an optimal solution with the minimal tight set. To obtain an optimal solution, we use a constraint generation algorithm instead of solving the full LP directly. To improve the optimal solution found, our key contribution is Theorem 1 where we just need to solve small LPs to find another optimal solution with smaller tight set. It is not clear in Derks and Kuipers [9] how the minimal tight set is found. Our method also differs from Potters et al. [31] in the size of the LPs. In our formulation, $\widehat{\mathbf{LP_k}}$ has $(n+1)$ decision variables and (2^n+n-1) constraints (since there are (2^n-2) non-trivial coalitions, each of which corresponds to a constraint, and there are $n+1$ constraints for enforcing $\boldsymbol{x}\in\mathcal{I}$). This means our LPs have $(n+1)$ columns compared to the (2^n-1) columns in the LPs of Potters et al. [31]. The number of rows is the same at (2^n+n-1) for both our method and Potters et al. [31].

3.5 Dealing with issue 4: The tight sets are exponentially large

For each optimal solution $\boldsymbol{x}$ of $\mathbf{LP_k}$, the tight set $\mathcal{T}_k(\boldsymbol{x})$ might be exponentially large for many games. If this is the case, then the constraint set in $\mathbf{LP_{k+1}}$ will be very difficult to keep track of, and $\mathbf{LP_{k+1}}$ will be even harder to solve. However, it is interesting to observe that the equality constraints can be represented by m_k constraints where m_k is the rank of $\mathcal{H}_k^*$ ($m_k \leq n$). Thus, if we can replace $\mathcal{H}_k^*$ by its m_k representative vectors (coalitions), the equality constraints will be tractable. The issue is then how we can create this representative set. The simplest way is to keep introducing new tight coalitions until we can no longer improve the rank of $\mathcal{H}_k^*$. However, this method only works if the size of the tight set is not too large. If the tight set is large, we have no way to check whether the tight constraints found so far can form the representative set unless we have already had a full rank set (it could be the case that $\mathcal{H}_k^*$ does not have full rank). A better way is to introduce only tight coalitions that can increase the rank of $\mathcal{H}_k^*$ while undertaking the construction of the representative set. This guarantees that we increase the rank of $\mathcal{T}_r^*$ each time we generate a new constraint.

3.5.1 Producing a representative set

Suppose so far we have generated $\mathcal{T}_r = \{\boldsymbol{z}^1, \boldsymbol{z}^2, ..., \boldsymbol{z}^s\}$. In order to restrict $\boldsymbol{z}$ not to be in the linear combination of $\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$, we modify the constraint generation problem to:

$$\mathbf{REP} := \max_{\boldsymbol{z} \in \{0,1\}^n} \quad \{v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} \mid \boldsymbol{z} \notin span\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}\}.$$

Note that we have replaced the constraint $\boldsymbol{z} \notin \{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$ in the original **CG** problem described in Section 3.2 by $\boldsymbol{z} \notin span\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$. This is equivalent to the existence of a vector $\boldsymbol{u}$ with $\boldsymbol{u}^t\boldsymbol{z}^j = 0, \forall j = 1, \ldots, s$ and $\boldsymbol{z}^t\boldsymbol{u} < 0$ (similar to Farkas' Lemma). This is also equivalent to the following set of constraints:

$$\left\{\boldsymbol{z} = \boldsymbol{u}^1 + \boldsymbol{u}^2,\ \boldsymbol{u}^1 \in span\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\},\ \boldsymbol{u}^2 \in null\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\},\ \boldsymbol{u}^2 \neq 0\right\}.$$

By our construction, $\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$ are s independent vectors. Therefore $null\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$ is a subspace in $\mathbb{R}^{n-s}$. Let $\boldsymbol{A} = [\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s]$ and let $\boldsymbol{B} \in \mathbb{R}^{n \times (n-s)}$ be a set of $(n-s)$ vectors in $\mathbb{R}^n$ that spans $null\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$. Let $\boldsymbol{u}^1 = \boldsymbol{A}\boldsymbol{\omega}_a$ and $\boldsymbol{u}^2 = \boldsymbol{B}\boldsymbol{\omega}_b$ for some vectors $\boldsymbol{\omega}_a$ and $\boldsymbol{\omega}_b$. The condition $\boldsymbol{u}^2 \neq 0$ is equivalent to $\boldsymbol{\omega}_b \neq 0$. The condition $\boldsymbol{z} \notin span\{\boldsymbol{z}^1, \boldsymbol{z}^2, \ldots, \boldsymbol{z}^s\}$ is equivalent to: $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{\omega}_a + \boldsymbol{B}\boldsymbol{\omega}_b,\ \boldsymbol{\omega}_b \neq 0$. Let us define the domain

$$\mathcal{D} \quad = \quad \{(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \ :\ \boldsymbol{z} \in \{0,1\}^n,\ \boldsymbol{\omega}_a \in \mathbb{R}^s,\ \boldsymbol{\omega}_b \in \mathbb{R}^{n-s},\ \boldsymbol{z} = \boldsymbol{A}\boldsymbol{\omega}_a + \boldsymbol{B}\boldsymbol{\omega}_b,\ \boldsymbol{\omega}_b \neq 0,\ v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} = \epsilon_k^*\}.$$

Notice that $\boldsymbol{x}$, ϵ_k^*, $\boldsymbol{A}$ and $\boldsymbol{B}$ are known. The constraint generation problem is equivalent to finding a feasible solution $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \in \mathcal{D}$. In order to check whether $\mathcal{I}_k(\boldsymbol{x})$ contains a unique point $\boldsymbol{x}$, we can find a feasible point in $\mathcal{D}$ by optimizing any arbitrary objective function over that domain. However, it is not straightforward to model the inequality constraint $\boldsymbol{\omega}_b \neq 0$. Instead, we need to optimize a set of objective functions over a domain $\mathcal{D}_r$ defined as follows:

$$\mathcal{D}_r \quad = \quad \{(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \ :\ \boldsymbol{z} \in \{0,1\}^n,\ \boldsymbol{\omega}_a \in \mathbb{R}^s,\ \boldsymbol{\omega}_b \in \mathbb{R}^{n-s},\ \boldsymbol{z} = \boldsymbol{A}\boldsymbol{\omega}_a + \boldsymbol{B}\boldsymbol{\omega}_b,\ v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} = \epsilon_k^*\},$$

where the inequality constraint is removed. We can find a feasible point in $\mathcal{D}$ or conclude it empty by using the following Proposition:

Proposition 1. *Let $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n-s}$ be any set of $(n-s)$ linearly independent cost vectors. If solving the problems $\min_{(\boldsymbol{z},\boldsymbol{\omega}_a,\boldsymbol{\omega}_b)\in\mathcal{D}_r} \boldsymbol{c}_j^t\boldsymbol{\omega}_b$ and $\min_{(\boldsymbol{z},\boldsymbol{\omega}_a,\boldsymbol{\omega}_b)\in\mathcal{D}_r} -\boldsymbol{c}_j^t\boldsymbol{\omega}_b$ for each $j = 1, \ldots, n$ all have optimal values that are equal to zero, then $\mathcal{D}$ is an empty set (which means* **REP** *is either infeasible or the optimal value is not equal to ϵ_k^*). Otherwise, any feasible solution $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b)$ of any of these $2(n-s)$ problems with a non-zero objective value will be a feasible point in $\mathcal{D}$. In that case, $\boldsymbol{z}$ will be a solution of* **REP**.

Proof. We notice that $\mathcal{D}_r$ is a non-empty domain since $(\boldsymbol{z} = \boldsymbol{z}^1,\ \boldsymbol{\omega}_a = (1, 0, \ldots, 0)^t,\ \boldsymbol{\omega}_b = 0)$ is a feasible solution with an objective value equal to zero. Therefore, solving all these $2(n-s)$ optimization problems will return either (a) at least one feasible solution with non-zero objective value or (b) all of them have the same objective values that are equal to zero. In the first case, suppose there exists a feasible solution $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \in \mathcal{D}_r$ such that $\boldsymbol{c}_j^t\boldsymbol{\omega}_b \neq 0$. Then $\boldsymbol{\omega}_b \neq 0$ and hence $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \in \mathcal{D}$. In the second case, suppose all feasible solutions of the $2(n-s)$ share the same objective value of zero. We need to prove that the set $\mathcal{D}$ is empty to complete the theorem. Suppose for the purposes of contradiction the set $\mathcal{D}$ is non-empty, i.e. there exists $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \in \mathcal{D}$. This means $(\boldsymbol{z}, \boldsymbol{\omega}_a, \boldsymbol{\omega}_b) \in \mathcal{D}_r$. We also have $\boldsymbol{c}_j^t\boldsymbol{\omega}_b = 0,\ \forall j \in \{1, \ldots, n-s\}$. This is impossible because $\boldsymbol{\omega}_b \neq 0$ while $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n-s}$ are linearly independent. □

Proposition 1 provides a way to find an optimal solution of **REP** (or to conclude that it is infeasible). Although the theorem appears to involve solving $2(n-s)$ LPs, an optimal solution of **REP** is found whenever an LP among them provides us with a feasible solution with non-zero objective value. This means if we are lucky to choose the right set of cost vectors, we do not have to solve all of these LPs. However, this approach still requires solving all the $2(n-s)$ LPs if **REP** is indeed infeasible. We can speed up the process by using a randomization technique. For a random cost vector $\boldsymbol{c}$, the condition $\boldsymbol{\omega}_b \neq 0$ is equivalent to either the optimal value of $\min\ \boldsymbol{c}^t\boldsymbol{\omega}_b$ or that of $\min\ -\boldsymbol{c}^t\boldsymbol{\omega}_b$ being different from zero with probability one (w.p.1).

3.5.2 Using the representative set

Let $\mathcal{R}_k^*$ be the representative set of $\mathcal{H}_k^*$. We prove that we still can solve $\mathbf{LP_{k+1}}$ even without the full knowledge of $\mathcal{H}_k^*$. Once we have obtained the representative set, we reformulate $\mathbf{LP_{k+1}}$ as follows:

$$\mathbf{LP_{k+1}} := \min_{\boldsymbol{x}\in\mathcal{I},\epsilon_{k+1}} \quad \{\epsilon_{k+1} \mid (\boldsymbol{x} - \boldsymbol{x}_k^*)(\mathcal{S}) = 0,\ \forall \mathcal{S} \in \mathcal{R}_k^*,\ \epsilon_{k+1} + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}), \qquad \forall \mathcal{S} \in \mathcal{C}\backslash\mathcal{H}_k^*\}.$$

In this case, the number of equality constraints has been reduced from $|\mathcal{H}_k^*|$ to $|\mathcal{R}_k^*|$ without changing $\mathbf{LP_{k+1}}$ since the remaining equalities are redundant. However, the inequalities seem to still involve $\mathcal{H}_k^*$ while we are trying to avoid it by using its representative $\mathcal{R}_k^*$. Fortunately, this is not an issue in our constraint generation method because what we need is to be able to generate violating constraints by solving the **CG** problem, $\max_{\mathcal{S}\in\mathcal{C}\backslash\mathcal{H}_k^*} \{v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})\}$, which is equivalent to

$$\max_{\mathcal{S}} \quad \{v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S}) \mid v(\mathcal{S}) - \boldsymbol{x}_k(\mathcal{S}) < \epsilon_k^*\},$$

since $\mathcal{H}_k^* = \left\{\mathcal{S} \mid v(\mathcal{S}) - \boldsymbol{x}_k(\mathcal{S}) \geq \epsilon_k^*\right\}$. Thus, our constraint generation algorithm still does not change when we replace $\mathcal{H}_k^*$ with its representative set. However, the **FBOS** problem for finding an improved imputation needs to be changed to:

$$\begin{aligned}
\textbf{FBOS2} = \min_{\boldsymbol{y} \in \mathcal{I}} \quad & \sum_{\mathcal{S} \in \mathcal{R}_k^*(\boldsymbol{x})} [v(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})], \\
s.t. \quad & |y_i - x_i| \leq \delta, \ \forall\ i = 1, .., n, \\
& (\boldsymbol{y} - \boldsymbol{x}_{k-1}^*)(\mathcal{S}) = 0, \qquad \forall \mathcal{S} \in \mathcal{R}_{k-1}^*, \\
& (\boldsymbol{y} - \boldsymbol{x})(\mathcal{S}) \geq 0, \qquad \forall \mathcal{S} \in \mathcal{T}_k(\boldsymbol{x}).
\end{aligned}$$

Notice that **FBOS2** is different from **FBOS** in the objective function and in the equality constraints. Here, we reduce the number of equality constraints from $|\mathcal{H}_k^*|$ to $|\mathcal{R}_k^*|$ without changing the problem because the remaining equality constraints are redundant. The constraints $(\boldsymbol{y} - \boldsymbol{x})(\mathcal{S}) \geq 0,\ \forall \mathcal{S} \in \mathcal{T}_k(\boldsymbol{x})$ can be dealt with using the same constraint generation procedure used to solve $\mathbf{LP_k}$. However, we need the following theorem to extend the results from Theorem 1:

Theorem 3. *Let $(\boldsymbol{x}, \epsilon_k^*)$ be any optimal solution of $\mathbf{LP_k}$ with the tight set $\mathcal{T}(\boldsymbol{x})$. Then solving* **FBOS2** *will lead to two cases:*

(a) If $\boldsymbol{x}$ is not an optimal solution of **FBOS2***, then any optimal solution $\boldsymbol{y}^*$ of* **FBOS2** *would result in a smaller tight set, i.e. $|\mathcal{T}(\boldsymbol{y}^*)| < |\mathcal{T}(\boldsymbol{x})|$,*

(b) If $\boldsymbol{x}$ is an optimal solution of **FBOS2***, then $\boldsymbol{x}$ is also an optimal solution of $\mathbf{LP_k}$ with the minimal tight set, i.e. $\mathcal{T}(\boldsymbol{x}) \equiv \mathcal{T}_k^*$.*

Proof. Let $\mathcal{R}_k^*(\boldsymbol{x}) = \{\boldsymbol{z}^1, \ldots, \boldsymbol{z}^r\}$.

(a) Notice that $\boldsymbol{x}, \boldsymbol{x}_k^*$ are feasible solutions of **FBOS2**. In addition, **FBOS2** is bounded and hence it has an optimal solution. Suppose $\boldsymbol{x}$ is not an optimal solution of **FBOS2**. Then any optimal solution $\boldsymbol{y}^*$ of **FBOS2** would have a smaller objective value. In particular, there exists a coalition $\mathcal{S}$ in $\mathcal{R}_k^*(\boldsymbol{x})$ such that $v(\mathcal{S}) - \boldsymbol{y}^*(\mathcal{S}) < v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})$. Since all other excess values of $\boldsymbol{y}^*$ are less than or equal to those of $\boldsymbol{x}$ by the construction of **FBOS2**, $\boldsymbol{y}^*$ must have a smaller tight set compared to $\boldsymbol{x}$ in $\mathbf{LP_k}$.

(b) Suppose $\boldsymbol{x}$ is an optimal solution of **FBOS2**. We will prove $\boldsymbol{x}$ is also an optimal solution of $\mathbf{LP_k}$ with the minimal tight set. Suppose, as a contradiction, that there exists another optimal solution $\boldsymbol{y}^*$ of $\mathbf{LP_k}$ with $|\mathcal{T}_k(\boldsymbol{y}^*)| < |\mathcal{T}_k(\boldsymbol{x})|$. This means there exists at least one coalition $\boldsymbol{z}^0$ such that $v(\boldsymbol{z}^0) - \boldsymbol{y}^*(\boldsymbol{z}^0) < v(\boldsymbol{z}^0) - \boldsymbol{x}(\boldsymbol{z}^0)$. Since any convex combination of $\boldsymbol{x}$ and $\boldsymbol{y}^*$ is an optimal solution of $\mathbf{LP_k}$ whose tight set is smaller than $\mathcal{T}_k(\boldsymbol{x})$, we can assume $\boldsymbol{y}^*$ is sufficiently close but not equal to $\boldsymbol{x}$. This means $\boldsymbol{y}^*$ is also a solution of **FBOS2**. Notice that we can rewrite the objective function as follows:

$$\sum_{S \in \mathcal{R}_k^*(\boldsymbol{x})} [v(\mathcal{S}) - \boldsymbol{y}(\mathcal{S})] \ = \ \sum_{S \in \mathcal{R}_k^*(\boldsymbol{x})} v(\mathcal{S}) - \sum_{\mathcal{S} \in \mathcal{R}_k^*(\boldsymbol{x})} \boldsymbol{y}(\mathcal{S}) \ = \ \sum_{S \in \mathcal{R}_k^*(\boldsymbol{x})} v(\mathcal{S}) - r\boldsymbol{c}^t\boldsymbol{y},$$

where r is the size of $\mathcal{R}_k^*(\boldsymbol{x})$ and $\boldsymbol{c} = \frac{\boldsymbol{z}^1+\ldots+\boldsymbol{z}^r}{r}$ is the average of all coalitions $\boldsymbol{z}$ in $\mathcal{R}_k^*(\boldsymbol{x})$. Since $\sum_{S\in\mathcal{R}_k^*(\boldsymbol{x})} v(\mathcal{S})$ and r are constants in **FBOS2**, the problem is equivalent to minimizing $-\boldsymbol{c}^t\boldsymbol{y}$. Since $\mathcal{R}_k^*(\boldsymbol{x})$ is the representative set, $\boldsymbol{z}^0$ can be expressed as $\boldsymbol{z}^0 = \sum_{i=1}^r \beta_i \boldsymbol{z}^i$. For any α_0, let $\alpha_i = 1/r - \alpha_0\beta_i$, we have: $\boldsymbol{c} = \frac{1}{r}(\boldsymbol{z}^1 + \ldots + \boldsymbol{z}^r) = \sum_{i=1}^{r}(\frac{1}{r} - \alpha_0\beta_i)\boldsymbol{z}^i + \alpha_0 \sum_{i=1}^{r} \beta_i \boldsymbol{z}^i = \sum_{i=1}^{r} \alpha_i \boldsymbol{z}^i + \alpha_0 \boldsymbol{z}_0$. We can choose $\alpha_0 > 0$ and small enough such that $\alpha_i \geq 0,\ \forall i$. Thus,

$$\boldsymbol{c}^t(\boldsymbol{x} - \boldsymbol{y}^*) \quad = \quad \underbrace{\sum_{i=1}^{r} \alpha_i (\boldsymbol{z}^i)^t (\boldsymbol{x} - \boldsymbol{y}^*)}_{\leq 0} + \underbrace{\alpha_0 (\boldsymbol{z}^0)^t (\boldsymbol{x} - \boldsymbol{y}^*)}_{<0} < 0.$$

This means $\boldsymbol{x}$ is not an optimal solution of **FBOS2** and so there is a contradiction. The proof of the theorem is complete.

□

4 Applicability to Combinatorial Games

We demonstrate the applicability of our algorithm to three classes of games. In these games, the entire characteristic functions are costly to computed for instances with more than 25 players, e.g. it might involves solving up to 2^n optimisation problems just for getting the input of the general flow games Kalai and Zemel [19]. We show that by using the constraint generation framework, we can eliminate this stage and only compute some of the characteristic values by searching for the most violating constraints.

4.1 General Flow Games

The flow games were proposed by Kalai and Zemel [19]. Consider a network $G(\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{c}, s, t)$ with vertices $\boldsymbol{V}$, edges $\boldsymbol{E}$, edge capacity $\boldsymbol{c}$, source s and sink t. Each player owns an edge among $n = |\boldsymbol{E}|$ number of edges. In the case all the players cooperate, the total payoff (reward) received by all the players will be the maximum flow that can be sent from source s to sink t: $v(\mathcal{N}) = \text{max-flow}(G)$. The max-flow problem can be formulated as an LP as follows:

$$\begin{aligned}
\max_{\boldsymbol{f}} \quad & \sum_{j\ :\ (s,j)\in\boldsymbol{E}} f_{sj}, \\
s.t. \quad & \sum_{j\ :\ (j,i)\in\boldsymbol{E}} f_{ij} - \sum_{k\ :\ (i,k)\in\boldsymbol{E}} f_{ik} = 0,\ \forall\ i \in \boldsymbol{V} \setminus \{s,t\}, \\
& 0 \leq f_{ij} \leq c_{ij},\ \forall\ (i,j) \in \boldsymbol{E}.
\end{aligned}$$

Suppose only a subset $\mathcal{S} \subset \mathcal{N}$ of players cooperate among themselves. The total payoff received by players in the set $\mathcal{S}$ will become:

$$v(\mathcal{S}) = \text{max-flow}(\boldsymbol{V}, \boldsymbol{E}(\boldsymbol{z}), \boldsymbol{c}(\boldsymbol{z}), s, t),$$

which is the maximum flow that can be sent from sources s to sink t by using only the edges in subset $\mathcal{S}$. Deng et al. [8] show that finding the nucleolus is NP-hard for general flow games. The nucleolus can only be found in polynomial time if all the edge has unit capacities. Even in that case, the algorithm relies on the ellipsoid method which performs poorly in practice.

In order to apply the constraint generation framework, we need to find the worst coalition for a current proposal $\boldsymbol{x}$, i.e. to solve

$$\begin{aligned} \max_{\boldsymbol{z}\in\{0,1\}^m} \quad & v(\boldsymbol{z}) - \boldsymbol{x}^t\boldsymbol{z}, \\ s.t. \quad & \boldsymbol{z} \notin span(\mathcal{H}^*_{k-1}), \end{aligned}$$

where $\boldsymbol{z}$ is the binary vector that indicates whether an edge is in the subset $\mathcal{S}$, and $v(\boldsymbol{z})$ is the payoff of coalition $\mathcal{S}$ which can be formulated as follows:

$$\begin{aligned} v(\mathcal{S}) \equiv v(\boldsymbol{z}) = \max_{\boldsymbol{f}} \quad & \sum_{j\,:\,(s,j)\in\boldsymbol{E}} f_{sj}, \\ s.t. \quad & \sum_{j\,:\,(j,i)\in\boldsymbol{E}} f_{ji} - \sum_{k\,:\,(i,k)\in\boldsymbol{E}} f_{ik} = 0,\ \forall\ i \in \boldsymbol{V}\setminus\{s,t\}, && \text{(5a)} \\ & 0 \le f_{ij} \le z_{ij}c_{ij},\ \forall\ (i,j)\in\boldsymbol{E}. && \text{(5b)} \end{aligned}$$

The set of constraints (5b) enforces the flow f_{ij} to be zero if an edge (i,j) is not in the coalition. With the characteristic function $v(\mathcal{S})$ available for each coalition $\mathcal{S}$, a stable reward shared among the players could be modeled as the nucleolus value of the cooperative game.

By replacing the formulation for $v(\boldsymbol{z})$ and combine the two max operators together, the CG problem can be reformulated as:

$$\begin{aligned} \max_{\boldsymbol{z},\boldsymbol{f}} \quad & -\boldsymbol{x}^t\boldsymbol{z} + \sum_{j\,:\,(s,j)\in\boldsymbol{E}} f_{sj}, \\ s.t. \quad & \sum_{j\,:\,(j,i)\in\boldsymbol{E}} f_{sj} - \sum_{k\,:\,(i,k)\in\boldsymbol{E}} f_{ik} = 0,\ \forall\ i \in \boldsymbol{V}\setminus\{s,t\}, \\ & 0 \le f_{ij} \le z_{ij}c_{ij},\ \forall\ (i,j)\in\boldsymbol{E}, \\ & \boldsymbol{z} \in \{0,1\}^n. \end{aligned}$$

This is an MILP and can be handled by CPLEX for games of reasonable size.

4.2 Weighted Voting Games

In the weighted voting games (WVG), each player has a voting weight and a coalition receives a payoff of one if the total voting weight of the coalition's members exceeds some threshold and a payoff of zero otherwise (see Leech [24], Elkind et al. [11], Elkind and Pasechnik [10] for details). WVGs have many applications in political science, reliability theory and computer science (see Aziz et al. [1] and the references therein). Let w_i be the number of votes that player i has. A coalition $\mathcal{S}$ has a total votes of $\boldsymbol{\omega}(\mathcal{S}) = \sum_{i\in\mathcal{S}} w_i$. The coalition will win the game if its total votes exceeds some threshold κ.

The characteristic function of the game is defined as: $v(\mathcal{S}) = 1$ if $\boldsymbol{\omega}(\mathcal{S}) \geq \kappa$ and $v(\mathcal{S}) = 0$ otherwise. Elkind et al. [11] and [10] show that computing the least core of the WVG is NP-hard. They also show that the nucleolus can be computed in polynomial time under the assumption that the size of the tight set $\mathcal{T}_k(\boldsymbol{x})$ is known. However, finding the size $|\mathcal{T}_k(\boldsymbol{x})|$ is not straightforward, especially for WVG with exponentially large tight sets. In addition, the polynomial running time property that the authors derived is based on the ellipsoid method which does not perform well in practice. Despite these NP-hardness results, we will show that, by using our method, the WVG can be solved efficiently for instances involving sizes up to 100 players.

Since the main routine in computing the nucleolus is to find the coalition with the largest deficit $v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})$ from a given imputation $\boldsymbol{x}$, we focus our discussion on this problem. The constraint generation problem arisen in solving $\mathbf{LP_k}$ is: $\max_{\boldsymbol{z}\in\{0,1\}^n} \{v(\boldsymbol{z}) - \boldsymbol{z}^t\boldsymbol{x} \mid v(\boldsymbol{z}) - \boldsymbol{x}_{k-1}(\boldsymbol{z}) < \epsilon^*_{k-1}\}$. We introduce a binary variable $z_0 = v(\boldsymbol{z})$ and reformulate the problem as: $\max_{(z_0,\boldsymbol{z})\in\{0,1\}^{n+1}} \{z_0 - \boldsymbol{z}^t\boldsymbol{x} \mid \boldsymbol{\omega}^t\boldsymbol{z} \geq z_0\kappa,\ z_0 - \boldsymbol{x}^t_{k-1}\boldsymbol{z} < \epsilon^*_{k-1}\}$. Here, the constraint $\boldsymbol{\omega}^t\boldsymbol{z} \geq z_0\kappa$ ensures that $z_0 = 0$ when $\boldsymbol{\omega}^t\boldsymbol{z} < \kappa$ (the strict inequality can be turned into a normal inequality like $\boldsymbol{\omega}^t\boldsymbol{z} <= \kappa - \min_i w_i$ by using the fact that $\boldsymbol{z}$ is binary and κ is sufficiently large compared to $\min_i w_i$). However, z_0 should be equal to 1 to drive the objective function to optimality if the condition $\boldsymbol{\omega}^t\boldsymbol{z} \geq \kappa$ holds. This is an MILP with $(n+1)$ binary variables and with two constraints and can be solved very efficiently.

4.3 Weighted Coalitional Skill Games

Weighted coalitional skill games (WCSG) were proposed by Bachrach and Rosenschein [3], Bachrach et al. [2]. In this game, there are n agents, T tasks and K skills. Each agent has a subset of skills and each task requires a subset of skills. Let Ψ be the agent-skill matrix with binary indicator ψ_{ik} denoting whether agent i has skill k. Let Φ be the task-skill matrix with binary indicator ϕ_{tk} denoting whether task t requires skill k. For each task t, and the skill vector Φ_t that it requires, the coalition $\boldsymbol{z}$ will be able to perform the task if there exists at least one agent in the coalition that has skill ϕ_{tk}. We consider a weighted average utility function defined as follows. Let $\Delta(\boldsymbol{z}, t)$ be the binary indicator on whether coalition $\boldsymbol{z}$ can perform task t. Then the coalition value is defined as $v(\mathcal{S}) = \sum_{t\in 1..T} \omega_t \Delta(\boldsymbol{z}, t)$ where

$$\Delta(\boldsymbol{z}, t) = \begin{cases} 1 & \text{if } \Psi^T\boldsymbol{z} \geq \Phi_t, \\ 0 & \text{otherwise.} \end{cases}$$

We will show that the constraint generation problem can be solved efficiently in the WCSG games with reasonable size (i.e. $n \leq 500$). Let us define variable $\delta_t = \Delta(\boldsymbol{z}, t)$. In this case, the constraint generation problem $min_{\boldsymbol{z}\in\{0,1\}^n} \left[\boldsymbol{z}^t\boldsymbol{x} - v(\boldsymbol{z})\right]$ can be reformulated as:

$$\min_{\boldsymbol{z}\in\{0,1\}^n} \boldsymbol{z}^t\boldsymbol{x} - \sum_{t=1}^{m} \omega_t \Delta(\boldsymbol{z}, t),$$

which is equivalent to:

$$\begin{aligned} \min_{\boldsymbol{z},\delta} \quad & \boldsymbol{z}^t\boldsymbol{x} - \sum_{t\in 1..m} \omega_t\delta_t, \\ s.t. \quad & \Psi^T\boldsymbol{z} \geq \delta_t\Phi_t, \ \forall t \in 1,..,m, \qquad (6a) \\ & \boldsymbol{z} \in \{0,1\}^n, \delta_t \in \{0,1\}^m. \qquad (6b) \end{aligned}$$

The set of constraints (6a) forces δ_t to be equal to zero if coalition $\boldsymbol{z}$ does not have all the skill required in Φ_t. Otherwise, δ_t should be equal to one to drive the objective function to the optimum. The constraint generation problem is a mixed integer programming problem with $(n+m)$ binary variables and with $(m+t)$ constrains. Although the problem is NP-hard, we will shown numerically that CPLEX can handle this class of games for instances with up to 75 players (under various choices of the skills and tasks).

5 Numerical Experiments

In our numerical experiments, we start with a small flow game to demonstrate the steps involved in our algorithm. Simulated large weighted voting games and coalitional skill games with 25 to 100 players are presented to demonstrate the performance of the algorithm in detail.

5.1 Algorithm Demonstration via a small flow game

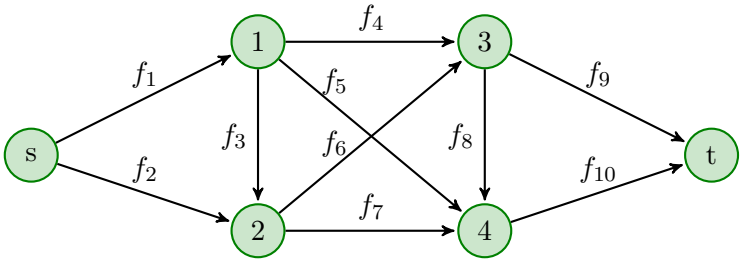

Figure 4: Example with 10 players (10 edges, 6 nodes).

Consider a flow game shown in Figure 4. There are $n = 10$ players (edges) that are numbered according to the figure. The capacities of the edges are $c_1 = c_9 = 3$, $c_i = 1, \forall\ i \in \{3,\ldots,8\}$ and $c_2 = c_{10} = 2$. Table 5.1 shows the step involved in computing the nucleolus of this game as described in Algorithm 1. The following three key tasks are undertaken iteratively:

(1) Solve $\mathbf{LP_k}$ by using the constraint generation algorithm to produce an optimal solution $\boldsymbol{x}_k$,
(2) Solve the **REP** problem to find the representative set of all tight constraints in $\mathcal{T}(\boldsymbol{x}_k)$,
(3) Solve the **FBOS** problem iteratively to find an optimal solution with the minimal tight set. If the stopping condition doesn't hold, formulate the subsequent $\mathbf{LP_k}$ and go back to step (1).

The first two columns in Table 5.1 show the steps and the tasks undertaken. The third column shows the input required. The fourth column shows the output and some remark about it. The rows shows the information for each step taken by the algorithm. For example, the first row shows step 1 of solving $\mathbf{LP_1}$. That step produces $\epsilon_1^* = 0$ and $\boldsymbol{x}_1 = [1, 1, 0, 0, 1, 0, 0, 0, 1, 0]$. Using the CG algorithm, we obtain an optimal solution after only 16 iterations. The tight set produced by the last relaxed LP outputs only four tight constraints among the total of 135 tight constraints. Instead of iteratively finding all the remaining tight constraints, our method requires finding only 6 other constraints to form the representative tight set by solving problem **REP** (the third row in Table 5.1). With this representative set, we solve **FBOS** to obtain an improved optimal solution. First, we obtain $\boldsymbol{x} = (1, 0.5, 0, 0, 1, 0, 0.5, 0, 1)$ with the tight set reduced to $|\mathcal{T}(\boldsymbol{x})| = 75$ and $rank(\mathcal{T}(\boldsymbol{x})) = 9$. Solving the **FBOS** three more times provides us with three new optimal solutions with the tight set sizes being 29, 17 and 11 respectively. The ranks of the tight sets are also reduced from 9 to 6, 5 and 4 respectively (shown in rows 4-7). The optimal solution found is $\boldsymbol{x} = (1, 0.4375, 0, 0.125, 0.75, 0.25, 0.5, 0, 0.875, 0.0625)$. At this point, the tight set has the smallest size and we can proceed to the subsequent LP. Solving $\mathbf{LP_2}$, again using the CG algorithm, provides us with the optimal solution $\epsilon_1^* = -0.2$ and $\boldsymbol{x} = (0.8, 0.2, 0, 0.2, 0.6, 0.6, 0.6, 0, 0.8, 0.2)$. We repeat the process of solving the **REP** and **FBOS** and formulate $\mathbf{LP_3}$. After finding the representative set and solving **FBOS**, we arrive at an optimal solution $\boldsymbol{x} = (1, 0.2, 0, 0.2, 0.4, 0.4, 0.6, 0, 1, 0.2)$ with the minimal tight set. Since $\mathcal{H}_3^*$ has full rank. We stop the algorithm and conclude that $\boldsymbol{x}$ is the nucleolus.

5.2 Large Weighted Voting Games

We generate WVGs with different size and parameters as follows. For each $n = \{25, 50, 75, 100\}$, we generate random weight vectors using the χ^2 distribution with different degrees of freedom ρ, i.e. $\rho = \{1, 5, n\}$. The winning fraction f is set to either $f = 0.5$ or $f = 0.75$, i.e. either $\kappa = \lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega} \rceil$ or $\kappa = \lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega} \rceil$. For each combination of (n, ρ, f), we generate $K = 10$ simulated games. Table 3 shows the computational results for these games in details. The first column in Table 3 shows the number of the players, which ranges from 25 to 100. The second and third columns show the distribution function of the weight vector and he winning fraction correspondingly. Columns 4-9 show the computation time (in seconds) required to run the entire algorithm or to complete different subtasks. Column 4 shows the total computational time from the beginning until the nucleolus is found. The total computation time is broken down into three parts; a) to solve all the $\mathbf{LP_k}$, b) to find the representative set, and c) to find an improved imputation. The computational times for these tasks are shown in column five, six and seven correspondingly. The computation time for solving the $\mathbf{LP_k}$ is further broken down to the time to solve the relaxed LPs and to solve the constraint generation sub-problems. These are shown in columns 8 and 9. Column 10 shows the number of $\mathbf{LP_k}$ required and column 11 shows number of iterations required for the constraint generation algorithm to solve $\mathbf{LP_k}$. Each row shows the average of these computation statistics over K games for each combination of (n, ρ, f). For example, the second row shows all the statistics for games with 25 players; the weight vector is from χ_1^2 and the winning fraction is 50% while the last row is for games with 100 players; the weight vector is from χ_{100}^2 and the winning fraction is 75%.

Steps	Tasks	Input	Output (update) & notes
1	Solve $\mathbf{LP_1}$	$\mathcal{H}_0^* = \boldsymbol{e}$	$\epsilon_1^* = 0$, $\boldsymbol{x}_1 = [1, 1, 0, 0, 1, 0, 0, 0, 1, 0]$. CG terminates in 16 iterations
2	Find REP	$\boldsymbol{x}_1$	Although $\mathcal{T}_k(\boldsymbol{x}_1)$ has 135 tight constraints that span $\mathbb{R}^{10}$, it is represented by $\mathcal{R}_1(\boldsymbol{x}_1)$ with 10 constraints.
3.1	Solve FBOS	$\boldsymbol{x}_1, \mathcal{R}_1(\boldsymbol{x}_1)$	Update $\boldsymbol{x}_1 = [1, 0.5, 0, 0, 1, 0, 0.5, 0, 1]$. $\mathcal{T}_1(\boldsymbol{x}_1)$ has a smaller size with 75 tight constraints. It is also represented by a smaller REP $\mathcal{R}_1(\boldsymbol{x}_1) \in \mathbb{R}^9$.
3.2	Solve FBOS	$\boldsymbol{x}_1, \mathcal{R}_1(\boldsymbol{x}_1)$	Update $\boldsymbol{x}_1 = [1, 0.5, 0, 0, 0.75, 0.25, 0.5, 0, 1]$. $\mathcal{T}_1(\boldsymbol{x}_1)$ has a smaller size with 29 tight constraints. It is also represented by a smaller REP $\mathcal{R}_1(\boldsymbol{x}_1) \in \mathbb{R}^6$.
3.2	Solve FBOS	$\boldsymbol{x}_1, \mathcal{R}_1(\boldsymbol{x}_1)$	Update $\boldsymbol{x}_1 = [1, 0.5, 0, 0.125, 0.75, 0.25, 0.5, 0, 0.875, 0]$. $\mathcal{T}_1(\boldsymbol{x}_1)$ has a smaller size with 17 tight constraints. It is also represented by a smaller REP $\mathcal{R}_1(\boldsymbol{x}_1) \in \mathbb{R}^5$.
3.2	Solve FBOS	$\boldsymbol{x}_1, \mathcal{R}_1(\boldsymbol{x}_1)$	Update $\boldsymbol{x}_1 = [1, 0.4375, 0, 0.125, 0.75, 0.25, 0.5, 0, 0.875, 0.0625]$. $\mathcal{T}_1(\boldsymbol{x}_1)$ has a smaller size with 11 tight constraints. It is also represented by a smaller REP $\mathcal{R}_1(\boldsymbol{x}_1) \in \mathbb{R}^4$.
3.3	Solve FBOS	$\boldsymbol{x}_1, \mathcal{R}_1(\boldsymbol{x}_1)$	Set $\boldsymbol{x}_1^* = \boldsymbol{x}_1$, $\mathcal{R}_1^* = \mathcal{R}_1(\boldsymbol{x}_1)$. No further improvement
End of first loop. Found an optimal solution of the first LP with minimal tight set. Check stopping condition. $\mathcal{H}_1^*$ does not have full rank. Continue!			
1-3	Solve $\mathbf{LP_2}$	$\mathcal{H}_1^*, \boldsymbol{x}_1^*, \epsilon_1^*$	$\epsilon_2^* = -0.2$, $\boldsymbol{x}_2 = [0.8, 0.2, 0, 0.2, 0.6, 0.6, 0.6, 0, 0.8, 0.2]$. CG terminates in 16 iterations
	Find REP	$\boldsymbol{x}_2$	Although $\mathcal{T}_2(\boldsymbol{x}_2)$ has 28 tight constraints that span $\mathbb{R}^{10}$, it is represented by $\mathcal{R}_2(\boldsymbol{x}_2)$ with 10 constraints.
3.1	Solve FBOS	$\boldsymbol{x}_2, \mathcal{R}_2(\boldsymbol{x}_2)$	Update $\boldsymbol{x}_2 = [0.875, 0.2, 0, 0.2, 0.525, 0.6, 0.6, 0, 0.8, 0.2]$. $\mathcal{T}_2(\boldsymbol{x}_2)$ has a smaller size with 26 tight constraints. It is also represented by a smaller REP $\mathcal{R}_2(\boldsymbol{x}_2) \in \mathbb{R}^9$.
3.2	Solve FBOS	$\boldsymbol{x}_2, \mathcal{R}_2(\boldsymbol{x}_2)$	Update $\boldsymbol{x}_2 = [0.875, 0.2, 0, 0.2, 0.525, 0.45, 0.6, 0, 0.95, 0.2]$. $\mathcal{T}_2(\boldsymbol{x}_2)$ has a smaller size with 24 tight constraints. It is also represented by a smaller REP $\mathcal{R}_2(\boldsymbol{x}_2) \in \mathbb{R}^6$.
3.3	Solve FBOS	$\boldsymbol{x}_2, \mathcal{R}_2(\boldsymbol{x}_2)$	Set $\boldsymbol{x}_2^* = \boldsymbol{x}_2$, $\mathcal{R}_2^* = \mathcal{R}_2(\boldsymbol{x}_2)$. No further improvement
End of second loop. Found an optimal solution of the second LP with minimal tight set. Check stopping condition. $\mathcal{H}_2^*$ does not have full rank. Continue!			
1-3	Solve $\mathbf{LP_3}$	$\mathcal{H}_2^*, \boldsymbol{x}_1^*, \epsilon_2^*$	$\epsilon_3^* = -0.4$, $\boldsymbol{x}_3 = [1, 0.2, 0, 0.2, 0.4, 0.4, 0.6, 0, 1, 0.2]$. CG terminates in 4 iterations
	Find REP	$\boldsymbol{x}_3$	Although $\mathcal{T}_3(\boldsymbol{x}_3)$ has 48 tight constraints that span $\mathbb{R}^{10}$, it is represented by $\mathcal{R}_3(\boldsymbol{x}_3)$ with 10 constraints.
3.1	Solve FBOS	$\boldsymbol{x}_3, \mathcal{R}_3(\boldsymbol{x}_3)$	Set $\boldsymbol{x}_3^* = \boldsymbol{x}_3$, $\mathcal{R}_3* = \mathcal{R}_3(\boldsymbol{x}_3)$ No further improvement
End of third loop. Found an optimal solution of the third LP with minimal tight set. Check stopping condition. $\mathcal{H}_3^*$ has full rank. Nucleolus found!			

Table 2: Step by step for computing the nucleolus of a small flow game with 10 players.

n	$\boldsymbol{\omega}$	κ	Computational Time						# LPs	# Iters
			Total	LPs	REP	STS	RLP	CG		
25	χ_1^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	5.4109	5.3203	0.040625	0.05	0.018526	0.045369	1	83
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	5.9688	5.8	0.11094	0.05	0.021469	0.04697	1	83.7
	χ_5^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	14.623	14.603	0	0.020313	0.020199	0.14232	1	89.7
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	14.569	14.55	0	0.017188	0.021865	0.11451	1	106.3
	χ_n^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	29.045	29.019	0.003125	0.023438	0.023356	0.22803	1	115.5
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	21.559	21.534	0	0.021875	0.02397	0.16267	1	115.3
50	χ_1^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	15.336	14.6	0.60938	0.12344	0.029747	0.070045	1	145.9
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	18.47	18.155	0.18594	0.12812	0.032849	0.077401	1	163.7
	χ_5^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	32.205	32.167	0.00625	0.029687	0.033361	0.15935	1	166.7
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	33.381	33.35	0.0046875	0.026562	0.038545	0.13765	1	189.2
	χ_n^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	54.527	54.491	0.003125	0.028125	0.039877	0.21631	1	211.6
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	38.852	38.817	0	0.032813	0.041819	0.15695	1	194.7
75	χ_1^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	36.392	34.036	2.1344	0.21875	0.051033	0.10556	1	216.9
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	44.752	44.33	0.13281	0.28906	0.060297	0.11868	1	246.8
	χ_5^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	64.602	64.547	0.0015625	0.045312	0.061851	0.19325	1	252.6
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	78.616	78.561	0.003125	0.048438	0.076321	0.19012	1	294.5
	χ_n^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	141.57	141.51	0.00625	0.05	0.078862	0.34791	1	328.3
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	83.872	83.808	0.0046875	0.054688	0.080333	0.20399	1	294.3
100	χ_1^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	59.952	51.941	7.7453	0.25781	0.056447	0.1266	1	283.4
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	76.617	75.719	0.5375	0.35156	0.078371	0.14417	1	338.2
	χ_5^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	123.31	110.06	0.0078125	13.23	0.083104	0.22813	1	352.7
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	162.95	162.88	0.003125	0.054688	0.13099	0.23486	1	443.4
	χ_n^2	$\lceil 0.5\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	226.7	226.64	0.00625	0.048438	0.12469	0.33447	1	483.4
		$\lceil 0.75\boldsymbol{e}^t\boldsymbol{\omega}\rceil$	238.39	238.33	0.00625	0.05	0.11905	0.52996	1	380

Table 3: Computational results of large simulated weighted voting games

From the third column, we can observe that the total computation time increase with the number of players. The longest computational time is 238.39 seconds for the largest games with 100 players. Most of the time is taken up for solving the $\mathbf{LP_k}$ while the total times to find the representative set (**REP**) and to find improved imputation (**FBOS**) are small. Column 10 shows that these voting games require solving only $\mathbf{LP_1}$. All these LPs require less than 500 iterations in the constraint generation algorithm to find an optimal solution. This means that instead of having to solve a big LP with 2^n constraints, the CG algorithm solves less than 500 small relaxed LPs and 500 constraint generation problems. Columns 8 and 9 show that the computation times required to solve these relaxed LPs and the CG problems are relatively small. All the numerical results are tested on a personal computer with 2.67GHz CPU, 12GB RAM, and the Windows 7, 64-bit operating system. We use MATLAB for coding and use IBM CPLEX

n	(m,k,ρ)	Computational Time						# LPs	# Iters
		Total	LPs	REP	STS	RLP	CG		
25	(10,5,0.5)	9.3655	4.6613	0.69732	4.0038	0.010201	0.096578	1.8	120.15
	(10,5,0.7)	1.5202	0.68172	0.34944	0.48906	0.0064025	0.04492	1.55	58.4
	(10,5,0.9)	4.478	1.6331	1.1117	1.7218	0.019776	0.24373	3.1053	31.789
	(20,10,0.5)	28.498	19.537	3.6933	5.2643	0.027263	0.33063	3.6	156.25
	(20,10,0.7)	48.051	45.404	0.97111	1.6723	0.028752	0.23869	2.3	176.2
	(20,10,0.9)	8.9014	7.1144	0.73242	1.0538	0.01325	0.068927	1.45	85.35
50	(10,5,0.5)	56.841	51.995	3.1863	1.6458	0.10959	0.35252	1.85	416.9
	(10,5,0.7)	39.625	32.653	3.1676	3.7939	0.076558	0.20414	1.6	301.5
	(10,5,0.9)	42.276	36.509	4.4788	1.2769	0.037106	0.26389	1.8	166.05
	(20,10,0.5)	115.03	106.77	1.8861	6.3602	0.18726	0.11657	1.4	488.95
	(20,10,0.7)	114.26	89.389	19.357	5.453	0.19206	1.3062	3.85	416.6
	(20,10,0.9)	36.285	31.58	2.2605	2.4352	0.1038	0.12702	1.45	233.15
75	(10,5,0.5)	337.39	253.08	74.244	10.023	0.32213	5.4044	2.6	873.4
	(10,5,0.7)	226.29	187.49	26.857	11.914	0.23974	1.6586	1.6	728.6
	(10,5,0.9)	175.59	125.34	29.247	20.97	0.148	1.185	1.6	460.4
	(20,10,0.5)	486.62	454.16	12.717	19.7	0.37461	0.37486	1.2	1101
	(20,10,0.7)	1375.3	394.43	626.16	354.61	0.47017	5.2229	2.6	869.4
	(20,10,0.9)	172.17	136.38	19.115	16.656	0.2497	0.91194	1.4	483.7

Table 4: Computational results of large simulated weighted voting games

Studio Academic version 12.4 for solving LPs and MILP problems under default settings.

5.3 Large Coalitional Skill Games

We generate coalitional skill games with different size and parameters as follows. For each $n = \{25, 50, 75\}$, we generate random skill matrix Ψ using the binomial distribution with different success rates $\rho = \{0.5, 0.7, 0.9\}$. We generate K simulated games with $K = 20$ for $n = \{25, 50\}$ and $K = 10$ for $n = 75$. Table 4 shows the computational results for these games in details. The columns and rows shown in Table 4 have the same interpretation compared to those in Table 3. Overall, the total computational time increases as the number of players increase (the total time variation by changing other parameters are mixed). The total time is broken down into three main tasks for solving the LPs, finding the representative set and for finding solution with the minimal tight set. In most cases except for the second last row with $(n, m, k, \rho) = (75, 20, 10, 0.7)$ the task that took the most time is for solving the large LPs. The second last column shows the number of LPs involved in the nested LPs. This ranges from solving a single LP to up to 8 LPs. Among all the 300 instances generated, the worst instance with the maximum total time is under 30 minutes.

6 Conclusion

We presented a nested LPs formulation to compute the nucleolus of large cooperative games. Our methodology was based on two main innovative ideas. First, we provided a method to find an imputation with a smaller tight set from any given imputation by solving a small LP. This allowed us to find an optimal imputation with the smallest tight set, an operation that is crucial in finding the nucleolus. Second, we used representative sets of tight coalitions to deal with situations when the tight set is exponentially large. In addition, we used various techniques such as generating cuts to find multiple coalitions with the same worst excess values and using randomization to improve the computational performance. All of these ideas were developed in conjunction with the constraint generation framework that was used to solve large LPs which means issues such as dealing with multiple optimal solutions and finding the tight sets became more challenging. We demonstrated our algorithm with the general flow games, the coalitional skill games and the weighted voting games. In the coalitional skill games, the nucleoli are found in less than 30 minutes for games with up to 75 players. In the weighted voting game, the nucleoli were found in less than 240 seconds on average even for games with up to 100 players. This is a significant improvement compared to existing methods in the literature where only games with at most 20 players have been studied. Our paper sheds light on future development in the following areas. On the methodology side, the ideas of finding an imputation with smallest tight set and of using representative sets can lead to future studies on solving more general lexicographical minimization problems. In addition, as finding the core and the least core is just one small stage in finding the nucleolus, the constraint generation framework can be further studied to compute these solution concepts for more generic cooperative games. On the practical side, we expect that applications of cooperative game theory that once were hindered by the difficulty in computing the nucleolus could now be possible by applying our results.

References

[1] H. Aziz, M. Paterson, and D. Leech. Efficient algorithm for designing weighted voting games. In *Multitopic Conference, 2007. INMIC 2007. IEEE International*, pages 1–6. IEEE, 2007.

[2] Y. Bachrach, R. Meir, K. Jung, and P. Kohli. Coalitional structure generation in skill games. *AAAI-2010*, 2010.

[3] Yoram Bachrach and Jeffrey S Rosenschein. Coalitional skill games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, AAMAS '08, pages 1023–1030, Richland, SC, 2008. ISBN 978-0-9817381-1-6.

[4] A. Caprara and A.N. Letchford. New techniques for cost sharing in combinatorial optimization games. *Mathematical Programming*, 124(1):93–118, 2010.

[5] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. Computational aspects of cooperative game theory. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 5(6):1–168, 2011.

[6] P. Chardaire. The core and nucleolus of games: A note on a paper by göthe-lundgren et al. *Mathematical programming*, 90(1):147–151, 2001.

[7] X. Chen and J. Zhang. A stochastic programming duality approach to inventory centralization games. *Operations research*, 57(4):840–851, 2009.

[8] X. Deng, Q. Fang, and X. Sun. Finding nucleolus of flow game. *Journal of combinatorial optimization*, 18 (1):64–86, 2009.

[9] J. Derks and J. Kuipers. Implementing the simplex method for computing the prenucleolus of transferable utility games. 1996.

[10] E. Elkind and D. Pasechnik. Computing the nucleolus of weighted voting games. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 327–335. Society for Industrial and Applied Mathematics, 2009.

[11] E. Elkind, L.A. Goldberg, P. Goldberg, and M. Wooldridge. Computational complexity of weighted threshold games. In *Proceeding of the National Conference On Artificial Intelligence*, volume 22, page 718, 2007.

[12] U. Faigle, W. Kern, and J. Kuipers. On the computation of the nucleolus of a cooperative game. *International Journal of Game Theory*, 30(1):79–98, 2001.

[13] B. Fromen. Reducing the number of linear programs needed for solving the nucleolus problem of n-person game theory. *European Journal of Operational Research*, 98(3):626 – 636, 1997.

[14] M. Göthe-Lundgren, K. Jörnsten, and Peter Värbrand. On the nucleolus of the basic vehicle routing game. *Mathematical programming*, 72(1):83–100, 1996.

[15] SH Gow and L.C. Thomas. Interchange fees for bank atm networks. *Naval Research Logistics (NRL)*, 45(4): 407–417, 1998.

[16] D. Granot and G. Huberman. On the core and nucleolus of minimum cost spanning tree games. *Mathematical Programming*, 29(3):323–347, 1984.

[17] D. Granot and M. Maschler. Spanning network games. *International Journal of Game Theory*, 27(4):467–500, 1998.

[18] Herbert Hamers, Flip Klijn, Tamás Solymosi, Stef Tijs, and Dries Vermeulen. On the nucleolus of neighbor games. *European Journal of Operational Research*, 146(1):1–18, 2003.

[19] E. Kalai and E. Zemel. Totally balanced games and games of flow. *Mathematics of Operations Research*, pages 476–478, 1982.

[20] W. Kern and D. Paulusma. On the core and f-nucleolus of flow games. *Mathematics of Operations Research*, 34(4):981–991, 2009.

[21] E. Kohlberg. On the nucleolus of a characteristic function game. *SIAM Journal on Applied Mathematics*, 20(1):62–66, 1971.

[22] E. Kohlberg. The nucleolus as a solution of a minimization problem. *SIAM Journal on Applied Mathematics*, 23(1):34–39, 1972.

[23] A. Kopelowitz. Computation of the kernel of simple games and the nucleolous of n person games. *Technical Report RM 31. The Hebrew Uni versity of Jerusalem.*, 1967.

[24] D. Leech. Computing power indices for large voting games. *Management Science*, pages 831–838, 2003.

[25] J. Lemaire. Cooperative game theory and its insurance applications. *ASTIN Bulletin*, 21(1):17–41, 1991.

[26] M. Leng and M. Parlar. Analytic solution for the nucleolus of a three-player cooperative game. *Naval Research Logistics (NRL)*, 57:667–672, 2010.

[27] M. Maschler, B. Peleg, and L.S. Shapley. Geometric properties of the kernel, nucleolus, and related solution concepts. *Mathematics of Operations Research*, 4(4):303–338, 1979.

[28] G. Owen. A note on the nucleolus. *International Journal of Game Theory*, 3(2):101–103, 1974.

[29] Bezalel Peleg and Peter Sudhölter. *Introduction to the theory of cooperative games*, volume 34. Springer, 2007.

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

[31] J.A.M. Potters, J.H. Reijnierse, and M. Ansing. Computing the nucleolus by solving a prolonged simplex algorithm. *Mathematics of operations research*, 21(3):757–768, 1996.

[32] J.K. Sankaran. On finding the nucleolus of an n-person cooperative game. *International Journal of Game Theory*, 19(4):329–338, 1991.

[33] D. Schmeidler. The nucleolus of a characteristic function game. *SIAM Journal on applied mathematics*, 17 (6):1163–1170, 1969.

[34] A.S. Schulz and N.A. Uhan. Sharing supermodular costs. *Operations research*, 58(4-Part-2):1051–1056, 2010.

[35] T. Solymosi and T.E.S. Raghavan. An algorithm for finding the nucleolus of assignment games. *International Journal of Game Theory*, 23(2):119–143, 1994.

[36] Tamás Solymosi, TES Raghavan, and Stef Tijs. Computing the nucleolus of cyclic permutation games. *European journal of operational research*, 162(1):270–280, 2005.

Appendix A: List of acronyms for subproblems and notations

In general, we use bold font for vectors and matrices.

- $\mathbf{LP_k}$: The k^{th} LP in the nested LPs formulation.
- $\mathbf{RLP_k}$: A relaxed version of $\mathbf{LP_k}$.
- **CG**: A constraint generation problem (the separation problem) that identifies a coalition with the worst excess among a subset of coalitions.
- **FBOS**: Find an 'improved' optimal solution of $\mathbf{LP_k}$, i.e. with a smaller tight set, from a given optimal solution.
- **REP**: Find the representative tight set of an optimal solution of $\mathbf{LP_k}$. This representative set spans the entire space of all the tight constraints of that particular solution.
- $\mathcal{I}$: The set of all imputations of the game.
- $(\boldsymbol{x}, \epsilon_k^*)$: An optimal solution of $\mathbf{LP_k}$. Notice that ϵ_k^* is unique but there could be multiple optimal $\boldsymbol{x}$.
- $\mathcal{T}_k(\boldsymbol{x})$: The set of all tight constraints at a feasible solution $\boldsymbol{x}$ of $\mathbf{LP_k}$. It is also referred to as the tight coalitions.
- $\boldsymbol{x}_k^*$: An optimal solution whose tight set $\mathcal{T}_k(\boldsymbol{x}_k^*)$ has the smallest size.
- $\mathcal{T}_k^*$: The minimal tight set, i.e. $\mathcal{T}_k^* = \mathcal{T}_k(\boldsymbol{x}_k^*)$.
- $\mathcal{R}_k^*$: The representative tight set at $\boldsymbol{x}_k^*$ in $\mathbf{LP_k}$, i.e. $\mathcal{R}_k^* \subset \mathcal{T}_k^*$ and $|\mathcal{R}_k^*| = rank(\mathcal{R}_k^*) = rank(\mathcal{T}_k^*)$, where $|\mathcal{R}_k^*|$ is the size of the set $\mathcal{R}_k^*$.
- $\boldsymbol{e} = (1, \ldots, 1)^t \in \mathbb{R}^n$: The identity vector in $\mathbb{R}^n$ of all ones.
- $\boldsymbol{z} = (z_1, \ldots, z_n)^t \in \{0, 1\}^n$: An alternative representation of a coalition $\mathcal{S}$ where z_i is a binary variable that indicates whether player i belongs to the coalition $\mathcal{S}$.

Appendix B: Constraint Generation Algorithm for Solving Large $\mathbf{LP_k}$

In this section we aim to solve $\mathbf{LP_k}$. Since the LP has an exponentially large number of constraints, it is often very difficult to solve. However, we notice that most of the constraints are non-binding at optimal solutions. That means we can still find an optimal solution by including only a subset of constraints. The idea of the constraint generation (CG) algorithm is to start with a relaxed problem and then to check whether the optimal solution of the relaxed problem satisfies all the constraints in the original problem. If that is the case, then the solution to the relaxed problem is also an optimal solution of the original problem. Otherwise, we have identified a violating constraint and that can be added to the relaxed problem. Let $\mathcal{C}_r \subset \mathcal{C}$ be a subset of all coalitions (ideally $|\mathcal{C}_r| \ll |\mathcal{C}|$). We relax the inequality constraint $\epsilon_k + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{C} \backslash \mathcal{H}^*_{k-1}$ in $\mathbf{LP_k}$ as follows:

$$\mathbf{RLP_k} := \min_{\boldsymbol{x} \in \mathcal{I}, \epsilon_k} \quad \{\epsilon_k \mid (\boldsymbol{x} - \boldsymbol{x}^*_{k-1})(\mathcal{S}) = 0,\ \forall \mathcal{S} \in \mathcal{H}^*_{k-1},\ \epsilon_k + \boldsymbol{x}(\mathcal{S}) \geq v(\mathcal{S}),\ \forall \mathcal{S} \in \mathcal{C}_r \backslash \mathcal{H}^*_{k-1}\}.$$

Once we have obtained an optimal solution $(\boldsymbol{x}, \epsilon)$ for the relaxed LP, we will solve the constraint generation problem: $\mathbf{CG} := \max_{\mathcal{S} \in \mathcal{C} \backslash \mathcal{H}^*_{k-1}} \{v(\mathcal{S}) - \boldsymbol{x}(\mathcal{S})\}$, and let $\mathcal{S}^*$ be an optimal solution of the constraint generation problem. If $v(\mathcal{S}^*) - \boldsymbol{x}(\mathcal{S}^*) \leq \epsilon$, then $(\boldsymbol{x}, \epsilon)$ satisfies all the constraints of $\mathbf{LP_k}$ and hence it is also an optimal solution of $\mathbf{LP_k}$. The CG algorithm then terminates and we have found an optimal solution. Otherwise, we can introduce $\mathcal{S}^*$ to the relaxed constraint set $\mathcal{C}_r$ and repeat the process. The formal constraint generation algorithm is described below: An attractive property of the

Algorithm 2: The Constraint Generation Method for solving $\mathbf{LP_k}$

1. Initialization: Find any initial imputation vector $\boldsymbol{x}^0$ that solves $\mathbf{LP_{k-1}}$, set the relaxed set $\mathcal{C}_r = \emptyset$, set the lower bound $\epsilon^0 = -\infty$, upper bound $\tau^0 = \infty$, and set the iterative index $j = 0$.;

while $\tau^j \neq \epsilon^j$ **do**

2. Let $j = j + 1$ and solve the constraint generation problem **CG**:

$$\tau^j = \max_{\mathcal{S} \in \mathcal{C} \backslash \mathcal{H}^*_{k-1}} \{v(\mathcal{S}) - \boldsymbol{x}^{j-1}(\mathcal{S})\}, \text{ and let } \mathcal{S}^j = \operatorname*{argmax}_{\mathcal{S} \in \mathcal{C} \backslash \mathcal{H}^*_{k-1}} \{v(\mathcal{S}) - \boldsymbol{x}^{j-1}(\mathcal{S})\}.$$

3. Add $\mathcal{S}^j$ to the relaxed set $\mathcal{C}_r$;

4. Solve the relaxed problem $\mathbf{RLP_k}$ and let $(\boldsymbol{x}^j, \epsilon^j)$ be an optimal solution;

end

5. Terminate the algorithm and record $(\boldsymbol{x}^j, \epsilon^j)$ as an optimal solution of $\mathbf{LP_k}$.;

constraint generation algorithm is that the lower bounds ϵ^j are increasing through iterations. We can show that the constraint generation algorithm never enters a loop of reintroducing coalitions into $\mathcal{C}_r$ before the final iteration. The proof for the correctness of the stopping condition in the iterative loop is quite standard and we skip for brevity. In addition, since the set $\mathcal{C}$ of all the possible coalitions is finite, the constraint generation algorithm will terminate at an optimal solution. This is independent of the starting imputation $\boldsymbol{x}^0$.