Computers and Operations Research 112 (2019) 104764

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

A heuristic procedure for computing the nucleolus

Federico Perea[a,*], Justo Puerto[b]

[a] Grupo de Sistemas de Optimización Aplicadas, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, València, 46021, Spain
[b] Instituto de Matemáticas de la Universidad de Sevilla. Avda. Reina Mercedes, s/n, Sevilla, 41012, Spain

ARTICLE INFO

Article history:
Received 13 November 2018
Revised 27 June 2019
Accepted 3 August 2019
Available online 6 August 2019

Keywords:
Nucleolus
Game theory
Linear programming
Heuristic

ABSTRACT

This paper introduces a row and column generation algorithm for finding the nucleolus, based on a linear programming model proposed in an earlier research. Since this approach cannot return an allocation for large games, we also propose a heuristic approach, which is based on sampling the coalitions space. Experiments over medium sized games show that the proposed heuristic finds allocations which are close to the true nucleolus, in a reasonable amount of time. Experiments over 100-player games show that the proposed heuristic can be applied to games of large size.

1. Introduction

A transferable utility (TU) game can be defined by means of:

- A set of *players* $N = \{1, \ldots, n\}$. Players are allowed to cooperate, but their objective is to maximize their own individual benefit.
- For each *coalition* $S \subset N$, the characteristic function $v(S)$ represents the profit that the cooperation of the players in S yields, without the help of the other $N \backslash S$ players. The set of all players N is referred to as the *grand coalition*.

TU-games can therefore be identified by means of their set of players and characteristic function: (N, v).

One of the main challenges in TU-games consists of sharing the profit that the grand coalition can make, among the different players. There are two main ways to address that question:

1. Using solution sets.
2. Using allocation rules.

The most well-known representative of the first type is the *core* (Owen, 1995). The core of a TU-game is defined as

$$C(v) = \{x \in \mathbb{R}^n : x(S) \geq v(S) \ \forall \ S \subset N, \ x(N) = v(N)\}.$$

Core allocations ensure that each coalition S gets a share of the profit obtained by the grand coalition which is, at least, as high as the profit that S can make on their own. Thus, core allocations are well accepted due to the fairness conditions they satisfy.

However, there are games without core allocations. Even for some games which do have core allocations, it might be very difficult to find them or, choose one among them. Therefore, at times one is interested in the second type of solutions: Allocation rules. Allocation rules are procedures that allocate to each player a share of the benefit obtained by the grand coalition. Two important such rules are the Shapley value and the nucleolus, which are well accepted for the properties they satisfy, and are widely used in complex TU-games (e.g. see Yu et al., 2017, where the Shapley value is computed in a pickup and delivery cooperative game). The reader should note that the Shapley value belongs to the core when the game is convex, and the nucleolus belongs to the core whenever the core is non-empty.

In this paper, we focus on the nucleolus (Kohlberg, 1972), which we now introduce for the sake of completeness.

Given a TU-game (N, v), the set of pre-imputations $\tilde{V}$ and imputations V of the game are:

$$\tilde{V} = \left\{ x \in \mathbb{R}^n : \sum_{j=1}^{n} x_j = v(N) \right\},$$

$$V = \left\{ x \in \mathbb{R}^n : \sum_{j=1}^{n} x_j = v(N), x_j \geq v(\{j\}), \ \forall \ j \in N \right\}.$$

Note that $V \subset \tilde{V}$. Given a pre-imputation $x \in \tilde{V}$, the excess vector of x is the vector $\theta(x) \in \mathbb{R}^{2^n-2}$

$$\theta(x) = (e(S, x)), \quad \text{with } e(S, x) = v(S) - \sum_{i \in S} x_i \ \forall \ S \subset N, \ S \neq \emptyset, N.$$

After introducing these two concepts, the nucleolus can be defined.

* Corresponding author.
E-mail addresses: perea@eio.upv.es (F. Perea), puerto@us.es (J. Puerto).

https://doi.org/10.1016/j.cor.2019.104764
0305-0548/

Definition 1.1. The (pre)nucleolus is the unique (pre)imputation that lexicographically minimizes ($<_L$) the non-increasingly sorted excess vector.

The nucleolus satisfies the following two properties:

- If $e(S, x) \leq 0 \, \forall S \in 2^N$, then x is a core allocation.
- Provided the core is non-empty, the nucleolus is a core allocation.

Despite the huge complexity of computing the nucleolus, several attempts have been made in order to compute this solution concept by means of a single linear programming (LP) model. Kohlberg (1972) computes the nucleolus from a LP problem with $O(2^n!)$ constraints. A bit later, Owen (1974) reduces the size of this problem to $O(4^n)$ constraints and $O(2^n)$ variables with large constraint coefficients. Puerto and Perea (2013) propose another single LP problem for computing the nucleolus, with coefficients in $\{-1, 0, 1\}$.

The extreme complexity of the computation of the nucleolus has provoked that more attempts have been made in the area of iterative approaches. To cite a few, Maschler et al. (1979) propose an algorithm that solves $O(4^n)$ LP problems with $O(2^n)$ variables and constraints, whose coefficients are -1, 0, or 1. Dragan (1981) finds the nucleolus by solving at most $n-1$ LP problems with $O(n)$ constraints and $O(2^n)$ variables. Sankaran (1991) proposes a new algorithm that needs $O(2^n)$ LP problems whose coefficients are -1, 0, or 1. Solymosi (1993) proves that the nucleolus can be found by solving at most $n-1$ LP's with $O(n)$ constraints and $O(2^n)$ variables. Later on, Hallefjord et al. (1995) introduce a constraint generation approach. Potters et al. (1996) describe a fast algorithm to find the nucleolus of any game with non-empty imputation set, based on solving at most $n-1$ LP's with at most $2^n + n - 1$ constraints and $2^n - 1$ variables. More recently, Nguyen and Thomas (2016) use nested linear programs in their approach to find the nucleolus of large games. The list of references including exact procedures for finding the nucleolus is very large. However, it must be noted that not all the articles published propose correct procedures to find the nucleolus, as can be derived from Guajardo and Jörnsten (2015). The authors of that paper show mistakes in some of the algorithms proposed in the literature.

In general, finding the nucleolus is a problem of exponential complexity. However, for some classes of games this solution concept can be found efficiently. We now review (non exhaustively) the literature dealing with the nucleolus for specific classes of games. Hamers et al. (2003) prove that the nucleolus of neighbor games can be computed by a cuadratic-order algorithm. Solymosi et al. (2005) study the nucleolus of permutation games, and prove its polynomial complexity under certain conditions. Brânzei et al. (2006) propose a cuadratic algorithm for computing the nucleolus of airport games. Deng et al. (2009) prove that the nucleolus of flow games can be computed in polynomial time, only when the game is defined on simple networks. Maschler et al. (2010) study the nucleolus of tree games. van den Brink et al. (2011) propose a polynomial time algorithm for computing the nucleolus of games in which some players need permission from other players, in order to enter the game. Martínez-De-Albéniz et al. (2013) compute the nucleolus of assignment games. Greco et al. (2014) characterize the complexity of the nucleolus on compact coalitional games. Kurz et al. (2014) study the nucleolus of majority games. Hou and Driessen (2015) use the indirect function of a cooperative game in characteristic function form in order to compute the nucleolus of compromise stable games. Kamiyama (2015) studies the nucleolus of arborescence games, proving that it can be found polynomially when the graph is acyclic and directed. Aiche et al. (2015) examine the nucleolus of a class of market games, and compare it with the Shapley value. Fang et al. (2015) propose a polynomial time algorithm for the nucleolus of path cooperative games. Fang et al. (2016) compute the nucleolus for threshold cardinality matching games, which is done polynomially for some types of graphs. Sziklai et al. (2017) study the nucleolus of directed acyclic graph games. Baïou and Barahona (2017) propose a polynomial time algorithm for computing the nucleolus of shortest path games.

Despite the vast literature proposing exact methods for computing the nucleolus, we have barely found three references that propose non-exact approaches for this solution concept, or variations of it. Chin (1997) proposes a genetic algorithm for computing the nucleolus of the specific class of assignment games. Kimms and Çetiner (2012) suggest a heuristic variation of an algorithm they propose for computing the nucleolus, which is based on constraint generation. However, the authors of that paper discard this heuristic approach because "there is no guarantee that using a heuristic would be more efficient" than the exact approach. Flisberg et al. (2015) propose cost allocation methods to solve cost sharing problems in a forest fuel transportation problem. Among them, they adapt the nucleolus when the characteristic function is incomplete. Wang et al. (2017) propose several nucleolus-based allocations, and a genetic algorithm for finding them.

As can be seen there is lack of good heuristic procedures that can provide reasonable approximations of the nucleolus for general purpose TU games, and this is one of the major motivations of this work.

Arguably, the nucleolus is one of the most well-known allocation rules in cooperative game theory. At the same time, its huge computational complexity has prevented many practitioners from applying it. Consider for example a game in which the players are the n bank entities operating on a given region. The characteristic function of each coalition would be the cost for these banks to operate an ATM network needed to serve their clients. Whenever n is large enough (for example $n = 50$, which is a realistic number for this example), even getting and storing the characteristic function of the game would be a hard task, not to mention finding the excess vector, sorting it, etc. For this reason, in this paper we introduce a heuristic approach for finding the nucleolus of a game, which does not rely on the complete knowledge of the characteristic function. Heuristic algorithms are efficient procedures which find a solution to a problem in a reasonable amount of time, although the solution returned is not guaranteed to be optimal. In our case, the allocation returned by the heuristic proposed is not guaranteed to be the nucleolus. However, as we will see in the experiments section, the allocation returned by our heuristic is *close* to the true nucleolus.

The rest of this paper is structured as follows. We begin by introducing a variable/constraint generation algorithm in Section 2, based on the LP model introduced in Puerto and Perea (2013). Because this approach does not seem very promising, especially for large games, we add a sampling phase to it in Section 3. In the same section, we propose a heuristic approach, which aims at finding an allocation close to the nucleolus, for large games in which exact procedures cannot be applied. All approaches proposed are computationally tested in Section 4, over a number of games randomly generated. The paper closes with some conclusions and the list of references.

2. A column/row generation algorithm

Puerto and Perea (2013) proved that the nucleolus can be found by means of the following LP problem:

$$\min \sum_{k=1}^{2^n-2} (\lambda_k - \lambda_{k+1}) \left(k t_k + \sum_{i=1}^{2^n-2} d_{ik} \right) \tag{1}$$

Table 1
Percentage of binding C_{ik} constraints in LP models.

n	Percentage %	Distribution
10	0.04	(1,4,3,1,0,...)
11	0.02	(0,7,1,1,1,0,...)
12	0.01	(0,5,1,1,1,0,...)
13	< 0.01	(0,7,1,2,2,0,0,0,1,0...)
14	< 0.01	(0,5,0,3,2,0,1,1,0,...)
15	< 0.01	(0,6,1,3,1,0,1,1,2,0,...)
16	< 0.01	(0,7,1,2,0,1,2,0,3,0,...)

$$\text{s.t.} \quad d_{ik} \geq \theta_i - t_k, \ \forall\ i, k = 1, \ldots, 2^n - 2, \tag{2}$$

$$\theta_i = v(S_i) - \sum_{j \in S_i} x_j, \forall\ i = 1, \ldots, 2^n - 2, \tag{3}$$

$$\sum_{j=1}^{n} x_j = v(N),$$

$$d_{ik} \geq 0, \ \forall\ i, k = 1, \ldots, 2^n - 2, \tag{4}$$

with parameters λ satisfying that $\lambda_k = \delta^{k-1}, k = 1, \ldots, 2^n - 2$, for a convenient choice of δ, and $\lambda_{2^n-1} = 0$. Note that finding a proper value of δ is key. Choosing a value too large might provoke numerical inaccuracy, whereas choosing a value too small might lead to δ^k being considered as zero by computer precision, even for small values of k.

In this LP problem, indexes i and k both refer to coalitions, whereas index j refers to players. Variable x_j stands for the allocation assigned to player j, variable θ_i refers to the excess of coalition i, and variable t_k is the kth excess, lexicographically sorted. There is no straightforward interpretation of d_{ik}.

The enormous number of variables and constraints makes this model intractable for a relatively large number of players (the aforementioned paper only reports results over 18-player games or less). Note that there are $O(4^n)$ variables d_{ik} and constraints (2). We denote C_{ik} the constraint (2) for a given i and a given k.

One immediate question that comes to our mind is: how many constraints C_{ik} are binding in the optimal solution to this LP model? In this paper, we say that a constraint is binding if it does not have any slack. Note that our concept of binding constraint does not require that all potentially multiple solutions are binding on this constraint. Therefore, in this case, C_{ik} is binding if $d_{ik} = \theta_i - t_k$. Note that, if a constraint C_{ik} is not binding, then the corresponding $d_{ik} = 0$. This is due to the fact that since we are minimizing, and $\lambda_k - \lambda_{k+1} > 0$, if $\theta_i - t_k \geq 0$, then d_{ik} attains this value and the constraint is binding. Otherwise, $d_{ik} = 0$ and the constraint is not binding. In the latter case, no need to define non-binding C_{ik} constraints, nor their corresponding d_{ik} variables. In order to gain more insights into this matter, we solved the LP models for the games in Puerto and Perea (2013), and checked how many C_{ik} constraints were binding, considering only the first $k_{\max} = 20$ largest excesses, that is, $k = 1, \ldots, k_{\max}$. The results of this experiment are shown in Table 1.

Column "Percentage" indicates the relative frequency of C_{ik} constraints which are binding (all of them way below 1%). Column "Distribution" indicates the number of binding constraints C_{ik} in terms of the size of the coalitions i that make these constraints binding. The jth component of each such vector is the number of constraints of size j which are binding. For example, for the game with 10 players, one coalition with one player is binding, four coalitions with two players are binding, three coalitions with three players are binding, and one coalition with four players is binding (for larger coalitions, none of them is binding, which is indicated by "... ").

These results show a promising conclusion: only very few C_{ik} constraints are binding. Besides, we have an indication that most of these binding constraints correspond to coalitions i with a "small" size. However, how to find these pairs (i, k) such that C_{ik} is binding?

A first approach to try to answer such question consists of introducing variables d_{ik} and constraints C_{ik} only when the corresponding C_{ik} constraint is violated, a so called row-column generation algorithm (RCG). For this, define the set $A \subset 2^N \times 2^N$. A pair (i, k) is in A if the corresponding variable d_{ik} and constraint C_{ik} are in the model. The LP programs to be solved in the iterative method we present are:

$$\min \quad \sum_{k=1}^{2^n-2} (\lambda_k - \lambda_{k+1}) \left(k t_k + \sum_{i:(i,k) \in A} d_{ik} \right) \tag{5}$$

$$LP(A): \ \text{s.t.} \quad d_{ik} \geq \theta_i - t_k \ \forall\ (i, k) \in A,$$
$$(3)\ \textit{and}\ (4)$$
$$d_{ik} \geq 0 \ \forall\ (i, k) \in A, \tag{6}$$

with $\lambda_k = \delta^{k-1}, k = 1, \ldots, 2^n - 2$ and $\lambda_{2^n-1} = 0$. This problem is denoted as $LP(A)$.

Note that the previous model implies that $d_{ik} = 0$ for all $(i, k) \notin A$, since $\lambda_k - \lambda_{k+1} > 0$, and d_{ik} must be non-negative. The algorithm first sets $A = \emptyset$. Then it updates A to include those $(i, k) \notin A$ such that $\theta_i - t_k > 0$ (note that, for these pairs, $d_{ik} < \theta_i - t_k$ and therefore this constraint would be violated in the original LP). The process is repeated until there are no more constraints violated. When such convergence is achieved, the solution returned is the nucleolus. This is true because the optimal solution to the relaxed problem (the one in which not all constraints are necessarily present) satisfies all the constraints (even those which are not imposed) of the full problem. Therefore, the optimal solution to the relaxed problem is also an optimal solution to the full problem. As proved in Puerto and Perea (2013), the optimal solution to the full problem is the nucleolus.

Algorithm 1 shows a pseudocode of this method.

Algorithm 1: Pseudo-code of the row/column generation algorithm for computing the nucleolus.

```
Data: The characteristic function of a game
Feasible = 0, A = ∅;
while Feasible = 0 do
    Feasible = 1;
    solve LP(A) → x*, t*, θ*;
    for i, k : (i, k) ∉ A do
        if θ*_i − t*_k > 0 then
            A = A ∪ {(i, k)}, Feasible = 0
        end
    end
end
Result: The nucleolus: x*
```

2.1. Preliminary experiments

Experiments over the games introduced in Puerto and Perea (2013), with number of players ranging from 10 to 16 and $k_{\max} = 20$, are summarized in Table 2. Column "Card A" shows the number of coalitions added in the last iteration of the algorithm (typically 3 iterations were needed to find the nucleolus). Column "Percentage %" reports the percentage of coalitions added in the algorithm, over the total number of possible coalitions.

Table 2
Number of C_{ik} coalitions used in the last iteration.

n	Card A	Percentage %
10	10 626	51.98
11	20 915	51.11
12	41 623	50.83
13	83 012	50.67
14	165 200	50.42
15	329 455	50.27
16	657 958	50.19

We also noted that, for these instances, the total CPU time does not vary much between the single LP and the row/column algorithm. We also note that the percentage of variables d_{ik} and its corresponding constraints included in the final iteration is roughly 50% of the total. Therefore, because less variables and constraints than in the full LP model are needed, we expect that this sequential procedure will be able to compute the nucleolus for games with larger number of players, with respect to the size of the games that the full LP can address. Nevertheless, 50% of the total number of 2^n constraints is still too much, if the games considered are large enough. This is why, in the next section we propose a more effective approach.

3. Combining row/column generation with sampling.

In this section we modify the previous row-column generation algorithm proposed before, by starting the algorithm with a set $A = I \times \{1, \dots, k_{\max}\}$, where I is a sample of randomly selected coalitions (index i) and $k_{\max}$ represents the index k associated to the coalition with the $k_{\max}$th largest excess, instead of $A = \emptyset$. This came to our mind because in the first iteration of the algorithm in Section 2, lots of coalitions were added to A, and very few are added in the following iterations (very few constraints are violated). This might be explained by the fact that only $(2n-1)$ coalitions are needed for computing the nucleolus, as proved by Granot et al. (1998), Reijnierse and Potters (1998), and therefore only a few constraints are actually active in our LP model. Therefore, if we start the algorithm with a set of coalitions that will lead to an allocation close to the nucleolus, we expect that very few constraints will be violated.

However, due to the different LP models (in the different iterations) that we have to solve, the total running times of this algorithm are quite similar to the running times of the original LP model. This fact will be tested more extensively in the experiments section, considering different sampling procedures.

3.1. A heuristic procedure based on sampling

In this section we propose a heuristic approach to compute an allocation that is expected to be close to the nucleolus, in a reasonable amount of time. Such approach consists of stopping the previous algorithm in the first iteration. This way, we do not need complete knowledge of the characteristic function, nor we need to check if all possible C_{ik} constraints are satisfied. Note that the gain in CPU time is immense, as we do not have to compute nor store the exponentially increasing characteristic function. Besides, as we will see in the experiment section, the allocations obtained are fairly close to the true nucleolus. An added value of this method is that one can control the size of the sigle LP problem to be solved by means of the size of the sample taken (set I) and $k_{\max}$. The only LP problem to be solved in this heuristic approach consists of:

$$\min \quad \sum_{k=1}^{k_{\max}} (\lambda_k - \lambda_{k+1}) \left(k t_k + \sum_{i \in I} d_{ik} \right) \tag{7}$$

$$\text{s.t.} \quad d_{ik} \geq \theta_i - t_k \ \forall\ i \in I, k = 1, \dots, k_{\max}, \tag{8}$$

$$\theta_i = v(S_i) - \sum_{j \in S_i} x_j, \forall\ i \in I, \tag{9}$$

$$\sum_{j=1}^{n} x_j = v(N),$$

$$d_{ik} \geq 0, \ \forall\ i \in I, k = 1, \dots, k_{\max}. \tag{10}$$

Note how the size of the LP problem has decreased to from $O(4^n)$ to $O(|I|k_{\max})$ variables and constraints. The reader may note that choosing an appropriate set I is a key aspect of this algorithm. Therefore, in the experiments section, several sampling techniques will be applied, for selecting set I. Besides, different sample sizes will also be tested and compared.

Other stopping criteria, like for example the number of constraints violated, or the proportion of such unsatisfied constraints, etc. are indeed interesting. Unfortunately, they require the knowledge of the complete characteristic function for all coalitions. Therefore, we do not apply these stopping criteria in our heuristic approach (which intends to find an allocation with excess vector close to that of the nucleolus, for very large games).

As a summary of this section, we have proposed one exact algorithm (which stops when the solution returned does not violate any of the C_{ik} constraints of the original LP problem) and a heuristic algorithm (which stops after the first iteration).

4. Experiments

In this section we summarize the computational experience we conducted in order to assess the algorithms proposed. All experiments are carried out on a desktop PC, with an Intel i7 processor at 4.2 GHz, 16 GBytes of RAM, running Windows 10 Enterprise 64 bits OS. Coding is done in GAMS 25.0.2, and the solver used is CPLEX 12.8. The analysis of results is done with the help of RStudio.

4.1. Instance generation

In order to test the algorithms proposed, we have built the following sets of instances:

- Random 12-player instances: a set of one hundred 12-player TU games have been randomly generated, in such a way that:
  - $v(S) \in \{1, 2, \dots, 9\}, \ \forall\ S \subset N, \ S \neq \emptyset, N$.
  - $v(\emptyset) = 0, \ v(N) = 15$.
- Balanced 12-player instances: a set of one hundred 12-player games, such that they have non-empty core, built in the following way:
  - A random allocation $x^c \in \mathbb{Z}^{12}$ is built, in such a way that for every player j, $x^c_j \in \{0, 5\}$, following a uniform distribution.
  - The characteristic function is built in such a way that x^c is a core allocation, as follows: $v(S) \in \{0, \dots, \sum_{j \in S} x^c_j\}$, for all $S \in 2^N$, $S \neq N$, and $v(N) = x^c(N)$.

  Note that $x^c(S) \geq v(S), \forall S$, and therefore the game has a non-empty core.
- The 18-player game defined in Puerto and Perea (2013) has been analyzed as well.
- Random 100-player instances: a set of ten 100-player TU games, where the characteristic function is built in the same way as the 12-player random instances.

4.2. Algorithm parameters

The algorithms proposed mainly depend on two factors: the type of sampling used to generate the set I, and the size of that set. We now detail these two factors and specify their levels considered.

- Factor 1: type of sampling. We have tested three types of sampling:
  1. Totally random: each coalition has the same probability of being chosen. This is denoted as "Random" sampling, or "Type1" sampling.
  2. Sampling per size, only small: We select the same number of coalitions of each size, only if the size is less than or equal to $n/2$. Coalitions of size greater than $n/2$ are not chosen. This is denoted as "Size_Small" sampling, or "Type2" sampling. This is justified by Table 1, since only "small" coalitions are binding in constraints C_{ik}.
  3. Sampling per size, all: We select the same number of coalitions of each size, and all sizes are eligible. This is denoted as "Size_All" sampling, or "Type 3 sampling".
  4. Semicore sampling: All semicore coalitions (those of size 1 and those of size $n-1$) are always chosen. The other coalitions until completing the sample are chosen randomly, like in Type 1 sampling. This is denoted as "Semicore" sampling, or "Type4" sampling.
- Factor 2: sample size. Regarding the sample size, we have tested the following values:
  - for the 12-player instances, $|I| \in \{100, 200, \ldots, 1000\}$ (ranging from 2.4% to 24.4% of all coalitions).
  - for the 18-player instance, $|I| \in \{500, 1000, \ldots, 5000\}$ (ranging from 0.19% to 1.91% of all coalitions).
  - for the 100-player instances, $|I| \in \{1000, 2000, \ldots, 10000\}$ (ranging from $7.8 \cdot 10^{-26}$% to $7.8 \cdot 10^{-25}$% of all coalitions)

  We emphasize here that coalitions are re-sampled from one size to the next, meaning that (for example) the 200 coalitions sampled for size 200 do not necessarily contain the 100 coalitions sampled for size 100.

Combining the three types of sampling with the 10 different sample sizes, we have in total 40 different versions of our RCG algorithm and 40 versions of our heuristic.

4.3. Experiments over 12-player random instances

For each of the 100 games in the 12-player random set, the true nucleolus has been computed by the RCG algorithm combined with sampling (for each sample size and type of sampling), as well as the allocation given by each of the 30 versions of our heuristic, using for $k_{\max} = 20$ as in Puerto and Perea (2013). Besides, in order to check if the number of iterations of the row-column algorithm depends on $k_{\max}$, different values of this parameter have been tested for the exact approach.

4.3.1. RCG results

We first analyze the results obtained for the row-column generation algorithm, for which we run the 100 instances for all sample sizes and all sampling types described before. In order to check if the value of $k_{\max}$ affects the number of iterations and/or the CPU time of this algorithm, we also tested three different values of $k_{\max} \in \{10, 20, 30\}$. The results are shown in Tables 3 and 4. For each value of $k_{\max}$ tested, columns "Size" and "Type" refer to the levels of these factors which define the sampling used in the first iteration of the RCG algorithm. Column "Iter" refers to the average number of iterations needed by the exact algorithm, and column "$Time_e$" refers to the CPU time used by the exact algorithm.

Table 3
Average results for the exact approach, for sampling types 1 and 2, each sample size, and different values of $k_{\max}$.

Sampling		$K_{\max} = 10$		$K_{\max} = 20$		$K_{\max} = 30$	
Size	Type	$Time_e$	Iter	$Time_e$	Iter	$Time_e$	Iter
100	1	5.39	6.59	5.98	6.59	7.36	6.92
200	1	3.61	5.18	3.81	5.18	4.02	4.55
300	1	1.76	2.05	1.83	2.05	2.23	2.31
400	1	1.81	2.09	1.86	2.09	2.64	2.79
500	1	2.18	2.63	2.25	2.63	2.63	2.63
600	1	2.12	2.49	2.23	2.49	2.73	2.62
700	1	2.30	2.56	2.36	2.56	2.63	2.40
800	1	2.48	2.78	2.56	2.78	3.10	2.80
900	1	2.72	3.03	2.82	3.03	3.58	2.93
1000	1	2.82	2.98	2.92	2.98	3.30	2.81
Avg. Type	1	2.72	3.24	2.86	3.24	3.42	3.28
100	2	1.59	1.46	2.24	1.95	1.59	1.46
200	2	1.75	3.88	2.21	3.80	1.75	3.88
300	2	1.88	3.57	2.88	4.10	1.88	3.57
400	2	2.02	3.76	3.00	4.05	2.02	3.76
500	2	2.12	3.98	2.86	3.52	2.12	3.98
600	2	2.02	3.18	3.10	3.64	2.02	3.18
700	2	2.15	3.34	3.04	3.42	2.15	3.34
800	2	2.21	3.19	3.50	3.50	2.21	3.19
900	2	2.18	2.75	3.17	3.23	2.18	2.75
1000	2	2.20	3.00	3.74	3.87	2.20	3.00
Avg. Type	2	2.01	3.21	2.97	3.51	2.01	3.21

Table 4
Average results for the exact approach, for sampling types 3 and 4, each sample size, and different values of $k_{\max}$.

Sampling		$K_{\max} = 10$		$K_{\max} = 20$		$K_{\max} = 30$	
Size	Type	$Time_e$	Iter	$Time_e$	Iter	$Time_e$	Iter
100	3	1.53	1.38	2.00	1.77	2.25	1.98
200	3	2.29	3.70	2.54	3.96	2.97	4.44
300	3	2.17	4.03	2.88	4.24	3.46	4.69
400	3	1.95	3.11	2.55	3.41	3.90	5.01
500	3	2.04	3.13	5.02	6.42	5.46	5.59
600	3	2.07	3.13	2.80	3.42	3.65	3.53
700	3	2.18	3.34	2.96	3.55	3.76	3.59
800	3	2.23	3.18	3.01	3.50	3.56	3.50
900	3	2.21	3.10	3.18	3.53	3.68	3.47
1000	3	2.24	3.01	3.06	3.19	3.46	3.05
Avg. Type	3	2.07	3.13	2.99	3.73	3.47	3.90
100	4	1.74	1.27	2.20	2.11	2.66	2.00
200	4	2.32	3.81	3.11	4.65	3.93	4.53
300	4	2.13	3.18	3.10	3.72	3.80	4.08
400	4	1.99	2.85	2.38	3.07	3.46	3.78
500	4	2.06	2.68	2.26	2.72	2.92	2.86
600	4	2.04	2.42	2.39	2.75	3.06	2.86
700	4	2.09	2.46	2.57	2.85	3.42	3.11
800	4	2.23	2.64	2.81	3.15	3.40	2.96
900	4	2.27	2.67	2.67	2.74	3.62	2.99
1000	4	2.40	2.88	2.85	2.88	3.59	2.81
Avg. Type	4	2.13	2.69	2.63	3.06	3.39	3.20

From our computational experience, we cannot conclude any clear link between the value of $k_{\max}$ and the number of iterations needed by the exact approach. It seems that the RCG algorithm starting with Type 2 sampling yields the best average results in terms of CPU time, for $k_{\max} \in \{10, 30\}$. Starting with Type 4 sampling seems to be best for the other value of $k_{\max}$. In terms of the number of iterations, the best average results are obtained when using Type 4 sampling, regardless the value of $k_{\max}$ employed.

4.3.2. Heuristics results

For each instance, and each version of our heuristic, different outputs will be analyzed, which include the needed CPU time by each algorithm, and the quality of the allocation x returned by the heuristics. In order to test the quality of the solution returned by

Table 5
Average results over the random 12-player instances for sampling types 1 and 2, each sample size, and $k_{\max} = 20$.

Sampling		Heuristic						Exact	
Size	Type	RD_a	RD_e	C %	$\max_a$	$\min_a$	$Time_h$	$Time_e$	Iter
100	1	0.88	0.18	10.36	2.34	0.14	1.23	5.98	6.59
200	1	0.77	0.14	10.86	2.07	0.12	1.22	3.81	5.18
300	1	0.73	0.13	3.08	1.98	0.11	1.25	1.83	2.05
400	1	0.69	0.12	4.00	1.85	0.11	1.28	1.86	2.09
500	1	0.68	0.11	6.56	1.78	0.10	1.30	2.25	2.63
600	1	0.62	0.11	8.02	1.67	0.09	1.34	2.23	2.49
700	1	0.64	0.10	8.99	1.72	0.09	1.41	2.36	2.56
800	1	0.63	0.10	10.99	1.67	0.11	1.41	2.56	2.78
900	1	0.63	0.10	11.65	1.68	0.09	1.47	2.82	3.03
1000	1	0.58	0.09	13.01	1.61	0.07	1.52	2.92	2.98
Avg. Type	1	0.68	0.12	8.75	1.84	0.10	1.34	2.86	3.24
100	2	0.62	0.15	2.15	1.58	0.09	1.22	2.24	1.95
200	2	0.45	0.09	2.10	1.21	0.08	1.21	2.21	3.80
300	2	0.35	0.06	4.53	0.96	0.04	1.24	2.88	4.10
400	2	0.27	0.04	5.81	0.74	0.03	1.27	3.00	4.05
500	2	0.24	0.04	7.44	0.63	0.02	1.31	2.86	3.52
600	2	0.23	0.03	9.23	0.61	0.02	1.36	3.10	3.64
700	2	0.21	0.02	9.95	0.57	0.01	1.38	3.04	3.42
800	2	0.17	0.02	10.90	0.47	0.01	1.49	3.50	3.50
900	2	0.17	0.02	11.25	0.46	0.00	1.56	3.17	3.23
1000	2	0.16	0.01	11.96	0.43	0.00	1.60	3.74	3.87
Avg. Type	2	0.29	0.05	7.53	0.77	0.03	1.36	2.97	3.51

Table 6
Average results over the random 12-player instances for sampling types 3 and 4, each sample size, and $k_{\max} = 20$.

Sampling		Heuristic						Exact	
Size	Type	RD_a	RD_e	C %	$\max_a$	$\min_a$	$Time_h$	$Time_e$	Iter
100	3	0.71	0.19	1.63	1.74	0.13	1.22	2.00	1.77
200	3	0.58	0.14	1.84	1.50	0.09	1.20	2.54	3.96
300	3	0.50	0.11	3.31	1.36	0.07	1.23	2.88	4.24
400	3	0.45	0.09	4.82	1.24	0.08	1.26	2.55	3.41
500	3	0.35	0.06	8.59	0.99	0.04	1.30	5.02	6.42
600	3	0.30	0.05	6.43	0.83	0.04	1.33	2.80	3.42
700	3	0.27	0.04	7.85	0.73	0.03	1.35	2.96	3.55
800	3	0.27	0.04	8.54	0.76	0.03	1.40	3.01	3.50
900	3	0.25	0.04	10.44	0.66	0.03	1.42	3.18	3.53
1000	3	0.22	0.03	10.71	0.61	0.02	1.50	3.06	3.19
Avg. Type	3	0.39	0.08	6.42	1.04	0.05	1.32	3.00	3.70
100	4	0.76	0.21	1.08	1.79	0.16	1.38	2.20	2.11
200	4	0.66	0.17	2.43	1.61	0.11	1.39	3.11	4.65
300	4	0.60	0.14	2.56	1.48	0.11	1.43	3.10	3.72
400	4	0.61	0.14	5.05	1.51	0.10	1.47	2.38	3.07
500	4	0.58	0.13	6.33	1.48	0.10	1.51	2.26	2.72
600	4	0.55	0.11	7.53	1.41	0.10	1.55	2.39	2.75
700	4	0.54	0.12	9.21	1.43	0.09	1.67	2.57	2.85
800	4	0.50	0.10	11.16	1.33	0.08	1.71	2.81	3.15
900	4	0.48	0.10	12.34	1.31	0.09	1.70	2.67	2.74
1000	4	0.45	0.09	12.89	1.25	0.07	1.79	2.85	2.88
Avg. Type	4	0.57	0.13	7.06	1.46	0.10	1.56	2.63	3.06

the heuristic, we compared such allocations with the true nucleolus in two different ways: The first one is a measure of the relative deviation (RD) between the two allocations, the second is a measure of the RD between the excess vectors lexicographically sorted. In other words, if $\tilde{x}$ and x are the nucleolus and a given allocation, and $e(x)$ is the vector of the $k_{\max}$ largest excesses produced by the allocation x, two measures we use to assess the quality of the allocations returned by the heuristics are:

- $RD_a = \frac{\sqrt{\sum_{j\in N}(\tilde{x}_j - x_j)^2}}{\sqrt{\sum_{j\in N}\tilde{x}_j^2}}$. This measure values how far allocation x is from the nucleolus $\tilde{x}$, in terms of Euclidean distance.
- $RD_e = \frac{\sqrt{\sum_{k=1}^{k_{\max}}(e(\tilde{x})_k - e(x)_k)^2}}{\sqrt{\sum_{k=1}^{k_{\max}} e(\tilde{x})_k^2}}$. This measure values how far the vector x is from the nucleolus $\tilde{x}$, in terms of their excess vectors lexicographically sorted.

The metrics RD_a and RD_e should not be interpreted as percentages. Actually, they measure the distance between x and $\tilde{x}$ (RD_a) and between the excess vector of the nucleolus $e(\tilde{x})$ and the excess vector of the given allocation $e(x)$(RD_e). They are normalized in such a way that, if they take value one, then x (or $e(x)$) is as far from $\tilde{x}$ ($e(\tilde{x})$) as the norm of $\tilde{x}$ ($e(\tilde{x})$). One could see these two metrics as the GAP of mathematical programs, which can take any non-negative value.

Table 7
Average results over 100 12-player balanced instances for sampling types 1 and 2, and each sample size.

Sampling		Heuristic				
Size	Type	RD_a	RD_e	C %	R %	$Time_h$
100	1	0.31	13.21	38.33	5.96	1.38
200	1	0.14	6.12	32.17	3.32	1.40
300	1	0.07	3.13	20.39	1.83	1.44
400	1	0.02	0.87	9.38	0.54	1.47
500	1	0.01	0.26	2.44	0.12	1.50
600	1	0.00	0.16	1.62	0.08	1.54
700	1	0.00	0.00	0.91	0.00	1.60
800	1	0.00	0.00	0.57	0.00	1.62
900	1	0.00	0.00	0.80	0.00	1.68
1000	1	0.00	0.00	0.58	0.00	1.72
Avg. Type	1	0.06	2.38	10.72	1.18	1.53
100	2	0.21	8.93	29.62	4.18	1.41
200	2	0.06	2.51	13.23	1.30	1.42
300	2	0.02	0.72	3.76	0.32	1.45
400	2	0.00	0.10	2.81	0.04	1.48
500	2	0.00	0.12	2.76	0.05	1.54
600	2	0.00	0.04	0.14	0.01	1.55
700	2	0.00	0.00	0.07	0.00	1.63
800	2	0.00	0.00	0.18	0.00	1.65
900	2	0.00	0.00	0.42	0.00	1.70
1000	2	0.00	0.00	1.81	0.00	1.80
Avg. Type	2	0.03	1.24	5.48	0.59	1.56

Table 8
Average results over 100 12-player balanced instances for sampling types 3 and 4, and each sample size.

Sampling		Heuristic				
Size	Type	RD_a	RD_e	C %	R %	$Time_h$
100	3	0.28	11.81	28.69	5.10	1.35
200	3	0.12	5.05	16.71	2.50	1.34
300	3	0.05	2.24	10.32	1.16	1.37
400	3	0.02	0.86	3.94	0.43	1.39
500	3	0.00	0.12	1.04	0.05	1.41
600	3	0.00	0.04	0.44	0.03	1.44
700	3	0.00	0.00	0.03	0.00	1.48
800	3	0.00	0.00	0.29	0.00	1.52
900	3	0.00	0.00	0.65	0.00	1.55
1000	3	0.00	0.00	2.09	0.00	1.62
Avg. Type	3	0.05	2.01	6.42	0.93	1.45
100	4	0.31	12.99	32.69	5.58	1.48
200	4	0.14	5.97	23.87	3.01	1.49
300	4	0.06	2.35	11.14	1.33	1.52
400	4	0.02	0.85	5.63	0.49	1.55
500	4	0.01	0.33	4.26	0.18	1.59
600	4	0.00	0.10	0.67	0.05	1.61
700	4	0.00	0.03	0.34	0.02	1.67
800	4	0.00	0.00	0.09	0.00	1.69
900	4	0.00	0.00	0.51	0.00	1.73
1000	4	0.00	0.00	0.86	0.00	1.81
Avg. Type	4	0.05	2.26	8.01	1.07	1.61

Table 9
Results over the 18-player instance for sampling types 1 and 2, each sample size, $k_{\max} = 30$.

Sampling		Heuristic						
Size	Type	RD_a	RD_e	RD_e^*	C %	$\max_a$	$\min_a$	$Time_h$
500	1	0.31	10.74	0.36	33.27	0.04	0.00	4.40
1000	1	0.20	4.79	0.16	23.09	0.03	0.00	8.24
1500	1	0.24	4.76	0.16	22.73	0.02	0.00	12.16
2000	1	0.20	4.18	0.14	31.75	0.03	0.00	16.63
2500	1	0.17	2.71	0.09	21.60	0.03	0.00	19.23
3000	1	0.16	3.13	0.10	29.78	0.02	0.00	24.28
3500	1	0.23	4.06	0.14	20.00	0.03	0.00	30.38
4000	1	0.15	2.65	0.09	19.02	0.02	0.00	32.26
4500	1	0.20	3.50	0.12	20.60	0.02	0.00	37.06
5000	1	0.18	3.15	0.11	16.97	0.03	0.00	44.04
Avg. Type	1	0.20	4.37	0.15	23.88	0.03	0.00	22.87
500	2	0.12	2.32	0.08	23.24	0.02	0.00	2.71
1000	2	0.22	5.84	0.19	32.86	0.02	0.00	4.70
1500	2	0.14	3.63	0.12	22.59	0.02	0.00	6.83
2000	2	0.13	2.05	0.07	31.52	0.02	0.00	9.33
2500	2	0.13	2.34	0.08	30.57	0.02	0.00	10.83
3000	2	0.15	3.44	0.11	20.62	0.03	0.00	12.72
3500	2	0.10	2.23	0.07	28.22	0.02	0.00	25.06
4000	2	0.08	1.08	0.04	26.87	0.01	0.00	18.58
4500	2	0.12	2.14	0.07	22.88	0.02	0.00	30.27
5000	2	0.08	1.57	0.05	23.89	0.01	0.00	23.65
Avg. Type	2	0.13	2.66	0.09	26.33	0.02	0.00	14.47

Table 10
Results over the 18-player instance for sampling types 3 and 4, each sample size, $k_{\max} = 30$.

Sampling		Heuristic						
Size	Type	RD_a	RD_e	RD_e^*	C %	$\max_a$	$\min_a$	$Time_h$
500	3	0.25	6.88	0.23	23.20	0.03	0.00	2.70
1000	3	0.15	2.44	0.08	22.92	0.02	0.00	5.87
1500	3	0.09	2.36	0.08	22.49	0.02	0.00	6.75
2000	3	0.12	1.38	0.05	31.33	0.01	0.00	9.54
2500	3	0.15	4.78	0.16	21.24	0.02	0.00	10.52
3000	3	0.08	2.28	0.08	29.21	0.01	0.00	12.77
3500	3	0.09	2.31	0.08	27.95	0.01	0.00	17.30
4000	3	0.10	1.73	0.06	21.25	0.02	0.00	17.92
4500	3	0.11	2.80	0.09	22.60	0.01	0.00	23.43
5000	3	0.11	2.06	0.07	16.51	0.02	0.00	23.06
Avg. Type	3	0.13	2.90	0.10	23.87	0.02	0.00	12.99
500	4	0.44	14.79	0.49	33.11	0.06	0.00	5.31
1000	4	0.28	7.53	0.25	22.85	0.03	0.00	8.82
1500	4	0.24	5.61	0.19	31.99	0.03	0.00	12.81
2000	4	0.23	5.13	0.17	21.78	0.02	0.00	16.88
2500	4	0.24	4.74	0.16	21.07	0.04	0.00	20.42
3000	4	0.21	4.07	0.14	28.95	0.02	0.00	26.30
3500	4	0.18	2.89	0.10	27.65	0.02	0.00	30.48
4000	4	0.19	3.48	0.12	18.34	0.02	0.00	33.57
4500	4	0.22	4.05	0.14	17.30	0.03	0.00	40.45
5000	4	0.17	3.13	0.10	16.21	0.02	0.00	42.49
Avg. Type	4	0.24	5.54	0.19	23.93	0.03	0.00	23.75

Besides, we also checked how many C_{ik} constraints are violated by the allocation given by the heuristic, and the absolute componentwise deviations between the true nucleolus and the allocations given by our heuristic.

The average results obtained by our heuristic approach, over the 100 random 12-player instances, for each sample size and each type of sampling, applying $k_{\max} = 20$, are summarized in Tables 5 and 6. Besides the columns already defined for the previous table, columns "RD_a", "RD_e" and "$Time_h$" show the average values of the two relative deviations computed, as well as the average time needed (in seconds) by the heuristic. Column "C %" shows the average percentage of constraints C_{ik} violated, whereas columns "max_a" and "min_a" show the maximum and minimum deviation between the heuristic allocation and the nucleolus, respectively.

In Tables 5 and 6 we obviously observe how the quality of the solutions found increases with the sample size, for each type of sampling tested, as both RD_a and RD_e decrease with the sample size. We also observe how the computational effort to find such allocations increases smoothly.

Our algorithms aim at finding allocations that are close to the "concept of nucleolus" (lexicographical minimization of the excess vector) and not close to the "nucleolus as an allocation". Since RD_a measures the distance between allocations, and RD_e measures the distance between excess vectors, it is logical that RD_a does not show as good results as RD_e does.

Table 11
Average CPU-times in seconds over the 100-player instances, for each sample size tested.

Sample size	CPU time
1000	2.16
2000	4.44
3000	8.80
4000	13.54
5000	26.26
6000	27.24
7000	36.03
8000	52.31
9000	57.52
10,000	69.27

4.4. Experiments over 12-player balanced instances

Another metric that could assess the quality of the allocation returned by the heuristic is, for balanced games, the proportion of rationality constraints violated. The average results of these experiments are shown in Table 7 and 8, where column "R %" shows the average number of rationality constraints violated by the allocation returned.

A first conclusion after these results is that the nucleolus is found if the sample size is greater than 700 or 800, depending on the sampling type. This minimum sample size required is between 700 and 800, depending on the sampling type. These results are extremely good, and suggest that, whenever the core is non-empty, our heuristic procedure finds the nucleolus quite easily.

4.5. Experiments over medium instances

In this section we analyze our heuristic procedure over the 18-player instance as presented and solved in Puerto and Perea (2013), for which we know the true nucleolus. Tables 9 and 10 show the results of our heuristic procedure, using the four sampling strategies suggested, and ten different sample sizes ranging from 500 to 5000. Parameter $k_{\max}$ is set to 30, as in Puerto and Perea (2013).

In this table we have added a new column "RD_e^*", to correct the fact that since the norm of the excess vector yielded by the nucleolus is too small (really close to zero), the numbers given in column "RD_e" are somehow affected by this small denominator. Therefore, $RD_e^* = RD_e^*(\sqrt{\sum_{k=1}^{k_{\max}} e(\tilde{x})_k^2})$. In this game, $\sqrt{\sum_{k=1}^{k_{\max}} e(\tilde{x})_k^2} = 0.03338436$.

The results confirm that our heuristic procedures can find allocations close to the nucleolus in a reasonable amount of time also for this larger game. The CPU time needed increases linearly with the sample size. The best results in terms of relative deviations are obtained with sampling type 2, in which only small coalitions are sampled.

4.6. Experiments over 100-player instances

These instances are randomly generated in such a way that only the data for the sampled coalitions (applying Type 1) is stored. The

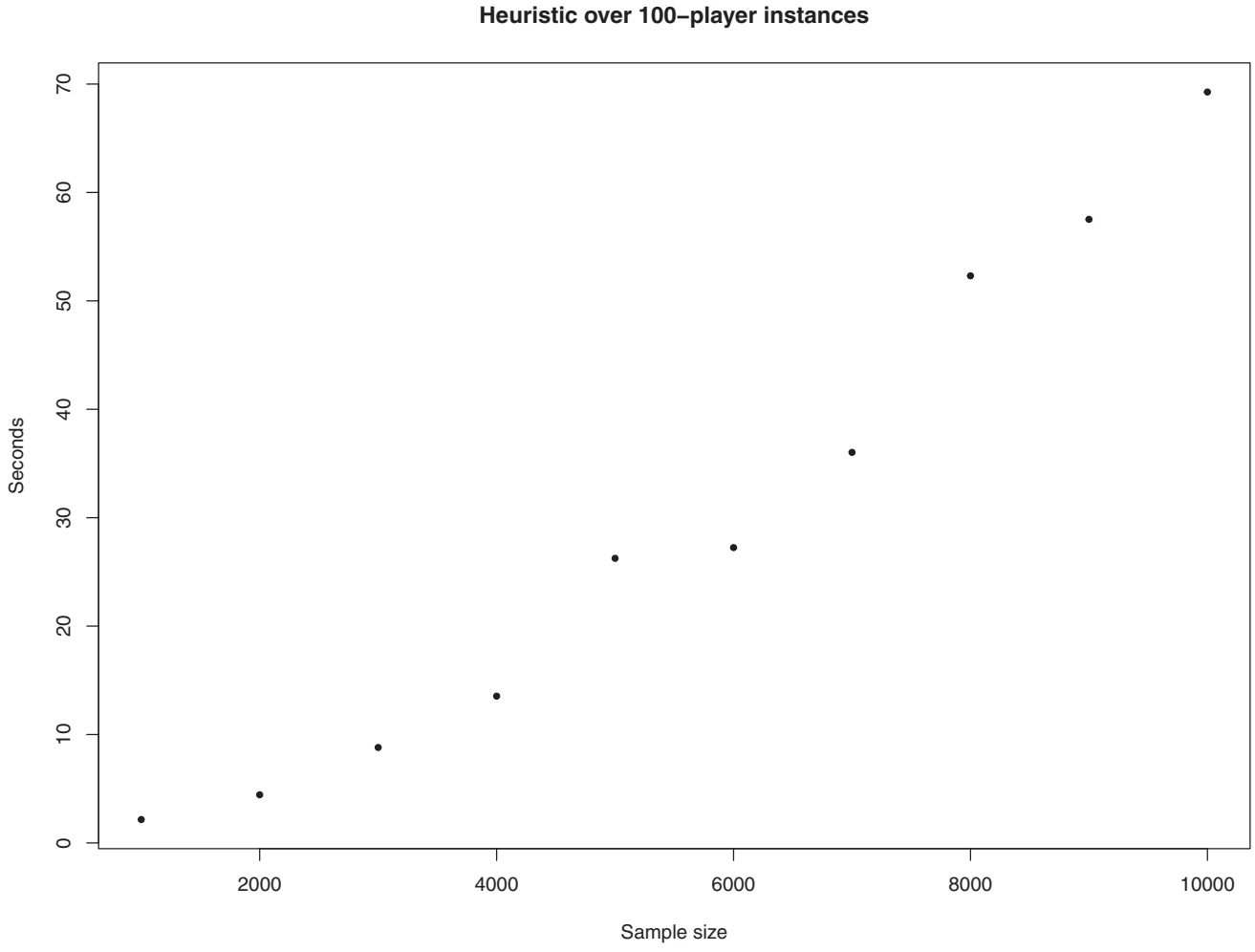

Fig. 1. Evolution of average CPU time used as a function of sample size, for each type of sampling, in the heuristic approach for the 100-player instances.

reader may note that storing the characteristic function and the coalition membership for $2^{100} = 1.267651e+30$ coalitions would be a real challenge, and therefore applying the exact approach seems impossible. The only purpose of this section is to show how one can obtain an allocation based on the proposed methodology for large games. Table 11 shows the average CPU time needed for our heuristic procedure to find an allocation (in seconds) for the different values of sample size tested. We observe in Fig. 1 how the increase in running times with respect to the size of the sample taken seems linear. In order to test the latter claim, we built a simple linear regression model to explain the average CPU time of the heuristic as a function of the sample size, which yielded significant parameters and a coefficient of determination of 0.9639 (96.39% of the variability in CPU time is explained by the sample size). Such large coefficient of determination supports that the CPU time increases only linearly with the sample size.

5. Conclusions

In this paper, we have introduced a row/column generation approach combined with sampling in order to find the nucleolus of any TU game. However, since the CPU time of this algorithm is only acceptable for relatively small instances, we have also proposed a heuristic approach for finding the nucleolus. Although both the literature on the nucleolus and the literature on heuristic and metaheuristic algorithms are vast, the combination of both disciplines is rather limited and does not include any serious analysis. We believe that, in order to compute the nucleolus for large games (which is more and more common in the current competing world) algorithms that are fast will be needed, even if the allocation returned is not guaranteed to be the true nucleolus. Therefore, we consider this piece of research as a first avenue for the interaction between heuristics and the nucleolus, which will surely gain more and more attention from the scientific community in the near future.

The heuristics proposed consist of sampling the set of coalitions, and solving a LP-model previously introduced in the literature for the nucleolus, considering only the coalitions chosen. The results obtained with this approach are quite satisfactory, as the allocations returned are close to the true nucleolus, as we tested over 12-player instances. Besides, our heuristics are capable of obtaining an allocation, which is also expected to be close to the nucleolus, in a reasonable amount of time for large games (we tested 100-player games). Specially good results are obtained when the corresponding game has non-empty core. In such balanced games with 12 players, the true nucleolus was always found by our heuristic procedure using relatively small sample sizes.

Further research on the nucleolus will necessarily focus on the search for fast algorithms for finding this allocation, or allocations close to it (as is the case of this paper).

Both quality measures used to assess the quality of the solutions returned by the heuristic are computed with respect to the origin. A different approach, in which the denominator considers some problem-specific point (e.g. the "centre" of the imputation set) is worth being explored. As mentioned in the experiments section, a disadvantage of using the distance relative to the origin is that the denominator is very close to zero. In fact, there are games with arbitrarily small or large denominator, which makes the measures considering the origin less meaningful. Therefore, further research will also focus on the search for new measures for assessing the quality of the solutions returned.

Acknowledgments

The authors would like to acknowledge the support from Spanish "Ministerio de Economia y competitividad" throughout grant number MTM2016-74983 and grant "SCHEYARD – Optimization of Scheduling Problems in Container Yards" (No. DPI2015-65895-R) financed by FEDER funds. Special thanks are due to two anonymous referees for their valuable comments.

References

Aiche, A., Rubinchik, A., Shitovitz, B., 2015. The asymptotic core, nucleolus and Shapley value of smooth market games with symmetric large players. Int. J. Game Theory 44 (1), 135–151.
Baïou, M., Barahona, F., 2017. On the nucleolus of shortest path games. Lect. Notes Comput. Sci. 10504, 55–66.
Brânzei, R., Iñarra, E., Tijs, S., Zarzuelo, J., 2006. A simple algorithm for the nucleolus of airport profit games. Int. J. Game Theory 34 (2), 259–272.
van den Brink, R., Katsev, I., van der Laan, G., 2011. A polynomial time algorithm for computing the nucleolus for a class of disjunctive games with a permission structure. Int. J. Game Theory 40 (3), 591–616.
Chin, H.H., 1997. Game Theoretical Applications to Economics and Operations Research. Genetic Algorithm for Finding the Nucleolus of Assignment Games. Springer, Boston, MA.
Deng, X., Fang, Q., Sun, X., 2009. Finding nucleolus of flow game. J. Comb. Optim. 18 (1), 64–86.
Dragan, I., 1981. A procedure for finding the nucleolus of a cooperative n person game. Zeitschrift für Oper. Res. 25, 119–131.
Fang, Q., Li, B., Shan, X., Sun, X., 2015. The least-core and nucleolus of path cooperative games. Lect. Notes Comput. Sci. 9198, 70–82.
Fang, Q., Li, B., Sun, X., Zhang, J., Zhang, J., 2016. Computing the least-core and nucleolus for threshold cardinality matching games. Theor. Comput. Sci. 609, 500–510.
Flisberg, P., Frisk, M., Rönnqvist, M., Guajardo, M., 2015. Potential savings and cost allocations for forest fuel transportation in Sweden: a country-wide study. Energy 85, 353–365.
Granot, D., Granot, F., Zhu, W.R., 1998. Characterization sets for the nucleolus. Int. J. Game Theory 27 (3), 359–374.
Greco, G., Malizia, E., Palopoli, L., Scarcello, F., 2014. The complexity of the nucleolus in compact games. ACM Trans. Comput. Theory 7 (1), 52pages.
Guajardo, M., Jörnsten, K., 2015. Common mistakes in computing the nucleolus. Eur. J. Oper. Res. 241, 931–935.
Hallefjord, A., Helming, R., Jörnsten., K., 1995. Computing the nucleolus when the characteristic function is given implicitly: a constraint generation approach. Int. J. Game Theory 24, 357–372.
Hamers, H., Klijn, F., Solymosi, T., Tijs, S., Vermeulen, D., 2003. On the nucleolus of neighbor games. Eur. J. Oper. Res. 146 (1), 1–18.
Hou, D., Driessen, T., 2015. Determining the nucleolus of compromise stable games. Bull. Aust. Math. Soc. 92 (3), 488–495. doi:10.1017/S0004972715001045.
Kamiyama, N., 2015. The nucleolus of arborescence games in directed acyclic graphs. Oper. Res. Lett. 43 (1), 89–92.
Kimms, A., Çetiner, D., 2012. Approximate nucleolus-based revenue sharing in airline alliances. Eur. J. Oper. Res. 220, 510–521.
Kohlberg, E., 1972. The nucleolus as a solution of a minimization problem. SIAM JAM 23, 34–39.
Kurz, S., Napel, S., Nohnc, A., 2014. The nucleolus of large majority games. Econ. Lett. 123, 139–143.
Martínez-De-Albéniz, F., Rafels, C., Ybern, N., 2013. A procedure to compute the nucleolus of the assignment game. Oper. Res. Lett. 41 (6), 675–678.
Maschler, M., Peleg, B., Shapley, L., 1979. Geometric properties of the kernel, nucleolus and related solution concepts. Math. Oper. Res. 4 (4), 303–338.
Maschler, M., Potters, J., Reijnierse, H., 2010. The nucleolus of a standard tree game revisited: a study of its monotonicity and computational properties. Int. J. Game Theory 39 (1), 89–104.
Nguyen, T.-D., Thomas, L., 2016. Finding the nucleoli of large cooperative games. Eur. J. Oper. Res. 248, 1078–1092.
Owen, G., 1974. A note on the nucleolus. Int. J. Game Theory 3, 101–103.
Owen, G., 1995. Game Theory. Academic Press, Inc. London.
Potters, J.A.M., Reijnierse, J., Ansing, M., 1996. Computing the nucleolus by solving a prolonged simplex algorithm. Math. Oper. Res. 21, 757–768.
Puerto, J., Perea, F., 2013. Finding the nucleolus of any n-person cooperative game by a single linear program. Computers and Operations Research 40, 2308–2313.
Reijnierse, H., Potters, J., 1998. The B-nucleolus of TU-games. Games Econ. Behav. 24 (1–2), 77–96.
Sankaran, J., 1991. On finding the nucleolus of an n-person cooperative game. Int. J. Game Theory 19, 329–338.
Solymosi, T., 1993. On Computing the Nucleolus of Cooperative Games. University of Illinois at Chicago Ph.D. thesis.
Solymosi, T., Raghavan, T., Tijs, S., 2005. Computing the nucleolus of cyclic permutation games. Eur. J. Oper. Res. 162 (1), 270–280.
Sziklai, B., Fleiner, T., Solymosi, T., 2017. On the core and nucleolus of directed acyclic graph games. Math. Program. Ser. A 163, 243–271.
Wang, Y., Yin, Z., Li, Y., 2017. The application of data-process interaction model in cost allocation. Acad. J. Manuf. Eng. 15 (3), 129–138.
Yu, Y., Lou, Q., Tang, J., Wang, J., Yue, X., 2017. An exact decomposition method to save trips in cooperative pickup and delivery based on scheduled trips and profit distribution. Comput. Oper. Res. 87, 245–257.