THE NAME OF THE GAME

1988 Summer REU Project

BY

Donna Brown
Wells College

Lori Jew
Worcester Polytechnic Institute

Diane Macdonald
Wells College

We would like to give many thanks and much gratitude to Professor David Housman of Worcester Polytechnic Institute, the "driving force" behind this project. Without whose seemingly unbounded energy and dedication to the pursuit of perfection, the value of our three person coalition would have been zero.

D. B.
L. J.
D. M.

July 1988

Introduction

An n-person cooperative game is a pair (N, v) where N = {1, 2, ..., n} is a set of n players 1, 2, ..., n and where v is a real-valued characteristic function on 2^N, the set of all subsets of N. Let S, a subset of N, be a coalition of players, and let v(S) assign a value to the coalition S when the members of S work together. Define $v(\emptyset) = 0$. The game (N, v) is called a value game. A cost game is defined as c(S) = - v(S).

A game is superadditive if for all coalitions S and T where $S \cap T = \emptyset$, $v(S \cup T) \geq v(S) + v(T)$.

A game (N, v) is additive if it can be decomposed into two games (N_1, v_1) and (N_2, v_2) such that, for all coalitions $S_1 \subset N_1$ and $S_2 \subset N_2$, $v(S_1 \cup S_2) = v_1(S_1) + v_2(S_2)$.

A game is said to be monotonic if $v(S) \geq v(T)$ whenever T is contained in S.

A vector $x = (x_1, x_2, ..., x_n)$ with real components is an imputation for the game if $x_i \geq v(i)$ for all i contained in N (individual rationality), and

$$\sum_{i=1}^{n} x_i = v(N) \ (efficiency).$$

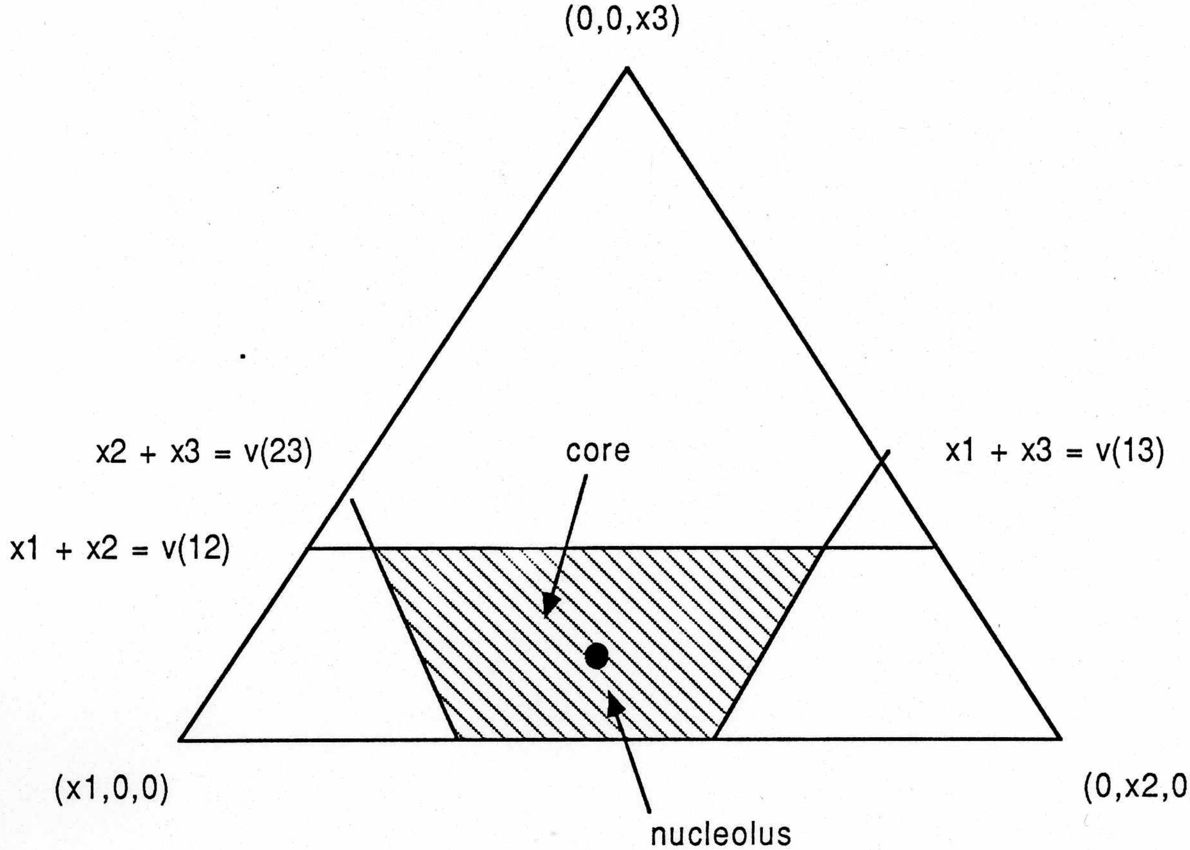
(See figure 1.)

An imputation x is coalitionally rational if, for all $S \subset N$,

$$\sum_{i \in S} x_i \geq v(S).$$

The core of a game is defined as the set of all imputations x such

Figure 1: The Imputation Space



that $\sum_{i \in S} x_i \geq v(S)$ *for all* $S \subset N$, *and* $\sum_{i \in N} x_i$. (*See figure* 1.)

An allocation method is any procedure which, given a game, assigns or allocates the amount x_i to player i for all players. The allocation is the vector $x = (x_1, x_2, ..., x_n)$.

There are several fair allocation methods which we considered in our research. They are as follows:

(1) Shapley value: The Shapley value measures each player's marginal

contribution to the grand coalition, and averages these values over all possible permutations of the players. Formally, the Shapley value for player i is defined as

$$\phi_i = \sum_{S \ni i} \frac{(s-1)!(n-s)!}{n!}[v(S) - v(S - \{i\})].$$

(2) Nucleolus: We first define the excess as

$$e(x,S) = v(S) - \sum_{i \in S} x_i.$$

This value represents the "size of the complaint" the coalition S would have against the allocation x. Define the excess vector as

$e(x) = [e(x,S_1), e(x,S_2), \ldots, e(x,S_{2^\wedge(n-1)})]$, where $e(x,S_i) > e(x,S_{i+1})$.

To define a lexicographic ordering on x and y, we say $x < y$ if there exists an i such that for $j = 1, \ldots, i$, $x_j = y_j$, and $x_{i+1} < y_{i+1}$. The nucleolus is the value which minimizes e(x) lexicographically. It is in one sense the "middle" of the core of a game, if the core is nonempty. (See figure 1.) The nucleolus can be found with a series of linear programs.

(3) The Tau value: To find the τ-value we first find the marginal vector $M_i(v) = v(N) - v(N\backslash\{i\})$, which is the amount player i will contribute by joining to form the grand coalition. This is considered as the most that player i can hope to receive if working cooperatively with the other players. The remainder for player i is calculated:

$$R_v(S, i) = v(S) - \sum_{j \in S-\{i\}} M_j(v).$$

From this a lower bound for an allocated amount for player i is found. The minimal right for player i is $m_i(v) = \min R_v(S, i)$. The τ-value for player i is the unique efficient vector lying on the line determined by m(v) and M(v).

stop

Kohlberg's Theorem

In order to state Kohlberg's theorem, we will need a few definitions.

Balanced collections: Let $\beta = \{S_1, S_2, ..., S_m)$ be a collection of subsets of

$N = \{1, ..., n\}$. β is N-balanced if we can find a balancing vector $y = (y_1, ..., y_m)$ such that, for every player i,

$$\sum_{j:\, i \in S_j} y_j = 1,$$

and all $y_j > 0$.

Given an imputation x, let β_k be the set of all coalitions with kth maximal excess, such that the excess of β_i is greater than the excess of β_{i+1}.

Define the array determined by x,

$$C_k = \bigcup_{i=1}^{k} \beta_i .$$

Theorem: The imputation x is the nucleolus of a game (N, v) if and only if the array $C_1, ..., C_q$ determined by x consists of only balanced collections.

Part I

PM
TPM
AM

Strengthening of Kohlberg's Theorem

Suppose (N,v) is a superadditive game. For a given imputation x, we define a sequence of collections $\beta_1 \ldots \beta_q$ such that β_k is the collection of coalitions with k^{th} maximal excess. Let

$$C_k = \bigcup_{j=1}^{k} \beta_j .$$

By the definition of β_i, we know that for a given i = 1...q, e(x,S) = e(x,T) for all coalitions S and T in β_i. From this we can derive no more than $|\beta|-1$ independent equations in x. Note that these equations need not all be independent. Let d_i be the number of independent equations determined by C_i, along with the efficiency equation

$$\sum_{i=1}^{n} x_i = v(N),$$

We define k^* as the minimal integer such that $d_{k^*} = n$. We have n independent equations and n unknowns, and we can uniquely determine x.

Theorem: The imputation x is the nucleolus if and only if the collections $C_1 \ldots C_{k^*}$ of coalitions determined by x are balanced and the excess equalities determined by C_{k^*} and efficiency are solved uniquely by x.

Definition: To prepare for the proof, we define the characteristic vector on S by representing $S \subset N$ as a vector $1_s = (s_1,...s_n)$ where

$s_i = 1$ if $i \in S$, and $s_i = 0$ if $i \notin S$.

Proof: (Modification of Kohlberg, 1971) Suppose $C_1...C_q$ are the collections of coalitions determined by x, and suppose $C_1...C_{k^*}$ are balanced. By Kohlberg's theorem, we can show that x is the nucleolus by showing that $C_{k^*+1},...C_q$ are balanced. We will show this by showing that any collection $C \supset C_{k^*}$ is balanced. This will be true if $C_{k^*} \cup \{T\}$ is balanced for all $T \notin C_{k^*}$ (because the union of balanced collections is balanced). Since C_{k^*} is balanced, there exists $y_S > 0$ for each $S \in C_{k^*}$ such that

$$\sum_{S \in C_{k^*}} y_S 1_S = 1_N.$$

Since C^{k^*} and efficiency uniquely determine x, there exists $s_1...s_n \in C_{k^*} \cup \{\emptyset,N\}$ such that $\{1_{s1}, -1_{s2},..., 1_{s2n-1}, -1_{s2n}\}$ are linearly independent and span $\Re_n$. Then there exists z_s for each $s \in C_{k^*}$ such that

$$1_T = \sum_{s \in C_{k^*}} z_S 1_S$$

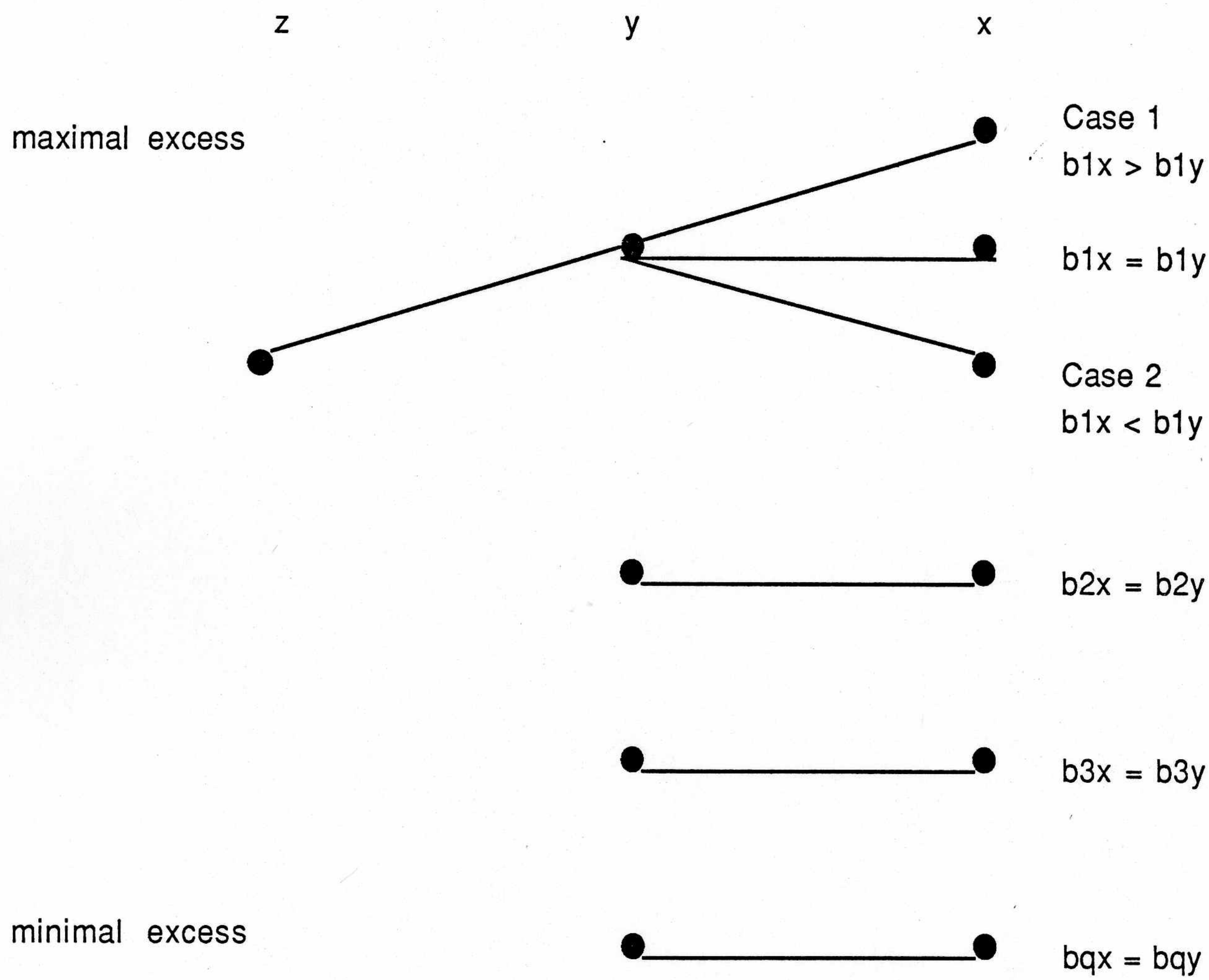
Adding these two equations and multiplying the second by ε gives us

$$\sum_{S \in C_{k^*}} (y_S - \varepsilon z_S)\, 1_S + \varepsilon 1_T = 1_N.$$

Since $y_S > 0$ for all $S \in C_{k^*}$, we can choose $\varepsilon > 0$ small enough so that $y_S - \varepsilon z_S$

> 0 for all $S \in \mathcal{C}_{k^*}$. Hence, $\mathcal{C}_{k^*} \cup \{T\}$ is balanced.

Figure 2: Graphic View of Proof of Theorem 1



Conversely, suppose x is the nucleolus. By Kohlberg's theorem, $\mathcal{C}_1 \ldots \mathcal{C}_k$ are balanced. So, the conclusion of the present theorem would be false only if there exists an allocation $y \neq x$ with vector c. Let

$b_{ix} = e(x,S)$, for all coalitions S in β_i, and
$b_{iy} = e(y,S)$, for all coalitions S in β_i.

Consider b_{1x} and b_{1y}. We have two cases: (Refer to Figure 2)

Case 1: $b_{1x} < b_{1y}$. Then $e(x,S) < e(y,S)$. Let $z = \lambda y + (1-\lambda) x$ where $\lambda < 0$.

Then for $S \in \beta_1$,

$$e(z,S) = \lambda e(y,S) + (1-\lambda)e(x,S)$$

$$= \lambda b_{1y} + (1-\lambda)b_{1x}$$

$$= \lambda b_{1y} - \lambda b_{1x} + b_{1x} < 0 + b_{1x} = e(x,S).$$

So, $e(z,S) < e(x,S)$ for all $S \in \beta_1$. Now since $e(x,R) < b_{1x}$ for all $R \notin \beta_1$, we can choose λ close enough to 0 so that $e(z,R) < e(z,S)$ for all $R \notin \beta_1$ and $S \in \beta_1$. This implies that $e(z) < e(x)$, which contradicts our assumption that x is the nucleolus.

Case 2: $b_{1x} > b_{1y}$. Then $e(y) < e(x)$, which contradicts our assumption that x is the nucleolus.

Then $b_{1x} = b_{1y}$.

Consider b_{2x} and b_{2y}. We have 2 cases:

Case 1: $b_{2x} < b_{2y}$. The same argument holds as for case 1 above.

Case 2: $b_{2x} > b_{2y}$. Then, since $b_{1x} = b_{1y}$, and $b_{2x} > b_{2y}$, $e(x) > e(y)$. This contradicts our assumption that x is the nucleolus.

Then $b_{2x} = b_{2y}$.

The same argument holds, by induction, for all b_{1x}, b_{1y} in $\beta_1 \ldots \beta_q$. (Refer to Figure 2.) We then have e(x) = e(y), and leads to a contradiction because of the uniqueness of the nucleolus. Then y is the nucleolus, and the array corresponding to x uniquely determines x.

Conjectures on Bounds for K*

It was hoped that a reasonable bound for k* could be established so as to further strengthen the theorem. Two approaches were tried, and counterexamples proved them incorrect. It was decided that further research into a bound for k* would not be useful, so the theorem stands as stated above. The two approaches and their counterexamples follow.

One idea was that each successive β_i adds only independent equations in x to the system. This seemed to hold true for the examples we had tried to this point. If this were true, then the following bound for k* follows:

$$n \leq 1 + (|\beta_1| - 1) + (|\beta_2| - 1) + \ldots + (|\beta_{k^*}| - 1)$$
$$\leq 1 + |C_{k^*}| - k^*$$

The counterexample is as follows:

Define a 3-person game:
$v_1(i) = 0$
$v_1(12) = 7$
$v_1(ij) = 0$ otherwise, and $v_1(N) = 9$.

nucleolus = (4,4,1).

Define the same game, call it v_2, on players 4, 5, and 6, then define a 6-person game by adding the two:

$v(S_1 \cup S_2) = v_1(S_1) + v_2(S_2)$.

By additivity, the nucleolus of the new game = (4,4,1,4,4,1). Now,
$\beta_1 = \{123,456\}$
$\beta_2 = \{12,12456,3,3456,45,12345,6,1236\}$

If the conjecture holds, then $k^* = 2$:

$n \leq 1 + |\mathcal{C}_{k^*}| - k^*$
$6 \leq 1 + 10 - 2$
$6 \leq 9$

These collections determine only 3 independent equations, and therefore do not uniquely determine x.

An unproven but intuitive idea was that for all β_i contained in $\mathcal{C}_{k^*}$, $|\beta_i| \geq 2$. This would imply that each successive β_i would yield at least 1 more equation in x. This is the "worst case" approach. It was conjectured that , because of the balancing restriction, each successive β_i must yield at least one additional equality, so $k^* \leq n-1$ must hold. The counterexample is as follows:
Define a 4-person game:

$v(i) = 0$
$v(12) = 1.4$
$v(13) = v(14) = v(23) = v(24) = 1.2$
$v(34) = 2$

v(123) = v(124) = 2.4
v(134) = v(234) = 3.1
v(N) = 5.4.

The nucleolus of this game is (1.2, 1.2, 1.5, 1.5). Note that the game is superadditive with no other special properties. If the conjecture is true, then $k^* = 3$:

β_1 = {12,34}
β_2 = {134,234}
β_3 = {1,2}

These collections yield 2 independent equations in x, for a total of 3 equations and 4 unknowns, so x is not uniquely determined.

Iterative Scheme For Calculating the Nucleolus

In 1967, R.E. Stearns produced a convergent transfer scheme for calculating the nucleolus of an arbitrary n-person game. We proposed a similar scheme based on the properties of consistency and covariance. Recall that Sobolev proved that if a method satisfies both consistency and covariance, then the method must be the nucleolus.

Define the quantity $S_{ij}(x)$ = max e(S,x) for all S containing i but not j. In other words, $S_{ij}(x)$ is the best that player i could do without the cooperation of player j. Call this quantity the surplus of i against j. Suppose that $S_{ij}(x) > S_{ji}(x)$. Then player i can make a demand on player j that player j cannot contest. This leads to a degree of instability in the allocation x. The iterative scheme for calculating the nucleolus considers this excess demand, $S_{ij}(x) - S_{ji}(x)$, and continues until $S_{ij}(x) - S_{ji}(x) = 0$. We then tested the algorithm to determine if it did, in fact, converge to the nucleolus, and if so, with what order of efficiency. Unfortunately, the algorithm converges to the nucleolus only in special cases, and may converge to other points depending upon the point at which the iterations begin.

The kernel of a game is defined as the set of all payoff vectors such that no player i can make an uncontestable demand on any player j. This can occur in one of three ways:

1. $S_{ij}(x) = S_{ji}(x)$,
2. $S_{ij}(x) > S_{ji}(x)$, but $x_j = v(j)$,
3. $S_{ji}(x) > S_{ij}(x)$, but $x_i = v(i)$.

Notice that the termination of the proposed algorithm will occur at any point in the kernel for which condition 1 is satisfied, and this is where problems arose. For all 3-person games, and for all 4-person constant sum games, the algorithm always converges to the nucleolus, because the kernel of these games consists of that single point. The algorithm may converge to different points in the kernel for games in which the kernel is not a single point. Further attempts to revise the iterative method would have been unproductive, as they most likely would have led to Stearns' algorithm of 1967.

Linear Programming Algorithm for Calculating the Nucleolus

Turning away from the iterative approach, and due to the need for a quick way to determine the nucleolus, we turned to the linear programming technique for calculating the nucleolus of a game. The model is based on Owen's article of 1982 in his book, Game Theory. The method used to solve the model is the revised simplex method presented by Chvàtal, 1980, with a slight modification to allow for free variables.

Suppose (N,v) is a game and w is a vector with n-1 positive components. We first define the w-nucleolus, a generalization of the nucleolus, as the value which lexicographically minimizes the maximal weighted excesses on the set of imputations:

$$e_N(x,S) = \frac{v(S) - \sum_{i \in S} x_i}{w_{|S|}}$$

The nucleolus as defined earlier is simply the weighted nucleolus when all $w_{|S|} = 1$. The sequence of linear program is as follows:

$N = \{S \;\; N: S \neq \emptyset\}$
$\beta_o = C_o = \{N\}$
$B_o = C_o = \emptyset$
$\alpha_o = 0$
$v_o(S) = v(S)$ for all $S \subset N$

For $k = 0,1,\ldots$ until a unique x has been determined/ $\mathcal{C}_{k+1} = N$.

$\alpha k+1 = \min \alpha$

subject to

$$w_{|S|}\alpha + \sum_{i\in S} x_i \geq v^k(S), \quad S \in N \setminus C_k$$

$$\sum_{i\in S} x_i = v^k(S), \quad S \in \mathbf{C}_k$$

$$x_i \geq v(i), \qquad i \in N \setminus \mathcal{C}_k$$

$$x_i = v(i), \qquad i \in \mathcal{C}_k.$$

$$\beta_{k+1} = \{S \in N \setminus C_k : w_{|S|}\alpha + \sum_{i\in S} x_i = v(S)$$

for all optimal solutions x for LP_k

$\mathcal{C}_{k+1} = \mathcal{C}_k \cup \beta_{k+1}$

$B_{k+1} = \{ i \in N \setminus \mathcal{C}_k : x_i = v(i)$ for all optimal solutions x of $LP_k \}$

$C_{k+1} = C_k \cup B_{k+1}$

$$v^{k+1}(S) = v^k(S) - w|S|\alpha_{k+1} \quad \text{if } S \in \beta_{k+1}$$
$$= v^k(S) \quad \text{otherwise.}$$

Note that β and B need not consist of all possible coalitions that satisfy the definition. All this means is that the procedure may take more than one LP to find all the members of β and B as defined above. This does not affect the final outcome, but may be of importance when further research is done on the efficiency of the algorithm.

In order to minimize the runtime needed to solve the LP, we instead consider the dual of the LP given above. It is defined as follows:

$$\max \sum_{S \in N} v^k(S) y_S + \sum_{i \in N} v(i) z_i$$

subject to

$$\sum_{S \in N \setminus C_k} w|S|\, y(S) = 1$$

$$\sum_{S : i \in S} y_S + z_i = 0, \quad i \in N$$

$$y_S \geq 0, \qquad S \in N \setminus C_k$$

$$z_i \geq 0, \qquad i \in N \setminus C_k.$$

$$\beta'_{k+1} = \{ S \in N \setminus C_k : y_S^* > 0 \} \subset \beta_{k+1},$$

$$B'_{k+1} = \{ i \in N \setminus C_k : z_i^* > 0 \} \subset B_{k+1}.$$

Using the strengthening of Kohlberg's theorem, it makes sense to terminate the procedure as soon as the nucleolus is uniquely determined, rather than continuing through all linear programs. Termination occurs after the k^{th} LP if the following system of linear equations possesses a unique solution:

$$\sum_{i \in S} x_i = v^{k+1}(S), \quad S \in C_{k+1}, \text{ and } x_i = v(i), \, i \in C_{k+1}.$$

Each time a linear program is solved, the constraints corresponding to the coalitions whose excesses must be maximal for all optimal solutions of the LP are changed to equalities in the next LP. These equalities are added, one at a time, to a matrix, as they are determined. The matrix is then reduced in order to determine if the new equation is independent of those already in the matrix. If it is not, it is not added to the matrix. Thus the matrix consists only of independent equations in x. When the matrix is of rank equal to the number of players, we are able to uniquely determine x. By the strengthening of Kohlberg's theorem, x must be the nucleolus, and the program terminates.

It must be noted that the method used to solve the linear programs is not the most efficient one, since many pivots are required to find the optimal basic feasible solution. The method will be revised as the efficiency of the program is improved.

Part II

Two-additive Games

Two-additive games are a class of games in which the value of all coalitions with more than two players depends on the values of the two-player coalitions. Given a set of players N = {1, 2, ..., n}, define the characteristic function as

$$v(S) = \begin{cases} 0 & , if\,|S|<2 \\ x_s & , if\,|S|=2 \\ \sum\limits_{R \subset S} x_R & , if\,|S|>2 \end{cases}$$

A way to visualize this is to draw a graph G = (N, E), where each player is represented by an element of the vertex set N, and the value of the coalition (ij) is represented by the weighted edge of E with endpoints i and j. On a coalition $S \subseteq N$ define

(1) an interior edge of S as an edge in E with both vertices in S;

(2) an exterior edge of S as an edge with no vertices in S; and

(3) a boundary edge of S as an edge with one vertex in S.

The value of a coalition S is defined as the sum of the weights of the interior edges of S.

Theorem: On two-additive games, the nucleolus, the Shapley value and the tau value achieve the same value: $\nu_i = \tau_i = \phi_i = 1/2$ * (the sum of the weights of the edges adjacent to i).

Proof for nucleolus: Given a game as described above, with player set

N = {1, 2, ..., n}, we will use Kohlberg's theorem to prove that the nucleolus for player i is 1/2 * (the sum of the weights of the edges adjacent to i). Let x be defined by x_i = 1/2 * (the sum of the weights of the edges adjacent to i). Now we find the excess e(x, S) for each $S \subset N$.

$$e(x, S) = v(S) - \sum_{i \in S} x_i$$

= (the sum of the weights of the interior edges of S) - 1/2 *

$(\sum_{i \in S}$ the sum of the weights of the edges adjacent to i)

= (the sum of the weights of the interior edges of S) -

1/2 * (2 * the sum of the weights of the interior edges of S + the sum of the weights of the boundary edges of S)

= -1/2 * (the sum of the weights of the boundary edges of S).

Since S^c is also a set in N, e(x, S^c) = -1/2 * (the sum of the boundary edges of S^c). Hence, for every $S \subset N$, e(x, S) = e(x, S^c) so for every $S \subset \beta_i$, $S^c \subset \beta_i$. Since each player is in the same number of coalitions in each β_i, each β_i is balanced so x is the nucleolus. (Οπερ Εδει Δειξαι)

Proof for Shapley value: Given a graph G = (N, E), N vertices, E edges. Consider an edge e with vertices i and j. When calculating the Shapley value the only player that will receive the weight of edge e is player i or player j, whichever is not the first to be added to the permutation. Players i and j appear after each other an equal number of times so they will split the value of edge e, each receiving half of the weight of edge e. So the Shapley value for player i is 1/2 * (the sum of the weights of the edges adjacent to vertex i). (Οπερ Εδει Δειξαι)

Proof for τ-value: To find the τ-value for player i, first we find the marginal vector M(v) corresponding to v with

$M_i(v) = v(N) - v(N - \{i\})$

= (the sum of the weights of all edges of the graph) - (the sum of the weights of the edges which are not adjacent to i)

= the sum of the weights of the edges adjacent to i.

Next the remainder for player i in the coalition S is calculated with

$$R_v(S,i) = v(S) - \sum_{j \in S} M_j(v)$$

= (the sum of the weights of the interior edges of S) - (the sum of the weights of the edges adjacent to j for each $j \in (S - \{i\})$).

Set $m_i(v) = \max R_v(S, i)$. This occurs when S is a singleton since for each set with more than one member, the sum of the weights of the edges adjacent to j for each $j \in (S - \{i\})$ is at least as large as the sum of the weights of the interior edges of S. So $m_i(v) = 0$ for all $i \in N$. Now let

$$\alpha_v = \frac{v(N) - \sum_{i=1}^{n} m_i(V)}{\sum_{i=1}^{n} M_i(v) - \sum_{i=1}^{n} m_i(v)}$$

= (the sum of the weights of the graph)/(2* the sum of the graph)

= 1/2.

Then $\tau_i(v) = m_i(v) + \alpha_v(M_i(v) - m_i(v))$

= 1/2 * (the sum of the weights of the edges adjacent to i).

(Οπερ Εδει Δειξαι)

Sufficient conditions for nucleolus to equal Shapley value

Consider a game (N, v) defined as follows:

(Example 1:)

$v(\varnothing) = 0$	$v(23) = 2$
$v(1) = 0$	$v(24) = 2$
$v(2) = 0$	$v(34) = 2$
$v(3) = 0$	$v(123) = 4$
$v(4) = 0$	$v(124) = 2$
$v(12) = 2$	$v(134) = 2$
$v(13) = 2$	$v(234) = 4$
$v(14) = 0$	$v(1234) = 6$.

Note that this game is not two-additive. The nucleolus and the Shapley value for this game are both (1, 2, 2, 1). If we look at the excess vector and consider each β_i, we see that for every $S \subset \beta_i$, S^c is also in β_i so each β_i is balanced. However, if we look at a game defined by

(Example 2:)

$v(\varnothing) = 0$	$v(23) = 2$
$v(1) = 0$	$v(24) = 5/2$
$v(2) = 0$	$v(34) = 5/2$
$v(3) = 0$	$v(123) = 4$
$v(4) = 0$	$v(124) = 5/2$

$v(12) = 3/2$ $\quad$ $v(134) = 5/2$

$v(13) = 3/2$ $\quad$ $v(234) = 4$

$v(14) = 1/2$ $\quad$ $v(1234) = 13/2.$

The nucleolus for this game is (1, 2, 2, 3/2). However the Shapley value is (13/12, 2, 2, 17/12). There is a slight difference in the β_i's for these two games. In the first game $\beta_1 = \{1,4,12,13,24,34,123,234\}$ and β_2 is all of the other coalitions. In the second game $\beta_1 = \{1,24,34,123\}$, $\beta_2 = \{4,12,13,234\}$, and β_3 is the other coalitions. In both games each β_i is balanced.

Since we noticed that in the first game, $S \subset \beta_i$ implied that $S^c \subset \beta_i$, and this did not occur in the second game, we wondered if this was a sufficient condition for $\phi = \nu$.

Theorem: Suppose ν is the nucleolus of game (N,v) and $\beta_1, ..., \beta_q$ is the array determined by ν. If for all $j \in \{1, ..., q\}$, $S \in \beta_j$ implies $S^c \in \beta_j$, then $\phi = \nu$.

Proof: Assume ν is the nucleolus for game (N, v) and for each $S \in \beta_j$, $S^c \in \beta_j$. This means

$$e(\nu, S) = e(\nu, S^c).$$

$$v(S) - \sum_{i \in S} \nu_i = v(S^c) - \sum_{j \in S^c} \nu_j$$

If $|N|$ is even, then by definition

$$\phi_i = \sum_{S \ni i} \frac{(s-1)!(n-s)!}{n!}[v(S) - v(S - \{i\})]$$

$$= \sum_{S \ni i, |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} \left[v(S) - v(S - \{i\}) + v((S - \{i\})^c) - v(S^c) \right]$$

$$= \sum_{S \ni i, \, |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} \left[\sum_{j \in S} v_j - \sum_{j \in (S - \{i\})} v_j + \sum_{j \in (S - \{i\})^c} v_j - \sum_{j \in S^c} v_j \right]$$

$$= 2v_i \sum_{S \ni i, \, |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!}$$

If |N| is odd, it can be shown similarly that

$$\phi_i = 2v_i \sum_{S \ni i, \, |S| < \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} + v_i \sum_{S \ni i, \, |S| = \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!}.$$

Case 1: (for |N| even)

For $\phi_i = v_i$, we need to prove that

$$\sum_{S \ni i, \, |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} = \frac{1}{2}.$$

We know that

$$\sum_{S \ni i, \, |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} = \sum_{s=1}^{\frac{n}{2}} \frac{(n-1)!}{(s-1)!(n-s)!} * \frac{(s-1)!(n-s)!}{n!}.$$

The part of the term "(n-1)!/[(s-1)!(n-s)!]" is the number of coalitions of size s which contain player i. So this equation reduces to 1/n for each s, of which there are n/2 yielding (1/n)(n/2) = 1/2.

$$\phi_i = 2v_i \sum_{S \ni i,\ |S| \leq \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} = (2v_i)(\tfrac{1}{2}) = v_i.$$

Case 2: (for |N| odd)

As in case 1, we know that

$$\sum_{S \ni i,\ |S| < \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} = \sum_{s=1}^{\frac{n}{2}} \frac{(n-1)!}{(s-1)!(n-s)!} * \frac{(s-1)!(n-s)!}{n!}.$$

For |S| = (n + 1)/2, we get

$$v_i \sum_{S \ni i,\ |S| = \frac{n}{2}} \frac{(s-1)!(n-s)!}{n!} = \frac{v_i}{2n}.$$

For all S with s < n/2, the term "(n-1)!/[(s-1)!(n-s)!]" is the number of coalitions of size s which contain player i. For s = (n + 1)/2 only half of the coalitions of size s are considered so the coefficient needed is (n-1)!/2*[(s-1)!(n-s)!]. Adding these we get:

$$\left[\frac{n-1}{2} * \frac{1}{n}\right] + \frac{1}{2n} = \frac{1}{2}.$$

So $\phi_i = (2v_i)(1/2) = v_i$. (Οπερ Εδει Δειξαι)

We had also conjectured that if $v = \phi$ then either the game is completely symmetric or for all $j \in \{1, ..., q\}$, if $S \in \beta_j$ then $S^c \in \beta_j$. However we were able to find a counterexample to show that this is not true. Consider the game defined as follows:

(Example 3:)

v(Ø) = 0	v(23) = 5
v(1) = 0	v(24) = 4

v(2) = 5/2	v(34) = 4
v(3) = 5/2	v(123) = 8
v(4) = 0	v(124) = 4
v(12) = 4	v(134) = 4
v(13) = 4	v(234) = 8
v(14) = 0	v(N) = 10

The nucleolus for this game is (1, 4, 4, 1), which is also the Shapley value. It is easy to see that this game is not completely symmetric, so we must just check each β_j until we find one which does not contain the complement of every set which it contains.

$\beta_1 = \{1,4,12,13,24,34,123,234\}$

$\beta_2 = \{2,3\}$

$\beta_3 = \{14,124,134\}$

$\beta_4 = \{23\}$

β_2, β_3, and β_4 all contain a set without containing that set's complement. Somehow the constraints on our conjecture must be broadened to contain more classes of games.

To summarize the results we have just presented, we give a Venn diagram depicting the classifications of the previous examples.

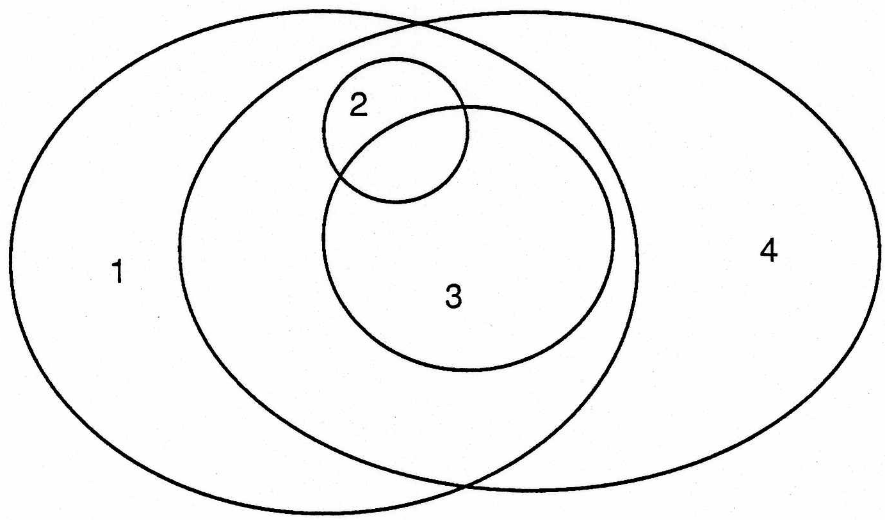


Figure 3.

1. The set of all classes of games in which each β_k is balanced.
2. The set of all games which are completely symmetric.
3. The set of all games in which $S \in \beta_j \Rightarrow S^c \in \beta_j$.
4. The set of all games in which $\phi = v$.

Example 1 lies in sets 1, 3, and 4; Example 2 is a game of class 1; and Example 3 is in set 4.

Part III

Methods

There are many different allocation methods each possessing various properties. Consider a cost game (N,c) where $i \in N$.

Let the separable cost of player i in game v be defined as,

$$s_i = c(N) - c(N - \{i\})$$

In other words, s_i is the marginal cost of player i. Also, define the remaining benefit to i as,

$$r_i = c(i) - s_i$$

The Separable Costs Remaining Benefits (SCRB) method is then given by,

$$x_i = s_i + \frac{r_i}{\sum_N r_j}\left[c(N) - \sum_N s_j\right]$$

where x_i is the amount (cost) being allocated to player i. This method distributes to each player their separable cost while the remaining benefits are given out proportionally.

Another method, which is based on the same idea as SCRB, is Equal Allocation of Nonseparable Costs (EANC). As one might guess from the name, each player pays their separable cost and is then given an even share of the nonseparable costs:

$$x_i = s_i + \frac{1}{n}\left[c(N) - \sum_N s_j\right].$$

Recall that the nucleolus is the vector, **x**, that lexicographically maximizes the vector of excesses/savings arranged in ascending order where the excess of a coalition, S, is defined by

$$e(x, S) = c(x) - \sum_{i \in S} x_i.$$

We can now define the per capita nucleolus which is the same as the nucleolus except that the excesses are now defined by

$$e(x, S) = \frac{c(x) - \sum_{i \in S} x_i}{|S|}$$

Also recall the Shapley Value, which takes the average over all possible orderings in which a player can enter the grand coalition, i.e.

$$x_i = \sum_{S \in N} \frac{(|S| - 1)!(n - |S|)!}{n!}\left[c(S) - c(S - \{i\})\right].$$

Properties

Although all of these methods, and many others, exist it is difficult to justify why one method is fairer than another. This leads us to the basic properties all of which an ideal supermethod would possess if one existed. If we consider a cost game (N, c) the following properties may be defined:

(1) efficiency - all costs are allocated

$$\sum_{j \in N} x_j(c) = c(N)$$

(2) symmetry - equal players get equally allocated

if $c(S \cup \{i\}) = c(S \cup \{j\})$ for all $S \in (N - \{i,j\})$ then

$x_i(c) = x_j(c)$

(3) proportionate - allocation is independent of unit of money used

if $c(S) = \alpha c(S)$ for all $S \subseteq N$, then $x_i(c') = \alpha x_i(c)$

(4) separates separable costs (ignores irrelevant costs)

if $b \in R^n$ and a game c' is defined as

$$c'(S) = c(S) + \sum_{i \in S} b_i$$

then, $x_i(c') = x_i(c) + b$

(5) dummy pays all - no savings created, no savings received

if $c(S \cup \{i\}) = c(S) + c(i)$ for all coalitions

S that do not contain i, then we call i a dummy and

$x_i(c) = c(i)$

(6) individually rational - no player can do better by themselves

$x_i(c) \leq c(i)$ ($x_i(c) \geq v(i)$ in a value game)

(7) coalitionally rational - no coalition can do better on their own

i. e. if core(c) $\neq \emptyset$, then $\mathbf{x}(c) \in$ core(c)

(8) aggregate monotone - an increase in c(N) (the grand coalition) does not cause a decrease in any players allocation

(9) coalitionally monotone - an increase in the cost of any particular coalition T, does not cause a decrease in the allocations of all players $i \in T$

i. e. let c' be the game in which the increase(s) occur

if $c(T) \leq c'(T)$ for some T, and $c(S) = c'(S)$ for all $S \neq T$ then for $i \in T$,

$$x_i(c) \leq x_i(c')$$

(10) strongly monotone - for every fixed $i \in N$

$c(S) - c(S - \{i\}) \leq c'(S) - c'(S - \{i\})$ for all S containing i then, $x_i(c) \leq x_i(c')$

Note: strongly monotone$\Longrightarrow$ coalitionally monotone$\Longrightarrow$ aggregate monotone

	SCRB	SV	EANC	Nucleolus	PC Nucleolus
efficient	yes	yes	yes	yes	yes
symmetric	yes	yes	yes	yes	yes
proportionate	yes	yes	yes	yes	yes
separate separable costs	yes	yes	yes	yes	yes
dummy pays all	yes	yes	no	yes	yes
individually rational	yes	yes	no	yes	no
coalitionally rational	no	no	no	yes	yes
aggregate monotone	no	yes	no	yes	yes
coalitionally monotone	no	yes	yes	no	yes
strongly monotone	no	yes	yes	no	no

The chart on the previous page shows which properties each of the methods being discussed possess. A 'yes' answer indicates that this method has that particular property for all games.

Young's Theorem

Along with these properties follow many different theorems which classify or characterize various methods axiomatically. My specific interest has been methods that are both core and monotone. It is interesting to note that although no methods of this type are known at this point the following was discovered by P. Young.

Theorem: (Young, 1985) There exists no efficient allocation method which is core and coalitionally monotone on games of $|N| \geq 5$.

Proof: (by counterexample) Consider the cost function c defined on $N=\{1,2,3,4,5\}$ as follows:

note that c(ijk..) equals the cost of coalition {i,j,k..}

$c(S_1) = c(35) = 3$

$c(S_2) = c(134) = 9$

$c(S_3) = c(123) = 3$

$c(S_4) = c(245) = 9$

$c(S_5) = c(1245) = 9$

$c(S_6) = c(N) = 11$

For any other existing coalition we define their cost, c(S), to be the

$$\min_{S_k \supset S} c(S_k)$$

where $1 \leq k \leq 6$, and let $c(\emptyset) = 0$. If **x** is in the core of c, then

$$\sum_{S_k} x_i \leq c(S_k) \text{ for } 1 \leq k \leq 5$$

adding these five equations we get,

$$3\sum_{N} x_i \leq 33 \Rightarrow \sum_{N} x_i \leq 11$$

However, we see by efficiency that the equality must hold in the latter equation. This in turn implies equalities for the previous five equations, which have a unique solution of **x** = (0, 1, 2, 7, 1) (i. e. any core method would choose this point).

Compare the game c' which is identical to c except $c'(S_5) = c'(S_6) = 12$. The same procedure above leads to a core with a unique point of **x**=(3,0,0,6,3). Because the allocation of both player 2 and player 4 decrease when the cost of some of the coalitions containing them monotonically increase, this shows that no core allocation method is coalitionally monotone for n = 5, and can be extended for n > 5 by simply adding dummy players to the game c.

Core and Coalitionally Monotone Methods

After examining Young's Theorem, one cannot help but wonder whether or not there are allocation methods that are core and coalitionally monotone on three and four person games. For the most part, this has been the heart of my research this summer, which has brought about the following results.

Theorem: The nucleolus is monotone on 3-person games.

Proof: If the vector **x** is the nucleolus, then all of the collection C_k, made up of the sets of excesses, β_i, will be balanced. In order to determine whether or not **x** is the nucleolus, we only need to check the collections up to the first K^* sets of excesses, where K^* is the point where **x** can be uniquely determined. Suppose we have a 3-person game with excess vector e(**x**). Then we can define the following five cases for β_1 and β_2:

	β_1	β_2
1.	{1,2,3}	
2.	{12,13,23}	
3.	{1, 23}	{2,3}
4.	{1,23}	{12,13}
5.	{1,23}	{2,13}

Of course, other permutations exist, but only bring about something symmetric to the above cases. Therefore, we can say that these five cases make up all possible cases because of the following:

Since $\mathbf{C}_1 = \beta_1$, β_1 must be balanced. This implies that β_1 contains a minimally balanced set {1,2,3}, {12, 13, 23}, or {1, 23}.

(1) if β_1 contains {1,2,3} the **x** is defined as follows:

$$x_1 = \frac{v(N) + 2v(1) - v(2) - v(3)}{3}$$
$$x_2 = \frac{v(N) + 2v(2) - v(1) - v(3)}{3}$$
$$x_3 = \frac{v(N) + 2v(3) - v(1) - v(2)}{3}$$

(2) if β_1 contains {12,13,23} then **x** is defined as follows:

$$x_1 = \frac{v(N) + v(12) + v(13) - 2v(23)}{3}$$
$$x_2 = \frac{v(N) + v(12) + v(23) - 2v(13)}{3}$$
$$x_3 = \frac{v(N) + v(13) + v(23) - 2v(12)}{3}$$

(3) if β_1 contains {1,23} then one of the following is true:

a. it also contains {2, 3} OR β_2 contains {2, 3} and **x** is defined by:

$$x_1 = \frac{v(N) + v(1) - v(23)}{2}$$

$$x_2 = \frac{v(N) + 2v(2) + v(23) - 2v(3) - v(1)}{4}$$

$$x_3 = \frac{v(N) + v(23) + 2v(3) - 2v(2) - v(1)}{4}$$

b. it also contains {12,13} OR β_2 contains {12,13} and **x** is defined by:

$$x_1 = \frac{v(N) + v(1) - v(23)}{2}$$

$$x_2 = \frac{v(N) + v(23) + 2v(12) - v(1) - 2v(13)}{4}$$

$$x_3 = \frac{v(N) + v(23) + 2v(13) - v(1) - 2v(12)}{4}$$

c. it also contains {2, 13} or $\beta 2$ contains {2,13} and x and **x** is defined by:

$$x_1 = \frac{v(N) + v(1) - v(23)}{2}$$

$$x_2 = \frac{v(N) + v(2) - v(13)}{2}$$

$$x_3 = \frac{v(23) + 2v(13) - v(1) - v(2)}{2}$$

In all of these cases **x** is uniquely determined.

Each of the previous equations for x_i do not have any negative coefficients of coalitions S, which contain that specific player, i. It is also known that the nucleolus is a continuous function. Therefore, we may conclude that if coalitions containing player i increase, then x_i cannot decrease.

This can also be extended to a generalized nucleolus. Let the excess of coalition S be defined as:

$$e(x,S) = \frac{\left[v(S) - \sum_{i \in S} x_i \right]}{w_{|S|}}$$

where **w** is a given vector consisting of positive values dependent on the size of the coalition, S. Then, let e(x) be the vector of excesses ordered from largest to smallest. The **w**-nucleolus is the imputation that lexicographically minimizes e(x) over the set of imputations. For example, the original nucleolus has a **w** = (1, 1, 1) on 3-person games, and the per capita nucleolus has a **w** = (1, 2, 3) on 3-person games. By the same reasoning used on the previous proof, we can say that the following about the **w**-nucleolus.

Theorem: The **w**-nucleolus is monotone on 3-person games, if and only if, $w_1 \leq w_2$.

Proof:

If we look at the formulas of x_i for the first four cases(respectively), we see again that there are no negative coefficients for coalitions containing player i. (Note: **w**>0)

Case 1 and Case 2 yield the same formulas from above.

Case 3:

$$x_1 = \frac{\left(w_1/w_2 + 1\right) v(N) + 2v(1) + v(2) - v(3) - 2w_1/w_2\, v(23)}{\left(w_1/w_2 + 3\right)}$$

$$x_2 = \frac{v(N) + \left(w_1/w_2 + 1\right)\left(v(2) - v(3)\right) + w_1/w_2\, v(23)}{\left(w_1/w_2 + 3\right)}$$

$$x_3 = \frac{v(N) + + w_1/w_2\, v(23) + 2v(3) - v(1) - 2v(2)}{\left(w_1/w_2 + 3\right)}$$

Case 4:

$$x_1 = \frac{v(N) + 2v(1) + \frac{2w_1}{w_2}v(1) - 2v(23)}{2w_2 + 1}$$

$$x_2 = \frac{w_2 v(N) - \frac{w_2}{w_2 1}v(1) + v(23) + (w_2+1)(v(12) - v(13))}{2w_2+1}$$

$$x_3 = \frac{w_2 v(N) - \frac{w_2}{w_2 1}v(1) + v(23) + (w_2+1)(v(13) - v(12))}{2w_2+1}$$

So far, the **w**-nucleolus has been monotone for the above cases. In case five, below, we can see that the formula for x_3 contains a coefficient of $(w_2\text{-}w_1)$ for v(N). This implies that if $w_2 < w_1$ then we can construct a game in which the **w**-nucleolus is not monotone. In fact, this also implies that we can construct a game in which the **w**-nucleolus is not aggregate monotone, also. Since, this is the only place where a possible negative coefficient can occur, we can say that if the **w**-nucleolus is not monotone, then is must be the case where $w_2 < w_1$.

Case 5:

$$x_1 = \frac{w_1 v(N) + w_2 v(1) - w_1 v(23)}{w_1 + w_2}$$

$$x_2 = \frac{w_1 v(N) + w_2 v(2) - w_1 v(13)}{w_1 + w_2}$$

$$x_3 = \frac{(w_2 - w_1)v(N) - w_2(v(1) + v(2)) + w_1(v(23) + v(13))}{w_1 + w_2}$$

Theorem: The nucleolus is not monotone on 4-person games.

Proof: (by counterexample)
Consider the following value game, v, on N = {1, 2, 3, 4}:

*v(1) = 0 v(12) = 0 *v(123) = 1

v(2) = 0	v(13) = 0	v(124) = 1
v(3) = 0	v(14) = 0	v(134) = 1
v(4) = 0	v(23) = 0	v(234) = 1
	*v(24) = 1	v(N) = 2
	*v(34) = 1	

The nucleolus , ν = (.25, .5, .5, .75). Now consider the same game, but increase v(123) = 2. The nucleolus in this new game, v', ν = (0,1,1,0). Notice that although we raised the value of v(123), $x_1(v') < x_1(v)$. Thus, we may conclude that the nucleolus is not monotone on 4-person games.

Of course, the question arises as to whether or not this proof can be extended to all core allocation methods, or maybe some class of core allocation methods. If we define a **strictly core** allocation method to be a core allocation method that always yields an allocation in the relative interior of the core (never yielding a point on the boundary of the core) whenever the core exists.

Theorem: There exists no efficient allocation method which is strictly core and monotone on 4-person games.

Proof:

If we again examine the original game, v, from above, we can see that there is more than one point in the core. For example, it is obvious that the points **t**=(0, 1, 1, 0), **y**=(.5, .5, .5, .5), and **z**=(.25, .5, .5, .75) are all in the core(v). However, in the game v', described above, it can be easily shown that **s**=(0, 1, 1, 0) is the unique point in the core(v'). Since, $t_1(v)$=0, **t**(v) lies on a boundary point of the core(v), and for this reason any strictly core allocation method would not choose this point. Thus, once v(123) is increased, these strictly core methods are forced to choose **s**=(0,1,1,0). Because this leads to a decrease in the value allocated to player 1, we can then conclude that there exists no efficient allocation method that is strictly core and monotone on 4-person games.

The above counterexample was discovered by examining the possible minimally balanced sets on 4-person games. Notice β_1 = {1, 24, 34, 123} is minimally balanced, and {1} ⊂ {123}. The set, T, is the unique set (except for other permutations of T) that has this subset property. Looking at Young's counterexample, we can see that his set, β_1 = {35,134,1245,123, 245}, also has this subset property, namely, {245} ⊂ {1245}. Further examination lead us to the following:

Conjecture: The nucleolus is monotone on a game (N,v), if and only if the sets, β_i, given by the excesses, do not contain coalitions S and T such that S ⊂ T.

Proof: (partial) (1) subsets ⟹ nonmonotonicity

Suppose we have a set β_k, given by the sets of excesses, and coalitions S and T such that:

$$S, T \in \beta_k \text{ and } S \subset T.$$

β_k yields equations in the form:

$$\alpha + x(R) = v(R), \; where\; x(R) = \sum_{i \in R} x_i$$

$$\alpha + x(T) = v(T), \; where\; x(T) = \sum_{j \in T} x_j$$

and we know:

$$\sum_{k \in N} x_k = v\,(N)$$

Because we are only interested in the coefficients of V(T), we can put the above equations in a generalized form:

$$\alpha + x(R) = 0$$
$$\alpha + x(T) = 1$$
$$x(N) = 0$$

We know there exists a collection $\mathbf{C}_k$ which is a balanced set with balancing vector $\mathbf{y} > 0$. Then by the following steps:

a. Multiply each of the $\alpha + x(R) = 0$ by the appropriate y_R
b. Multiply $\alpha + x(T) = 1$ by y_T
c. Add results from a and b
d. Subtract the equation $x(N) = 0$

one can arrive at:

$$\alpha \sum_{R \in \beta_k} y_R = y_T \Rightarrow \alpha = \frac{y_T}{\sum_{R \in \beta_k} y_R}$$

substituting yields:

$$x(S) = -\frac{y_T}{\sum_{R \in \beta_k} y_R}$$

Since $\mathbf{y} > 0$, we see that the coefficient of v(T) will turn out to be a negative value for at least one player, $i \in S$. Thus, we can conclude that if there exists a β_k such that $S, T \in \beta_k$ and $S \subset T$, then the nucleolus is not monotone.

(2) nonmonotonicity $\Rightarrow$ subsets

We believe, but have yet to prove, that if the nucleolus is not monotone on a given game (N, v) then there must be coalitions $S, T \in \beta_k$ such that $S \subset T$.

Conclusion

Our original goal was to find allocation methods that are core and monotone. Although this particular goal was not achieved, we did show for some classes of games there are certain core methods that are monotone and certain methods that are not. Further work should focus on determining whether all core allocation methods are monotone on 4-person games. In addition, one could hope for a proof of the conjectured classification of all games for which the nucleolus is not monotone.

Appendix
Nucleolus Algorithm

```
ses printer, crt;

onst maxn = 8;          { maximum number of players }
     maxc = 255;         { maximum number of coalitions less one }

ype  Filepath  = String[33];
     Game = array[0..maxc] of Real;
     Soln = array[1..maxn] of Real;

ar   n : Integer;                                     { number of players }
     v : Game;                                { payoffs for each coalition }
     x : Soln;                                 { payoffs in the allocation }
     snum, tnum : Integer;                    { number of elements in s,t }
     parm1, parm2, i : Byte;                     { allocation parameters }
     errorparm : Real;                  { epsilon for ending recursion in
                                                       finding nucleolus }
     key : Char;

unction power( a, b : byte ): Integer;
 var x : Integer; i : byte;
 begin
   x := 1;
   for i :=1 to b do x := x * a;
   power := x;
 end;

unction ele( i : Byte; s : Integer ) : Boolean;
 var k : byte;
 begin
   for k:=1 to i-1 do s:=s div 2;
   ele := Odd(s);
 end;

unction singleton( i : Byte ) : Integer;
ar j, k : Integer;
egin
  j := 1;
  for k:= 1 to i-1 do j := j * 2;
  singleton := j;
nd;

unction size( j : Integer ) : Byte;
ar i, k : Byte;
egin
  k := 0;
  for i := 1 to maxn do if ele(i,j) then k := k + 1;
  size := k;
nd;

{$I WTNUC }


procedure LoadProblem;
 var FF : Filepath; F : Text;
 begin
  Write('Filename:  '); Readln(FF); Writeln;
  Assign(F,FF); Reset(F);
  Readln(F,n);
  for snum:=0 to power(2,n) - 1 do Readln(F,v[snum]);
  Close(F);
 end;
```


procedure SaveProblem;

```
begin
 Write('Filename: '); Readln(FP); Writeln;
 Assign(F,FP); Rewrite(F);
 Writeln(F,n);
 for snum:=0 to power(2,n) - 1 do Writeln(F,v[snum]);
 Close(F);
end;

rocedure ReadProblem;
begin
 Write('n = '); Read(n); Writeln;
 for snum:=0 to power(2,n) - 1 do  begin
   Write('v( ');
   for i:=1 to n do if ele(i,snum) then Write(i,   ');
   Write(') = ');
   Read(v[snum]); Writeln;
 end;
end;

rocedure WriteProblem;

ar ans: string[1];

begin
write('Destination:  Screen  Printer   ');
readln(ans);

if ans = 's' then begin
 Writeln('n = ',n);
 for snum:=0 to power(2,n) - 1 do
 begin
   Write('v( ');
   for i:=1 to n do if ele(i,snum) then Write(i,   ') else Write(     ');
   Write(') = ');
   Writeln(v[snum]:parm1:parm2);
 end;
end

else begin
 Writeln(lst,'n = ',n);
 for snum:=0 to power(2,n) - 1 do
 begin
   Write(lst,'v( ');
   for i:=1 to n do if ele(i,snum) then Write(lst,i,' ')
    else Write(lst,'   ');
   Write(lst,') = ');
   Writeln(lst,v[snum]:parm1:parm2);
 end;
end;
nd;


rocedure WriteAllocation;
begin
 for i:=1 to n do Write(X[i]:parm1:parm2);
 Writeln;
end;

rocedure CoreCheck;
var t : Real; i : Byte; violation : Boolean;
begin
  snum := 1; tnum := power(2,n) - 1;
  repeat
     t := 0; for i := 1 to n do if ele(i,snum) then t := t + x[i];
     violation := t > v[snum];                        {individual rationality}
```

```
  if violation then Write(' ') else Write('*'):
 end:

procedure Fair(a.r : byte: var va:Real):
var t: Real:
    snum. k: byte:
begin
    for snum := 1 to power(2.n) -1 do
    begin
    { is allocation for player a fair? }
           if ele(a.snum) and not ele(r.snum) then
           begin
             t := v[snum]:  {initialize surplus to total payoff for coalition}
             for k:=1 to n do
               if ele(k.snum) and (k <> a) then t := t - x[k]:
                   {take out payoffs to others in this coalition - surplus}
             if t > va then va := t:
                {if surplus more than what player a has. give all to player a}
           end:                                                         {if-then}
    end:                                                                   {snum}
end:                                                                  {procedure}


procedure SpecifyAllocation:

var i: integer;

begin:
  for i := 1 to n do
  begin
    write ('x[',i,'] = '):
    readln(x[i]):
  end:
writeln:
end:
procedure Nucleolus:
var i    : Integer:
    key  : Char:
    w    : Soln:
begin
  Writeln('Nucleolus  Per capita nucleolus  W-nucleolus '):
  Write('Press desired key: '): REadln(key): Writeln:
  case Upcase(key) of
    'N' : for i:=1 to n-1 do w[i] := 1:
    'P' : for i:=1 to n-1 do w[i] := i:
    'W' : for i:=1 to n-1 do begin Write('W[',i,'] = '): Readln(w[i]) end:
  end:
  WeightedNucleolus(n.v.w.x):
  WriteAllocation:
end:


procedure TransferScheme:
var i. j. k. jmin : Byte:                          { loop counters }
    vi.                                    { possible payoff for player i alone }
    vj.                                    { possible payoff for player j alone }
    vij.                                   { amount to be divided between i and j }
    t.                                     { }
    error : Real:                          { allowable error for ending procedure }

    xold : array[1..maxn] of Real:         { save previous iteration in vector }
    ans : string[1]:
```



```
begin
  write ('Do you want to specify the starting allocation?  '):
```

```
  tnum := power(2,n) - 1;                 { assign set of all players }
  error := errorparm*v[tnum]/n;           { divide error between each player }
  if ans = 'y' then SpecifyAllocation
  else
  for k:=1 to n do x[k] := v[tnum]/n;  { start with equal allocation
                                          among all n players }
  repeat
    WriteAllocation;                      { write allocation }
    for i:=1 to n do xold[i] := x[i]; { save old allocation for comparing
                                          with new allocation }
    for i:=1 to n-1 do                  { consider every coalition of 2 players }
      for j:= i + 1 to n do
      begin
        vij := x[i] + x[j];                         { requirement of consistency }
        vi := v[power(2,i-1)];                       { payoff for player i alone }
        vj := v[power(2,j-1)];                       { payoff for player j alone }

        fair(i,j,vi);                         { satisfy consistency for player i }
        fair(j,i,vj);                         { satisfy consistency for player j }

        t := (vij - vi - vj)/2;                  { formula to satisfy covariance }
        x[i] := vi + t;
        x[j] := vj + t;
      end;                                                              { j-loop}

    t := 0; for k:=1 to n do t := t + Sqr(x[k] - xold[k]);
                         {find "sum of squared residuals" for this iteration}
  until t < error;
  CoreCheck; WriteAllocation; Writeln;
end;                                                               {procedure}

procedure ParameterChange;
  begin
    Write('Allocation column width (',parm1,'): '); Read(parm1); Writeln;
    Write('Allocation decimal places (',parm2,'): '); Read(parm2); Writeln;
    Write('Nucleolus relative error (',errorparm:10:6,'): '); Read(errorparm);
    Writeln;
  end;

procedure exvector ;
var xsum : real;
    excess: array[0..maxc] of real;
    coal: array[0..maxc] of integer;
    temp1: real;
    i,j,temp2 : integer;
    ans: string[1];

begin
{ calculate excess vector }
  for i:=0 to power(2,n) - 1 do  begin
    xsum := 0;
    for j:=1 to n do if ele(j,i) then xsum := xsum + x[j];
    excess[i] := v[i] - xsum;
    coal[i] := i;
  end;

{ sort excess vector }
  for i:= 0 to power(2,n) - 2 do
    for j := i + 1 to power(2,n) - 1 do
      if excess[i] < excess[j] then
      begin
        temp1 := excess[i];
        excess[i] := excess[j];
        excess[j] := temp1;
        temp2 := coal[i];
```

```
      coal[i] := coal[j];
      coal[j] := temp2;
    end;

write ('Destination:  Screen   Printer  ');
Readln(ans);

if ans = 's' then begin
{ write sorted vector }
  for i := 0 to power(2,n) - 1 do begin
    Write('e(x, ');
    for j:=1 to n do if ele(j,coal[i]) then write(j,' ') else write('  ');
    Write (') = ');
    Write (excess[i]:parm1:parm2);
    Writeln;
  end;
end

else begin
{ print sorted vector }
  for i := 0 to power(2,n) - 1 do begin
    Write(lst, e(x, ');
    for j:=1 to n do if ele(j,coal[i]) then write(lst,j,' ')
     else write(lst,'  ');
    Write (lst,') = ');
    Write (lst,excess[i]:parm1:parm2);
    Writeln(lst);
  end;
end;
nd;


egin
 n := 0; parm1 := 10; parm2 :=3; errorparm := 0.001;
 repeat
   Writeln('Select Activity');
   Writeln('Load Save Read or Write problem   Parameter change');
   Writeln('Nucleolus  Excess vector  specify Allocation  Transfer scheme');
   Writeln('Quit');
   Write('Press desired key: '); Readln(key); Writeln;
   case Upcase(key) of
     'L' : LoadProblem;
     'S' : SaveProblem;
     'R' : ReadProblem;
     'W' : WriteProblem;
     'T' : TransferScheme;
     'N' : Nucleolus;
     'E' : Exvector;
     'A' : SpecifyAllocation;
     'P' : ParameterChange;
   end;
 until Upcase(key) = 'Q';
```

```
nd.procedure WeightedNucleolus( n : Integer; v : Game; w : Soln; var x : Soln )

onst MaxRows  =   9;    { must be > n + 1 }
     MaxCols  = 520;    { must be > 2**n - 1 + n }
     Zero =  1.0E-3;    { round-off tolerance for some comparisons }

ype DualVector       = array[1..MaxRows] of Real;
    BasisVector      = array[1..MaxRows] of Integer;
    PrimalVector     = array[1..MaxCols] of Real;
    BoolVector       = array[1..MaxCols] of Boolean;
    BasisMatrix      = array[1..MaxRows,1..MaxRows] of Real;
    Pointer          = ^RedundantType;
    RedundantType    = record col:integer; next:pointer; end;

ar LPM       : Integer;        { n + 1 }
   LP2       : Integer;        { 2**n - 1 }
   LPN       : Integer;        { 2**n - 1 + n }
   C         : PrimalVector;   { objective function coefficients }
   Finite    : BoolVector;     { true if y(j) > 0 constraint is present }
   Rank      : Integer;        { rank of equations restricting x }
   R         : BasisMatrix;    { equations restricting x }
   LPiter    : Integer;
   LPX, LPY  : DualVector;
   LPV       : Real;
   Basis     : BasisVector;
   Bounded   : Boolean;
   Redundant, Old : Pointer;


rocedure FirstEquation;   {sum of all xi = v(N)}
ar i, j : Integer;
egin
 Rank := 1;
 for j:=1 to n do R[1,j] := 1;
 for i:=2 to n do for j:=1 to n do R[i,j] := 0;
nd;

unction InList(j: integer):Boolean;
egin
ld := Redundant;
hile (Old <> Nil) and (Old^.col <> j) do Old := Old^.next;
nList := Old <> nil;
nd;

rocedure AddEquation;
ar i, j1: Integer;
   Key: char;

 procedure NewEquation;
 var k : Integer;
 begin
   Rank := Rank + 1;
   if j1 <= LP2 then
    for k := 1 to n do
      begin  {convert number to equation in Rank matrix}
       if ele(k,j1) then R[Rank,k] := 1
       else R[Rank,k] := 0
      end
   else
    begin  {set a free variable to equality}
    for k := 1 to n do R[Rank,k] := 0;
    R[Rank,j1-LP2] := 1;
```

```
  end;
end;

function FirstNonZero(i:integer):integer;
{find position of first nonzero element in row i}
var s:integer;
begin
s := 0;
repeat  s:= s + 1 until (abs(R[i,s]) > Zero) or (s > n);
FirstNonZero := s;
end;


procedure Reduce(variable:integer);
var i, j, k : integer;
    t : Real;
    Temp: array [1..MaxRows] of Real;

begin
 repeat
   j := FirstNonZero(Rank);
   if j <= n then {new row is not all zeros}
     begin
     i := 1;
     while (FirstNonZero(i) < j) do i := i + 1;
      {find position of new row in rank matrix}
     if (FirstNonZero(i) = j) and (i < rank) then
          begin
            { reduce }
            t := R[Rank,j]/R[i,j];
            for k := j to n do R[Rank,k] := R[Rank,k] - t*R[i,k];
          end;
      end;
 until (j >= n) or (FirstNonZero(i) > j) or (i = Rank);
 if FirstNonZero(i) > j then
   begin
   { flip }
   repeat
     for k := 1 to n do temp[k] := R[i,k];
     for k := 1 to n do R[i,k] := R[Rank,k];
     for k := 1 to n do R[Rank,k] := temp[k];
     i := i + 1;
   until i = rank;
   end;
 if FirstNonZero(Rank) > n then
   begin
   Rank := Rank - 1; {new row is all zeros}
   Old := Redundant;
   New(Redundant);
   Redundant^.col := variable;
   Redundant^.next := Old;
   end;
 end;


egin  { procedure AddEquation }
 j1 := 0;{cols}
 i := 1; {rows}
   repeat
   if LFY[i] > +Zero then
   begin
     j1 := Basis[i];
     NewEquation;
     Reduce(j1);
```

```
  i := i + 1; {increment rows}
until (i > LPM) or (Rank = n);
nd;  { procedure AddEquation }

unction LPA( i, j : Integer ) : Real;
egin
 LPA := 0;
 if j <= LP2 then
 begin
   if (i = LPM) and Finite[j] then LPA := w[size(j)];
   if (i < LPM) and ele(i,j) then LPA := 1;
 end
 else
   if i = j - LP2 then LPA := 1;
nd;

unction LPB( i : Integer ) : Real;
egin
 if i = LPM then LPB := 1 else LPB := 0;
nd;

unction LPC( j : Integer ) : Real;
egin
 LPC := C[j]
nd;

unction LPF( j : Integer ) : Boolean;
egin
 LPF := Finite[j]
nd;

rocedure BasisSetup;
ar i, j : Integer;
egin
 j := 1; while not LPF(j) do j := j + 1; Basis[1] := j;
 Basis[2] := LP2;
 i := 1; while not ele(i,j) do begin Basis[2+i] := LP2 + i; i := i + 1 end;
 i := i + 1; while i <= LPM do begin Basis[1+i] := LP2 + i; i := i + 1 end;
nd;

rocedure LPSetup;
ar i, j : Integer;
egin
 LPM := n + 1;
 LP2 := power(2,n) - 1;
 LPN := LP2 + n;
 for i:=1 to LP2 do C[i] := v[i];
 for i:=1 to n do C[LP2 + i] := v[singleton(i)];
 for i:=1 to LPN do Finite[i] := true; Finite[LP2] := false;
 BasisSetup;
nd;

rocedure LPChange;
ar i, j : Integer;
egin
 for i:=1 to LPM do
   if LPY[i] > +Zero then
   begin
     j := Basis[i];
     C[j] := C[j] - LPV;
     Finite[j] := false;
   end;
 BasisSetup;
nd;
```

```
*
LP is a procedure that solves linear programs of the form
                 max  cx
                 s.t. Ax = b
                      x(j) ≥ 0, j ∈ F
given a basic feasible solution.  The problem data is passed to this
procedure by way of the following user defined variables/constants:
     LPM              number of rows of A
     LPN              number of columns of A
and functions (returning Real values):
     LPA(i,j)         (i,j) component of A
     LPB(i)           i component of b
     LPC(j)           j component of c
and function (returning Boolean values):
     LPF(j)          true  if the j component of x is nonnegative

The input/output parameters have the following interpretation:
     x(j)   =   X[i]   if   j = Basis[i]      a basis feasible solution
                0      otherwise
     y(i)   =   Y[i]                           a dual solution
     cx     =   V                              objective function value
                Bounded                        true  if optimal solution found
                                               false if problem is unbounded

The routine uses the revised simplex method as outlined in Vasek Chv tal.
"Linear Programming," p. 103. with a small modification to handle free
variables.  Note that the basis matrix  AB  is refactored in each
iteration; hence, the procedure is slow but accurate.

Say something about MaxRows and MaxCols and Zero.
}


rocedure LP( var X, Y    : DualVector;
             var V       : Real;
             var Basis   : BasisVector;
             var Bounded : Boolean );


ar i          : Integer;      { Leaving row }
   j          : Integer;      { Entering variable/column }
   AB         : BasisMatrix;  { Basis matrix or transpose }
   D          : DualVector;   { Exchange vector }
   tmax       : Real;         { Maximum change possible for variable j }
   eligible   : Boolean;      { true if a column can enter }
   positive   : Boolean;      { true if reduced cost c - ya is positive }
   iter       : Integer;      { number of iterations }
   Key        : Char;

procedure SolveSystem( var A :BasisMatrix; var b :DualVector );
var i, j, k : Integer;
    t       : Real;
begin
  for i:=1 to LPM do
  begin
    k:=i;
    for j:=i+1 to LPM do if abs(A[k,i])<abs(A[j,i]) then k:=j;
    if k<>i then
    begin
      t:=b[i]; b[i]:=b[k]; b[k]:=t;
      for j:=i to LPM do begin t:=A[i,j]; A[i,j]:=A[k,j]; A[k,j]:=t end;
    end;
    for k:=i+1 to LPM do
      if A[k,i]<>0 then
      begin
```

```
      for j:=i to LPM do A[k,j]:=A[k,j]-t*A[i,j];
    end;
  end;
  for k:=LPM downto 1 do
  begin
    for j:=LPM downto k+1 do b[k]:=b[k]-A[k,j]*b[j];
    b[k]:=b[k]/A[k,k]
  end;
end;

procedure Step0;  { Find current bfs and obj fn value }
var i, j, k : Integer;
begin
  for k:=1 to LPM do for i:=1 to LPM do AB[i,k]:=LPA(i,Basis[k]);
  for i:=1 to LPM do X[i]:=LPB(i);
  SolveSystem(AB,X);
  V := 0;
  for i:=1 to LPM do V := V + LPC(Basis[i]) * X[i];
end;

procedure Step1;  { Find dual vector }
var i, k : Integer;
begin
  for k:=1 to LPM do for i:=1 to LPM do AB[k,i]:=LPA(i,Basis[k]);
  for k:=1 to LPM do Y[k]:=LPC(Basis[k]);
  SolveSystem(AB,Y);
end;

procedure Step2;  { Find an entering variable j }
var t : Real; i : Integer; B : set of 1..MaxRows;
begin
  B := []; for i:=1 to LPM do B := B + [Basis[i]];
  j := 0;
  repeat
    j := j + 1;
    if not (j in B) and (not InList(j)) then
    begin
      t := LPC(j); for i:=1 to LPM do t := t - Y[i] * LPA(i,j);
      positive := (t > +Zero);
      eligible := positive or ((not LPF(j)) and (abs(t) > +Zero));
    end
    else eligible := false;
  until eligible or (j = LPN);
end;

procedure Step3;  { Find exchange vector for entering column j }
var i, k : Integer;
begin
  for k:=1 to LPM do for i:=1 to LPM do AB[i,k]:=LFA(i,Basis[k]);
  for i:=1 to LPM do D[i]:=LFA(i,j);
  SolveSystem(AB,D);
end;

procedure Step4;  { Find leaving row i }
var t : Real; k : Integer; FoundFirst, FoundOne: boolean;
begin
  FoundFirst := false;
  for k:=1 to LPM do
  begin
    if LPF(Basis[k]) then
    begin
      FoundOne := false;
      if positive and (D[k] > Zero) then
      begin
        t := X[k]/D[k];
```

```
        FoundOne := true
      end;
      if (not positive) and (D[k] < -Zero) then
      begin
        t := -X[k]/D[k];
        FoundOne := true
      end;
    end;
    If FoundOne and (not FoundFirst) or (FoundFirst and (t < tmax)) then
    begin
      i := k;
      tmax := t;
      FoundFirst := true
    end;
  end;
end;

procedure Step5;   { Update primal basic feasible solution }
begin
  Basis[i] := j;
end;

gin   { procedure LP }
iter := 0;
repeat
  iter := iter + 1;
  Step0;
  Step1;
  Step2; if not eligible then begin Bounded := true; Exit end;
  Step3;
  Step4; if i=-1 then begin Bounded := false; Exit end;
  Step5;
until false;
d;    { procedure LP } .

r i : Integer;
gin   { procedure WeightedNucleolus }
LPiter := 0;
Redundant := nil;
FirstEquation;
repeat
  LPiter := LPiter + 1;
  if LPiter = 1 then LPSetup else LPChange;
  LP(LPY,LPX,LPV,Basis,Bounded);
  AddEquation;
until Rank = n;
for i:=1 to n do x[i] := LPX[i];
```

References

1. Chvàtal, Vasek, Linear Programming, New York: W.H. Freeman and Company , 1983, p. 103.
2. Kohlberg, Elon "On the Nucleolus of a Characteristic Function Game," SIAM Journal of Applied Mathematics, Vol. 20, 1981, pp. 62-66.
3. Lucas, William F. "Applications of Cooperative Games," Game Theory and its Applications: Proceedings of Symposia in Applied Mathematics, Providence, RI: American Mathematical Society, 1981, Vol 24, pp.19-36.
4. Lucas, William F. "The Multiperson Cooperative Games," Game Theory and its Applications, pp. 1-17.
5. Owen, Guillermo, "Balanced Collections," Game Theory, New York: Academic Press, 1982, pp. 156-164.
6. Owen, Guillermo, "The Bargaining Set," Game Theory, pp. 236-242.
7. Owen, Guillermo, "The Kernel," Game Theory, pp. 242-244.
8. Owen, Guillermo, "The Nucleolus," Game Theory, pp. 244-256.
9. Shapley, L.S., "Valuation of Games," Game Theory and its Applications, pp. 55-67.
10. Sobolev, A.I., "Characterization of the Principle of Optimality for Cooperative Games through Functional Equations," in N.N. Voroby'ev (Ed.), Mathematical Methods in the Social Sciences, Vipusk, 6, Vilnius, USSR (1975), 92-151.
11. Stearns, R.E., "Convergent Transfer Schemes for N-person Games," Transactions of the American Mathematical Society, Vol. 134, 1968, pp. 449-459.

12. Tijs, S.H., "An Axiomatization of the τ-value," Mathematical Social Sciences, Vol. 13, 1987., pp. 177-181.

13. Young, Peyton H. "Cost Allocation," Fair Allocation: Proceedings of Symposia in Applied Mathematics, Providence, RI: American Mathematical Society, 1985, Vol. 33, pp. 69-94.

14. Young, Peyton H. "Monotonic Solutions of Cooperative Games," International Journal of Game Theory, 1985, Vol. 4, Issue 2, pp. 65-72.