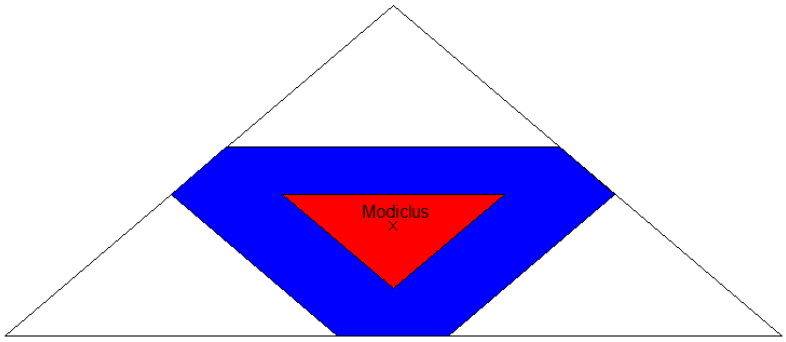
Using the R package `CoopGame` for the analysis, solution and visualization of cooperative games with transferable utility



Jochen Staudacher, Johannes Anwander
(Hochschule Kempten)
Contact: jochen.staudacher@hs-kempten.de

2021-08-21

Abstract

This document gives a brief and concise overview of the various functionalities of the package `CoopGame` and presents a few use cases. In particular, we introduce the capabilities of `CoopGame` to create special families of cooperative games, to check game properties and to compute set-valued and point-valued solutions. We also introduce the usage of `CoopGame` for visualizing set-valued and point-valued solutions in the case of three or four players. We end with a brief outlook to future developments. This vignette accompanies version 0.2.2 of the package `CoopGame`.

Keywords: Cooperative game theory, bankruptcy games, cost games, simple games, unanimity games, weighted voting games, game properties, set-valued solution concepts, point-valued solution concepts, imputation, core, Weber set, Shapley value, Banzhaf value, nucleolus, nucleolus derivatives, modiclus, Gately point, Tau value, power indices.

Contents

1 Introduction to Cooperative Games

Today, the term game theory is mostly interpreted as interactive decision theory, i.e. game theorists wish to study how agents take strategic decisions interactively. Since the publication of the seminal book "Theory of Games and Economic Behavior" by John von Neumann and Oskar Morgenstern [1] in 1944, modern game theory developed into two main branches. These two branches are most frequently referred to as noncooperative game theory and cooperative game theory, respectively. In our understanding of the field we side with the views expressed by Nobel laureate Robert Aumann in his famous interview [2] with Eric van Damme that a better name for noncooperative game theory would be "strategically oriented game theory" whereas cooperative game might be characterized more precisely as "coalitional game theory" or "outcome oriented game theory".

In the introduction of this vignette we wish to borrow a thought from the first chapter of the book by Peleg and Sudhölter [3]. In the seminal papers [4] and [5] Bezalel Peleg pointed out that Nash's program could not be implemented, i.e. that it is not possible to express each and every cooperative game as a noncooperative game in extensive form with the solution of the cooperative game being defined in terms of equilibrium points of the corresponding noncooperative game.

In other words: Peleg's results from [4] and [5] removed any doubts that cooperative game theory was truly a theory of its own right. So, to quote Robert Aumann (see [2], p. 195) again

`... It is not only strategic interaction.`

In this spirit our R-package `CoopGame` is devoted to the study of coalitional games with transferable utility and aims to provide a comprehensive set of methods.

1.1 What this package provides and what it does not provide

The package `CoopGame` focusses on a cost-savings approach to cooperative games.

We are studying a transferable utility game (TU game) v in characteristic function form consisting of the player set $N = \{1, \ldots, n\}$ and the characteristic function $v : 2^N \to \mathbb{R}$ with $v(\emptyset) = 0$. We specify such a TU game with n players as a (game) vector of length $2^n - 1$.

1.1.1 Vignette for version 0.2.2 of `CoopGame`

This is the place to emphasize once again that this vignette accompanies version 0.2.2 of the `R` package `CoopGame`. Future releases of `CoopGame` might incorporate additional functionality and will be accompanied by an update of this vignette.

1.1.2 Specifying a TU game

Let us look at an introductory example of a three-player TU game. Let $N = \{1, 2, 3\}$ and $v : \mathcal{P}(N) \to \mathbb{R}$ be specified as

$$
\begin{aligned}
v(\emptyset) &= 0, \\
v(\{1\}) &= 0, \\
v(\{2\}) &= 0, \\
v(\{3\}) &= 0, \\
v(\{1,2\}) &= 100, \\
v(\{1,3\}) &= 115, \\
v(\{2,3\}) &= 125, \\
v(\{1,2,3\}) &= 220.
\end{aligned}
$$

We may interpret the values of the individual coalitions in various ways, e.g. as the worth of a coalition or the cost-savings of a coalition. In our above example the singleton coalitions have worth zero whereas the grand coalition $N = \{1, 2, 3\}$ makes a total of 220 (monetary units). In brief, the basic question in coalitional games with transferable utility is how we can share the worth of 220 among our three players. In `R` we can simply specify the above game as a vector v of lenght 7, i.e.

```
library(CoopGame)
## Loading required package: geometry
## Loading required package: rcdd
## If you want correct answers, use rational arithmetic.
## See the Warnings sections added to help pages for
##     functions that do computational geometry.
(v <- c(0,0,0,100,115,125,220))
## [1]   0   0   0 100 115 125 220
```

Sometimes it is convenient to use the shorthand notations

$$
v_i = v(\{i\}) \quad \text{for} \quad i = 1, \ldots, n,
$$

for the singleton coalitions.

1.1.3 A popular example: The Maschler game

Throughout this vignette we will frequently employ the following three-player example

$$
v(C) = \begin{cases} 0, & |C| = 1 \\ 60, & |C| = 2 \\ 72, & |C| = 3 \end{cases}
$$

It appears in the article [6] where the authors attribute it to the famous Israeli game theorist Michael Maschler. The above game has thus frequently been referred to as the Maschler game in the literature since.

```
library(CoopGame)
(Maschler <- c(0,0,0,60,60,60,72))
## [1]  0  0  0 60 60 60 72
```

1.1.4 Functionality and structure of the package `CoopGame`

The package `CoopGame` provides functions for

- generating TU games with special structure (see chapter 2 of this vignette), like e.g. bankruptcy games, cost games, weighted voting games and unanimity games

- checking game properties (see chapter 3 of this vignette), like e.g. superadditivity, convexity and balancedness
- computing a number of set-valued solution concepts for TU games (see chapter 4 of this vignette), including the core and various core catchers
- computing a large array of point-valued solution concepts for TU games (see chapter 5 of this vignette), including the Shapley value, the nucleolus as well as various nucleolus derivatives and numerous power indices
- drawing both set- and point-valued solution concepts for the 3- and 4-player cases (see chapter 6 of this vignette)
- some general functionality useful in the context of TU games (see section 1.2 of this vignette), like e.g. computing the unanimity coefficients for a given game vector

1.1.5 Models and questions beyond the scope of `CoopGame`

Coalitional games can be turned into more realistic models if, in addition one allows for partitions of the player set (see e.g. the original paper by Thrall and Lucas [7]) or for specifying an undirected graph connecting the players. The latter case is frequently referred to as a communication game (see e.g. the book by Slikker and van den Nouweland [8]). The authors are currently developing additional `R` packages for both games with partitions of the player set as well as communication games. These additional software packages will make use of `CoopGame`, but `CoopGame` itself will allow for neither partitions or communication structures.

Since `CoopGame` can neither handle partitions nor other more general coalition structures (see e.g. the paper by Aumann and Dreze [9] from 1974), the package neither provides methods for computing bargaining sets (see e.g. the original paper by Aumann and Maschler [10] from 1964) nor methods for computing the kernel (see e.g. the original article by Aumann, Peleg and Rabinowitz [11] from 1965).

Another feature this first release of `CoopGame` also lacks is functionality for general market games (see e.g. the original paper by Aumann [12] on markets with a continuum of traders from 1964).

Also, NTU games, i.e. cooperative games with non-transferable utility (see e.g. the article by Aumann and Maschler [13] from 1960), are beyond the scope of this `R` package.

1.1.6 This is not a book on cooperative game theory and these are not lecture notes

This text is a CRAN vignette only and it is supposed to give users of `CoopGame` a concise overview of the package. Users looking for an introduction to the fascinating field of cooperative game theory will be disappointed by this text. We strictly adhere to the CRAN policy that CRAN vignettes are not to be misused as lecture notes.

For excellent introductions to cooperative game theory we refer to the books by Peleg and Sudhölter [3], by Driessen [14], by Branzei, Dimitrov and Tijs [15], by Chakravarty, Mitra and Sarkar [16] and by Gilles [17].

Excellent introductions to cooperative game theory can also be found in the game theory books by Maschler, Solan and Zamir [18], by Peters [19] by Osborne and Rubinstein [20], the recent book by Narahari [21] and the classic textbook by Straffin [22].

We finally wish to acknowledge that beyond the aforementioned excellent books the development of `CoopGame` also benefitted greatly from two excellent German textbooks, i.e. the books by Wiese [23] and by Holler and Illing [24].

1.2 Some general functionality for cooperative games

In this section we introduce some useful functionality for cooperative games which we will use later when we discuss games with special structure, game properties as well as set-valued and point-valued solution concepts for cooperative games.

1.2.1 Zero-normalized and zero-one-normalized games

We call a cooperative game v **zero-normalized** if

$$v_i = 0 \quad \text{for} \quad i = 1, \ldots, n,$$

i.e. the values of all singleton coalitions are zero (see e.g. Peleg and Sudhölter [3], p. 11).

We can easily zero-normalize a given game vector v into a corresponding zero-normalized game vector w via

$$w(C) = v(C) - \sum_{i \in C} v_i$$

for every coalition $C \in \mathcal{P}(N)$ (see e.g. Branzei, Dimitrov and Tijs [15], p. 9). In `CoopGame` we provide a corresponding function `getZeroNormalizedGameVector`:

```
library(CoopGame)
v <- c(30,40,50,90,100,110,180)
(w <- getZeroNormalizedGameVector(v))
## [1]  0  0  0 20 20 20 60
```

We call a cooperative game v **zero-one-normalized** if it is zero-normalized and for the grand coalition N there holds $v(N) = 1$. We can easily zero-one-normalize a given game vector v by dividing the zero-normalized game vector w by the value of the grand coalition $v(N)$. In `CoopGame` we provide a corresponding function `getZeroOneNormalizedGameVector`:

```
library(CoopGame)
v <- c(30,40,50,90,100,110,180)
(w01 <- getZeroOneNormalizedGameVector(v))
## [1] 0.0000000 0.0000000 0.0000000 0.3333333 0.3333333
## [6] 0.3333333 1.0000000
```

For a detailed discussion on the importance of zero-one-normalized games and strategic equivalence we refer the reader to the book by Maschler, Solan and Zamir [18], p. 670.

1.2.2 Why bit matrices are helpful in cooperative game theory

We introduce the concept of a bit matrix as we find it extremely useful for working with cooperative games. Bit matrices unambigously map the values of coalitions to the players in that coalition. We again make use of the Maschler game for our example.

```
library(CoopGame)
(Maschler <- c(0,0,0,60,60,60,72))
## [1]  0  0  0 60 60 60 72
createBitMatrix(n=3,Maschler)
##            cVal
## [1,] 1 0 0    0
## [2,] 0 1 0    0
## [3,] 0 0 1    0
## [4,] 1 1 0   60
```

```
## [5,] 1 0 1   60
## [6,] 0 1 1   60
## [7,] 1 1 1   72
```

For a TU game with n players the function `createBitMatrix` creates a bit matrix with $n+1$ columns and $2^n - 1$ rows which contains all possible coalitions of the players (apart from the null coalition). Each player is represented by a column which states if this player is participating in a coalition (value 1) or not (value 0). The last column (named `cVal`) contains the values of each coalition. Accordingly, each row of the bit matrix expresses a coalition as a subset of all players.

However, we need to admit that our usage of bit matrices in `CoopGame` also consumes plenty of storage space.

1.2.3 Marginal contributions

Let our TU game be specified by its characteristic function v. Then for every player $j \in N$ and for every coalition $C \in \mathcal{P}(N)$ with $j \in C$ we can define the so-called **marginal contribution** of player j to coalition C as

$$v(C) - v(C \backslash \{j\}),$$

see e.g. the book by Branzei, Dimitrov and Tijs [15], pp. 6, or the book by Peters [19], pp. 156. For some point- and set-valued solution concepts we need to know the marginal contributions of every player for every permutation of the set of players. Our function `getMarginalContributions` provides the user with a list of all combinations, i.e. permutations of the players, used and a corresponding matrix of marginal contributions:

```
library(CoopGame)
v <- c(3,4,5,9,10,11,18)
(MC <- getMarginalContributions(v))
## $A
## [1]  3  4  5  9 10 11 18
##
## $combinations
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    1    3    2
## [3,]    2    1    3
## [4,]    2    3    1
## [5,]    3    1    2
## [6,]    3    2    1
##
## $marginal_values
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]    3    8    7
## [3,]    5    4    9
## [4,]    7    4    7
## [5,]    5    8    5
## [6,]    7    6    5
# Look at all the permutations computed
MC$combinations
##      [,1] [,2] [,3]
```

```
## [1,]    1    2    3
## [2,]    1    3    2
## [3,]    2    1    3
## [4,]    2    3    1
## [5,]    3    1    2
## [6,]    3    2    1
# Look at the matrix of marginal values
# corresponding to these permutations
MC$marginal_values
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]    3    8    7
## [3,]    5    4    9
## [4,]    7    4    7
## [5,]    5    8    5
## [6,]    7    6    5
```

It appears worthwhile to interpret the second line of the marginal values we just computed

```
MC$marginal_values[2,]
## [1] 3 8 7
```

along the lines of the book by Peters [19], pp. 156. The above results corresponds to the permutation $(1, 3, 2)$.

- Player 1 enters first and contributes 3.
- Player 3 enters second and contributes $11 - 3 = 8$.
- Player 2 enters last and contributes $18 - 11 = 7$.

1.2.4 The dual game

Let our TU game be specified by its characteristic function v. Then we can specify the so-called **dual game** v^* corresponding to v via

$$v^*(C) = v(N) - v(N \backslash C)$$

for every coalition $C \in \mathcal{P}(N)$ (see e.g. the book by Peleg and Sudhölter [3], p. 125, or the book by Branzei, Dimitrov and Tijs [15], p.7, for more details). The package `CoopGame` provides a function `getDualGameVector`:

```
library(CoopGame)
v <- c(3,4,5,9,10,11,18)
# Compute dual game vector
(vStar <- getDualGameVector(v))
## [1]  7  8  9 13 14 15 18
```

1.2.5 The utopia payoff vector

In cooperative game theory the so-called **utopia payoff** of player j is defined as

$$M_j = v(N) - v(N \backslash \{j\}) \quad \text{for} \quad j = 1, \ldots, n,$$

i.e. the utopia payoff M_j is the marginal contribution of player j to the grand coalition N (see e.g. the book by Branzei, Dimitrov and Tijs [15], p. 20). The package `CoopGame` provides a function `getUtopiaPayoff` for computing a vector of utopia payoffs for all players:

```
library(CoopGame)
v <- c(3,4,5,9,10,11,18)
# Compute utopia payoff vector for specified game v
(M <- getUtopiaPayoff(v))
## [1] 7 8 9
```

It is clear that player j can not ask for more than M_j in the grand coalition. The utopia payoff vector will lateron play a role in defining game properties as well as point- and set-valued solution concepts.

1.2.6 The minimal rights vector

The so-called remainder $R(C, j)$ of player j in coalition $C \in \mathcal{P}(N)$ is the amount which remains for player j if the coalition C forms and the rest of the players in coalition C all obtain their individual utopia payoffs, i.e.

$$R(C, j) = v(C) - \sum_{k \in C, k \neq j} M_k.$$

We can define a vector of **minimal rights** with components

$$m_j = \max_{C: j \in C} R(C, j), \quad \text{for} \quad j = 1, \ldots, n,$$

since player j has a justification to demand at least m_j in the grand coalition, see e.g. the book by Branzei, Dimitrov and Tijs [15], p. 20.
The package `CoopGame` provides a function `getMinimalRights` for computing a vector of minimal rights for every player:

```
library(CoopGame)
v <- c(2,3,4,8,9,10,13)
# Compute minimal rights vector for specified game v
(m <- getMinimalRights(v))
## [1] 4 5 6
```

1.2.7 The excess coefficients

The excess $e(C, x)$ of a coalition C with respect to a vector x measures the gain or loss of the members of C in case they decide to abandon the grand coalition N in favour of their own coalition C, see e.g. the book by Driessen [14], p. 12:

$$e(C, x) = v(C) - \sum_{j \in C} x_j = v(C) - x(C)$$

In the above formula we use the shorthand notation $x(C) = \sum_{j \in C} x_j$.

The package `CoopGame` provides a function `getExcessCoefficients` for computing a vector of excess coefficients for every coalition:

```
library(CoopGame)
A <-c(3,4,5,9,10,11,18)
x <-c(5,6,7)
# Compute vector of excess coefficients for specified game v
(ec <- getExcessCoefficients(A,x))
## [1] -2 -2 -2 -2 -2 -2  0
```

Note that the last component of a vector of excess coefficients is always 0 as long as x is efficient, i.e. $\sum_{j \in C} x_j = v(N)$.

The concept of excesses is important in various solution concepts, like e.g. the nucleolus (see e.g. the original paper by Schmeidler [25] which appeared in 1969). Computing a vector of excesses comes in handy for checking the correctness of nucleolus or prenucleolus computations, see the article by Guajardo and Jörnsten [26].

1.2.8 The gap function

The gap function is the additive inverse of the vector of excesses with respect to the utopia payoff vector, see the book by Driessen [14], p. 57, for more details.

The package `CoopGame` provides a function `getGapFunctionCoefficients` for computing the vector of gap function coefficients for every coalition:

```
library(CoopGame)
A <-c(3,4,5,9,10,11,18)
# Compute vector of gap function coefficients for specified game v
(gap <- getGapFunctionCoefficients(A))
## [1] 4 4 4 6 6 6 6
```

1.2.9 The per capita excess coefficients

Per capita excess coefficients replace excess coefficients in the computation of the per capita nucleolus (see the original paper by Young [27]). The function `getPerCapitaExcessCoefficients` computes a vector of per capita excess coefficients for every coalition and a vector x, i.e. the excess coefficients are divided by the number of players in each coalition:

```
library(CoopGame)
A <-c(3,4,5,9,10,11,18)
x <-c(5,6,7)
# Compute vector of per capita excess coefficients for specified game v
(ecpc <- getPerCapitaExcessCoefficients(A,x))
## [1] -2 -2 -2 -1 -1 -1  0
```

Computing a vector of per capita excess coefficients comes in handy for checking the correctness of computations of the per capita nucleolus, see [26].

1.2.10 Propensities to disrupt

For a cooperative game v and a payoff vector x with $\sum_{j=1}^{n} x_j = v(N)$ player i's propensity to disrupt (see e.g. the article by Littlechild and Vaidya [28]) is defined as

$$d(i,x) = \frac{\sum_{j=1, j \neq i}^{n} x_j - v(N \backslash \{i\})}{x_i - v_i}$$

The above expression quantifies the disruption caused if player i breaks away from the grand coalition. Within this expression the denominator stands for the loss incurred by player i for breaking away from the grand coalition, whereas the numerator stands for the joint loss of the rest of the players due to the breakup caused by player i. This concept is important in solution concepts like the Gately point (see e.g. the original paper by Gately [29] from 1974, the paper by Littlechild and Vaidya [28] or the recent article [30] by the authors) and the disruption nucleolus (see [28]).

The function `getVectorOfPropensitiesToDisrupt` computes a vector of propensities to disrupt for every coalition and a vector x:

```
library(CoopGame)
A <-c(3,4,5,9,10,11,18)
x <-c(5,6,7)
# Compute vector of propensities to disrupt for specified game v
(propVec <- getVectorOfPropensitiesToDisrupt(A,x))
## [1] 1 1 1 1 1 1 0
```

Note that the last component of a vector of propensities to disrupt is always set to 0. Computing a vector of coefficients of propensities to disrupt comes in handy for checking the correctness of computations of the disruption nucleolus, see [26].

1.2.11 The equal propensity to disrupt

`CoopGame` also provides a function for computing the so-called equal propensity to disrupt. This concept originates from the paper [29] by Gately for three-person games and was generalized to n-person games by Littlechild and Vaidya (see [28], p. 152). The goal is to find an imputation x with minimal propensity to disrupt. It can be shown that this minimal propensity to disrupt can be found by equating the propensity to disrupt over all players, i.e.

$$d(i,x) = d^* \quad \text{for} \quad i = 1,\ldots,n.$$

As pointed out by Littlechild and Vaidya (see [28], p. 153) using the above approach one can easily find the following closed-form expression

$$d^* = \frac{(n-1)v(N) - \sum_{j=1}^{n} v(N\backslash\{j\})}{v(N) - \sum_{j=1}^{n} v_j} = \frac{\sum_{j=1}^{n} M_j - v(N)}{v(N) - \sum_{j=1}^{n} v_j}$$

for the equal propensity to disrupt d^* of a TU game with n players. Our recent paper [30] provides some insight and interpretation for the cases $d^* = 0$ and $d^* < 0$.

1.2.12 Minimum winning coalitions and real gaining coalitions

In a simple game v we call a player j critical for a coalition C if the departure of player j turns C from a winning coalition into a losing coalition, i.e. $v(C) = 1$ and $v(C\backslash j) = 0$. A minimum winning coalition in a simple game is a coalition where every member of the coalition is critical, see e.g. the paper by Deegan and Packel [31], the paper by Holler [32] or the article by Bertini [33]. The function `getMinimumWinningCoalitions` identifies all minimal winning coalitions in a simple game v and returns a corresponding data frame. The function `getCriticalCoalitionsOfPlayer` identifies all coalitions for which a given player is critical.

```
library(CoopGame)
# Define a simple game
A <-c(0,0,0,1,1,0,1)
# Find the minimum winning coalitions
getMinimumWinningCoalitions(A)
##   V1 V2 V3 cVal
## 4  1  1  0    1
## 5  1  0  1    1
# Find the coalitions where player 2 is critical
getCriticalCoalitionsOfPlayer(2,A)
```

```
##   V1 V2 V3 cVal bmRow
## 4  1  1  0    1     4
```

The concept of minimum winning coalitions can be generalized to general cooperative games v via the concept of real gaining coalitions (see e.g. the original paper by Holler and Li [34] or the article by Bertini and Stach [35]). C is called a real gaining coalition (RGC) iff for any true subset $T \subset C$ there holds $v(C) - v(T) > 0$. We provide a corresponding function `getRealGainingCoalitions`.

```
library(CoopGame)
A <-c(0,0,0,0.8,0.9,0,0.9)
# Find the real gaining coalitions
getRealGainingCoalitions(A)
##   V1 V2 V3 cVal
## 4  1  1  0  0.8
## 5  1  0  1  0.9
```

We provide corresponding functions `getWinningCoalitions` and `getGainingCoalitions` for computing the winning coalitions in simple games (see e.g. the original paper by Bertini, Gambarelli and Stach [36]) and the gaining coalitions in general TU games (see e.g. the original paper by Bertini and Stach [35] from 2015), respectively.

1.2.13 The unanimity coefficients

The unanimity coefficients represent a cooperative game in an alternative basis, the so-called unanimity basis. They were introduced in the seminal paper by Shapley [37] and are also called Harsanyi dividends (see also the books by Peleg and Sudhölter [3], p. 153, or by Gilles [17], pp. 15–17, for more details).

Our package provides a function `getUnanimityCoefficients`.

```
library(CoopGame)
# The Maschler game
Maschler <-c(0,0,0,60,60,60,72)
# Find the unanimity coefficients for the Maschler game
(unCoeff <- getUnanimityCoefficients(Maschler))
## [1]    0    0    0   60   60   60 -108
```

1.2.14 The k-cover

In [14], p. 173, Driessen defines an associated cover v_k that majorizes the original game v for a given integer k.

$$v_k(C) = \begin{cases} v(C) & \text{if} \quad |C| < k, \\ v(N) - \sum_{j \in N \setminus C} M_j & \text{else,} \end{cases}$$

with M denoting the utopia payoff vector. In case the gap function g of original game v satisfies

$$0 \leq g(N) \leq g(C)$$

for all coalitions $C \subseteq N$ with k or more elements, then v_k is called the k-cover of the game v. In `CoopGame` we provide a function `getkCover` computing v_k according to the previous formula. In case v does satisfy the above condition, `getkCover` will return `NULL`.

```
library(CoopGame)
# Example from book by T. Driessen, p. 75
```

```r
A=c(0,0,0,9,9,15,18)
# Compute 1-cover of this 1-convex game
(A1 <- getkCover(A,k=1))
## [1]  0  6  6  9  9 15 18
```

1.3 Cooperative game theory in the R Ecosystem

There already exist two small R packages offering functionality for cooperative game theory on CRAN, i.e. `GameTheoryAllocation` [38] and `GameTheory` [39]. Both packages are very limited in scope, e.g. according to its documentation [39] the package `GameTheory` only computes the nucleolus for a maximum of four players. However, `GameTheory` provides some very nice insight into conflicting claims problems and is also employed in the R package `coopProductGame` [40] for computing linear production games (see the original paper by Owen [41] from 1975 for more details on linear production games).

Still, we found it very sensible to develop `CoopGame`. Our goal was to provide a comprehensive package for cooperative game theory that goes beyond providing routines for the Shapley value or the nucleolus. One of our goals was to provide reference implementations for lesser-known and lesser-used solution concepts. Another was to be able to visualize solution concepts in the case of three or four players.

1.4 A few general remarks on CoopGame

We wish to end this first chapter of our vignette with four general remarks on `CoopGame`.

1.4.1 Package maintainer

This is supposed to be the second R package on CRAN maintained by the first author after the publication of the significantly smaller package `EvolutionaryGames` [42] in November 2017. Please feel free to email questions regarding `CoopGame` to `jochen.staudacher@hs-kempten.de`.

1.4.2 No formal maximum number of players

`CoopGame` does not enforce a maximum number of players. Still, realistically for most concepts provided in `CoopGame` users should limit their studies to at most 20 players if they expect quick answers on modern computers. The authors have employed `CoopGame` for up to $n = 24$ players, but R normally ran out of storage for $n > 24$.

1.4.3 Minimum: 2 Players

In `CoopGame` a minimum of two players is needed in order to define a TU game.

1.4.4 No null games allowed

In case the user specifies a null game, i.e. a trivial TU game with each coalition generating zero value, `CoopGame` will stop with an error message.

2 Cooperative Games with Special Structure

For the convenience of the users we provide the possibility to generate special families of games. Our approach is to generate the game function as a list. The corresponding

game vector can either be generated directly or extracted as the corresponding element of the list. Still, for each special family of cooperative game there is

- a function for the game object (as a list)
- a function for the corresponding game vector and
- a function for the value of a coalition specified by the user.

This chapter gives an overview of the special families of games available in `CoopGame` and their usage.

2.1 Bankruptcy games

Bankruptcy games are studied frequently in game cooperative game theory, see e.g. the book by Gura and Maschler [43], the seminal paper by Aumann and Maschler [44] from 1985, the article by Aumann [45] from 2002 or the original paper by O'Neill [46] on a problem of rights arbitration from the Talmud from 1982.

Imagine a person dies leaving debts $d_1, \ldots, d_n$ to n creditors. However, the sum $\sum_{i=1}^{n} d_i$ of the debts is greater than the value of the estate E of the deceased. Now we face the problem that the debts are mutually inconsistent as the estate is too small in order to meet all of the debts of the n creditors. The game theoretic approach to bankruptcy problems started in 1982 with the article by O'Neill [46] where O'Neill defines a TU game v for a set of creditors $N = \{1, \ldots, n\}$, a debt vector d of lenght n and an estate E as

$$v(C) = \max(0, E - \sum_{i \in N \setminus C} d_i)$$

for any coalition C of creditors.

`CoopGame` provides the function `bankruptcyGame`. It creates a list containing all information about a bankruptcy game specified by the user, i.e. the list contains the number of players n, the vector of claims d, the estate E, plus the bankruptcy game vector v. Employing the function `bankruptcyGameVector` users can alternatively generate the corresponding game vector directly specifying the number n of players, the value of the estate E and a vector of claims d of length n. `CoopGame` will check whether the specification of the bankruptcy game is consistent in the sense that $E \leq \sum_{i \in N} d_i$.

We look into the usage of `bankruptcyGame` and `bankruptcyGameVector` by studying an important example from the Babylonian Talmud frequently employed in the game theory literature.

2.1.1 A problem from the Babylonian Talmud

This important example from the Babylonian Talmud deals with a man who married three women. In their marriage contracts the three women were promised the sums of 100, 200 and 300 units of money after the death of their husband. The man dies and his estate amounts to less than 600 units. In the following we will not study this important example in more detail, but simply focus on an estate E worth 300 units of money.

`CoopGame` allows us to create the corresponding bankruptcy game.

```
library(CoopGame)
bankruptcyGame(n=3,d=c(100,200,300),E=300)
## $n
## [1] 3
##
```

```
## $d
## [1] 100 200 300
##
## $E
## [1] 300
##
## $v
## [1]   0   0   0   0 100 200 300
```

As stated we provide the user with various ways to extract the corresponding game vector. One is to use `bankruptcyGameVector`, another to access the element `$v` from the `bankruptcyGame` object.

```
library(CoopGame)
# First approach
bankruptcyGameVector(n=3,d=c(100,200,300),E=300)
## [1]   0   0   0   0 100 200 300
#
# Alternative approach
bankruptcyGame(n=3,d=c(100,200,300),E=300)$v
## [1]   0   0   0   0 100 200 300
```

In addition, the user can also compute the value of any individual coalition using the function `bankruptcyGameValue`. For example, let us extract the value of the coalition of the second and third widow:

```
library(CoopGame)
bankruptcyGameValue(S=c(2,3),d=c(100,200,300),E=300)
## [1] 200
```

We will briefly revisit bankruptcy games in section 5.3.2 of this vignette. For more information on bankruptcy games and the contested-garment principle (as well as it physical interpretation via hydraulic rationing) we refer the reader to the book by Gura and Maschler [43], the paper by Aumann and Maschler [44] from 1985, the paper by Aumann [45] from 2002 and the original paper by O'Neill [46] from 1982.

2.2 Cost allocation games

We can look at cost allocation games in characteristic function form consisting of the set $N = \{1, \ldots, n\}$ of agents (or purposes, projects or services) and the characteristic function $c : \mathcal{P}(N) \to \mathbb{R}$ with $c(\emptyset) = 0$. We are introducing the shorthand notation

$$c_i = c(\{i\}) \quad \text{for} \quad i = 1, \ldots, n,$$

for the costs of single agents. The connection between cost games and TU games is given by the associated savings game v for $N = \{1, \ldots, n\}$ defined by

$$v(C) = \sum_{i \in C} c_i - c(C)$$

for every coalition C. Note that the associated savings game v is automatically 0-normalized. For more information on cost games see e.g. the article by Otten [47], the article by Young [48], the overview paper by Parrachino, Zara and Patrone [49], the paper by Tijs and Driessen [50] and the paper by Straffin and Heaney [51].

CoopGame will perform computations on the corresponding cost-savings game. Note that if x_i, $i = 1, \ldots, n$, is the solution of the cost-savings game provided by CoopGame then the actual costs for player i will be $y_i = c_i - x_i$.

CoopGame provides the function costSharingGame. The user needs to specify the number of players n and a vector of costs of length n. We look into the usage of costSharingGame by briefly studying an important example from the literature.

2.2.1 The TVA problem

The TVA (Tennessee Valley Authority) problem is a classic example of a cost-sharing problem. We refer to the original article by Ransmeier [52] on the Tennessee Valley Authority from 1942 and the book by Driessen [14], pp. 1–3, for more details and the history of this problem.

The following code example shows how to specify the TVA problem in CoopGame:

```
library(CoopGame)
TVACosts=c(163520,140826,250096,301607,378821,367370,412584)
(tvaCostGame <- costSharingGame(n=3, TVACosts))
## $n
## [1] 3
##
## $Costs
## [1] 163520 140826 250096 301607 378821 367370 412584
##
## $v
## [1] 0 0 0 2739 34795 23552 141858
#
# Alternatively, generate and output only the corresponding game vector
(v <- costSharingGameVector(n=3, TVACosts))
## [1] 0 0 0 2739 34795 23552 141858
```

We will revisit the TVA problem in section 5.3.3 of this vignette.

2.3 Glove games

In glove games we have a set $N = \{1, \ldots, n\}$ of n players and a disjoint union $N = L \cup R$. L is the set of players owning one left-hand glove each and R is the set of players owning one right-hand glove each. The worth of a coalition C equals the number of pairs of gloves that the members of C can provide. In short: For every coalition C there holds: $v(C) = \min(|S \cap L|, |S \cap R|)$. For more details on glove games we refer to the book by Peters [19], p. 155–156.

CoopGame provides the function gloveGame. The user needs to specify the number of players n, the set L of players owning one left-hand glove each and the set R of players owning one right-hand glove each.

In the following example we compute the game vector for a glove game with three players and players 1 and 3 owning one left-hand glove each whereas player 2 owns the only right-hand glove.

```
library(CoopGame)
gloveGame(n=3,L=c(1,3),R=2)$v
## [1] 0 0 0 1 0 1 1
#
```

```
# Equivalent alternative approach
gloveGameVector(n=3,L=c(1,3),R=2)
## [1] 0 0 0 1 0 1 1
```

2.4 Cardinality games

For a cardinality game the value of each coalition is simply the number of the members of the coalition, i.e. $v(C) = |C|$ for every coalition C. In our opinion cardinality games make very good test cases.

`CoopGame` provides the function `cardinalityGame`. The user only needs to specify the number of players n in the cardinality game.

```
library(CoopGame)
cardinalityGame(4)$v
##  [1] 1 1 1 1 2 2 2 2 2 2 3 3 3 3 4
#
# Equivalent alternative approach
cardinalityGameVector(4)
##  [1] 1 1 1 1 2 2 2 2 2 2 3 3 3 3 4
```

2.5 Weighted voting games

We now look into decision-making and voting in committees and define a so-called weighted majority game (aka quota game, aka weighted voting game) for n players as follows. Each player j is assigned a weight w_j (which in some situations we may also interpret the number of votes of a group). A law or motion gets passed in the committee if the quota q is reached or exceeded, i.e.

$$v(C) = \begin{cases} 1 & \text{if} \quad \sum_{j \in C} w_j \geq q \\ 0 & \text{else.} \end{cases}$$

We refer to the book by Maschler, Solan and Zamir [18], pp. 825 – 831, for more details and examples.

By definition weighted voting games are always simple games. Along the lines of the book by Peleg and Sudhölter [3], p. 16, we call a TU game v **simple** if v is monotonic and the values of all coalitions are 0 or 1, see also subsection 3.2.5 of this vignette.

`CoopGame` provides the function `weightedVotingGame`. The user needs to specify the number of players n, a numeric vector of weights w of length n with the weights of the individual players and the quota q in the weighted voting game. We provide a simple example with the three players.

```
library(CoopGame)
weightedVotingGame(n=3, w=c(3,5,4), q=8)$v
## [1] 0 0 0 1 0 1 1
#
# Equivalent alternative approach
weightedVotingGameVector(n=3, w=c(3,5,4), q=8)
## [1] 0 0 0 1 0 1 1
```

For a deeper analysis and a broader view on voting in committees we recommended the article by Peleg [53] from 2002.

2.6 Weighted majority games with a single veto player

Weighted majority games with a single veto player are a special case of quota games. We take our definition of a weighted majority game with a single veto player game from the book by Matthew O. Jackson [54], p. 415. The only winning coalitions are those containing the veto player i and at least one other player.

$$v(C) = \begin{cases} 1 & \text{if} \quad |C| \geq 2 \quad \text{and} \quad i \in C \\ 0 & \text{else.} \end{cases}$$

`CoopGame` provides the function `majoritySingleVetoGame`. The user needs to specify the number of players n and the number of the veto player in the weighted majority game.

The following example produces the game vector for a majority game with four players and player 2 in the role of the veto player.

```
library(CoopGame)
majoritySingleVetoGame(n=4, vetoPlayer=2)$v
##  [1] 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1
#
# Equivalent alternative approach
majoritySingleVetoGameVector(n=4, vetoPlayer=2)
##  [1] 0 0 0 0 1 0 0 1 1 0 1 1 0 1 1
```

2.7 Unanimity games

Unanimity games are a special family of simple games. For a unanimity game the winning coalitions are exactly those coalitions containing the set $T \in \mathcal{P}(N)$, i.e.

$$v(C) = \begin{cases} 1 & \text{if} \quad T \subseteq C \\ 0 & \text{else,} \end{cases}$$

see Peleg and Sudhölter [3], p. 152. We can understand T as the powerful (or productive) lot among the players. `CoopGame` provides the function `unanimityGame`. The user needs to specify the number of players n and the set of powerful players T in the unanimity game.

The following example produces the game vector for a unanimity game with four players and T consisting of players 1 and 4.

```
library(CoopGame)
unanimityGame(n=4, T=c(1,4))$v
##  [1] 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1
#
# Equivalent alternative approach
unanimityGameVector(n=4, T=c(1,4))
##  [1] 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1
```

2.8 Apex games

Apex games are a special family of simple games. For an apex game there are two types of winning coalitions: The first type contains the so-called apex player plus at the least one other player. The only winning coalition without the apex player is a coalition of all the other players except the apex player, see in particular the book by Peters [19], p. 164, for more details on apex games. `CoopGame` provides the function `apexGame`. The user needs to specify the number of players n and the number of the apex player in the apex game. The following lines construct the game vector for an apex game with four players and player 2 in the role of the apex player.

```
library(CoopGame)
apexGame(n=4, apexPlayer=2)$v
##  [1] 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
#
# Equivalent alternative approach
apexGameVector(n=4, apexPlayer=2)
##  [1] 0 0 0 0 1 0 0 1 1 0 1 1 1 1 1
```

2.9 Dictator games

Dictator games are a special case of unanimity games, i.e. the set T in the unanimity game consists of a single player i, the dictator.

$$v(C) = \begin{cases} 1 & \text{if} \quad i \in C \\ 0 & \text{else.} \end{cases}$$

For more details on dictator games we refer to the book by Peters [19], p. 295, and the book by Maschler, Solan and Zamir [18], p. 764. `CoopGame` provides the function `dictatorGame`. The user needs to specify the number of players n and the number of the dictator in the dictator game. The following lines generate the game vector for a dictator game with four players and player 3 in the role of the dictator.

```
library(CoopGame)
dictatorGame(n=4, dictator=3)$v
##  [1] 0 0 1 0 0 1 0 1 0 1 1 0 1 1 1
#
# Equivalent alternative approach
dictatorGameVector(n=4, dictator=3)
##  [1] 0 0 1 0 0 1 0 1 0 1 1 0 1 1 1
```

2.10 Divide-the-dollar games

The divide-the-dollar games in `CoopGame` are a special family of simple games. We take our definition of a divide-the-dollar game from the book by Matthew O. Jackson [54], p. 413. For a divide-the-dollar game with n players the winning coalitions are exactly those coalitions containing at least $n/2$ players, i.e.

$$v(C) = \begin{cases} 1 & \text{if} \quad |C| \geq n/2 \\ 0 & \text{else.} \end{cases}$$

`CoopGame` provides the function `divideTheDollarGame`. The user only needs to specify the number of players n in the divide-the-dollar game. We end this chapter with the game vector for a divide-the-dollar game with three players.

```
library(CoopGame)
divideTheDollarGame(3)$v
## [1] 0 0 0 1 1 1 1
#
# Equivalent alternative approach
divideTheDollarGameVector(3)
## [1] 0 0 0 1 1 1 1
```

3 Game Properties

The package `CoopGame` provides users with the possibility to check a given game vector v for a number of different game properties. In this chapter of our vignette we would like to give a quick overview of these game properties and the precise definitions we use.

3.1 Checking a game property

Given that the 17 different functions for checking various game properties all come with detailed examples in their individual documentations we will only specify the basic principle for checking a game property. Users simply need to specify a game vector and then check the corresponding game property (which always starts with `is`). The result will be `TRUE` if the game shares the property in question, else it will be `FALSE`.

```
library(CoopGame)
A <-c(0,0,0,1,1,0,1)
isSuperadditiveGame(A)
## [1] TRUE
```

3.2 A quick overview of available game properties

3.2.1 Nonnegative games

We call a game vector A **nonnegative** if all of its entries are nonnegative. This game property can be checked via the function `isNonnegativeGame`. We provide certain solution concepts for nonnegative games only.

3.2.2 Essential games

We call a TU game v **essential**, if the value of the grand coalition $v(N)$ is greater than the sum of the values of the singleton coalitions $v(\{i\})$, i.e.

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

Our definition follows the books by Chakravarty, Mitra and Sarkar [16], p. 23, and by Gilles [17], p. 18. We find it very convenient that according to this definition the imputation set of an essential game is nonempty and consists of more than one point. This game property can be checked via the function `isEssentialGame`.

3.2.3 Degenerate games

We call a TU game v **degenerate** (or **inessential**), if the value of the grand coalition $v(N)$ equals the sum of the values of the singleton coalitions $v(\{i\})$, i.e.

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

We find it very convenient that according to this definition the imputation set of a degenerate game consists of exactly one point (specified by the singleton coalitions). This game property can be checked via the function `isDegenerateGame`.

3.2.4 Monotonic games

According to the book by Peleg and Sudhölter [3], p. 12, we call a TU game v **monotonic** if

$$S \subseteq T \subseteq N \Rightarrow v(S) \leq v(T).$$

This game property can be checked via the function `isMonotonicGame`.

3.2.5 Simple games

Along the lines of the book by Peleg and Sudhölter [3], p. 16, we call a TU game v **simple** if v is monotonic and the values of all coalitions are 0 or 1. This game property can be checked via the function `isSimpleGame`.

3.2.6 Symmetric games

Following the book by Peleg and Sudhölter [3], p. 12, we call a TU game v **symmetric** if the values of all coalitions containing the same number of players are identical, i.e.

$$|S| = |T| \Rightarrow v(S) = v(T)$$

for all coalitions $S, T \subseteq N$. This game property can be checked via the function `isSymmetricGame`.

3.2.7 Constant-sum games

In a **constant-sum** game v for any coalition S the sums of $v(S)$ and its complement $v(N \backslash S)$ equal $v(N)$, i.e.

$$v(S) + v(N \backslash S) = v(N)$$

for all $S \subseteq N$, see e.g. the book by Peleg and Sudhölter [3], p. 11. This game property can be checked via the function `isConstantSumGame`.

3.2.8 Weakly constant-sum games

We call a TU game v with n players **weakly constant-sum** as long as the constant-sum condition holds for coalitions of sizes 1 and $n-1$, i.e.

$$v(\{i\}) + v(N \backslash \{i\}) = v(N)$$

for all players $i = 1, \ldots, n$. In other words: For weakly constant-sum games the vector of singleton coalitions and the utopia payoff vector coincide. This game property comes in handy when checking uniqueness conditions for the Gately point, see the paper [30] by the authors of this vignette. Also note that any constant-sum game is weakly constant-sum, but not vice versa. This game property can be checked via the function `isWeaklyConstantSumGame`.

3.2.9 Superadditive games

We call a TU game v **superadditive** if

$$v(S \cup T) \geq v(S) + v(T)$$

for all coalitions $S, T \subseteq N$ with $S \cap T = \emptyset$, see e.g. the book by Peleg and Sudhölter [3], p. 10, or the book by Maschler, Solan and Zamir [18], p. 671. The idea of superadditivity is that disjoint groups of players are never punished for cooperating. This game property can be checked via the function `isSuperadditiveGame`.

3.2.10 Additive games

We call a TU game v **additive** if

$$v(S \cup T) = v(S) + v(T)$$

for all coalitions $S, T \subseteq N$ with $S \cap T = \emptyset$, see e.g. the book by Peleg and Sudhölter [3], p. 11, or the book by Maschler, Solan and Zamir [18], p. 792. Note that an additive game is always superadditive, constant-sum, weakly constant-sum and degenerate whereas none of these four game properties guarantee additivity in return. This game property can be checked via the function `isAdditiveGame`.

3.2.11 Weakly superadditive games

We call a TU game v with n players **weakly superadditive** if

$$v(S \cup \{i\}) \geq v(S) + v(\{i\})$$

for all coalitions $S \subseteq N$ and all players $i = 1, \ldots, n$ with $i \notin S$, see e.g. the book by Peleg and Sudhölter [3], p. 10. Note that weak superadditivity is equivalent to 0-monotonicity, i.e. the zero-normalization of the game is monotonic, see e.g. Maschler, Solan and Zamir [18], p. 789, or Peleg and Sudhölter [3], p. 12. This game property can be checked via the function `isWeaklySuperadditiveGame`.

3.2.12 Quasi-balanced games

Given a TU game v with n players let m denote the minimal rights vector and M the utopia payoff vector. We call the TU game v **quasi-balanced** if $m(i) \leq M(i)$ for all $i = 1, \ldots, n$ and

$$\sum_{i=1}^{n} m_i \leq v(N) \leq \sum_{i=1}^{n} M_i,$$

see e.g. the book by Branzei, Dimitrov and Tijs [15], p. 31. Quasi-balanced games can equivalently be characterized as the TU games with a non-empty core cover. Note also that the τ-value, an important solution concept, is only defined for quasi-balanced games, see e.g. the original paper by Tijs [55] from 1981. This game property can be checked via the function `isQuasiBalancedGame`.

3.2.13 Balanced games

A TU game v is called **balanced** if and only if the core of v is non-empty. For the sake of brevity we do not go into any details of the balancedness condition and the Bondareva-Shapley theorem in this vignette. Instead, we only refer to the original papers

by Bondareva [56] and by Shapley [57] as well as to section 3.1 of the book by Peleg and Sudhölter [3] for further details. This game property can be checked via the function `isBalancedGame`.

3.2.14 Convex games

We call a TU game v **convex** if

$$v(S \cup T) + v(S \cap T) \geq v(S) + v(T)$$

for all $S, T \subseteq N$, see e.g. the book by Peleg and Sudhölter [3], p. 10, for more details. Convex games are always balanced, i.e. the core of a convex game is never empty, and they arise in various important application areas of cooperative game theory. This game property can be checked via the function `isConvexGame`.

3.2.15 Semiconvex games

A TU game v with n players is called **semiconvex** if for its gap function g there holds

$$0 \leq g(i) \leq g(S)$$

for all players $i = 1, \ldots, n$ and all coalitions $S \subseteq N$ with $i \in S$. Note that convex games are always semiconvex but not vice versa. We refer to the book by Driessen [14], p. 76, or the original paper by Driessen and Tijs [58] for more details. This game property can be checked via the function `isSemiConvexGame`.

3.2.16 1-convex games

A TU game v with n players is called **1-convex** if for its gap function g there holds

$$0 \leq g(N) \leq g(S)$$

for all coalitions $S \subseteq N$ with $S \neq \emptyset$. Note that the 1-cover v_1 of a 1-convex game v will always be convex. We refer to the book by Driessen [14], p. 73, for more details. This game property can be checked via the function `is1ConvexGame`.

3.2.17 k-convex games

k-convexity can be regarded as a generalization of 1-convexity. A TU game v with n players is called **k-convex** if and only if its k-cover (see 1.2.14 of this vignette) exists and is convex. We refer to section 7.1 of the book by Driessen [14] for more details on k-convex games. This game property can be checked via the function `iskConvexGame` by specifying both a game vector v and an integer k. We end this chapter with a small example inspired by the book by Driessen [14], p. 175:

```
# The following game is 2-convex
library(CoopGame)
alpha = 0.48
v=c(0,0,0,alpha,alpha,0,1)
iskConvexGame(v,2)
## [1] TRUE
# The following game is not 2-convex
library(CoopGame)
alpha = 0.52
v=c(0,0,0,alpha,alpha,0,1)
```

```
iskConvexGame(v,2)
## [1] FALSE
```

4 Set Solution Concepts and Allocation Properties

The package `CoopGame` provides the following five set-based solution concepts:

- the imputation set
- the core
- the core cover
- the reasonable set
- the Weber Set

For all five set-based solution concepts we provide

- a routine for computing the vertices of the corresponding set for an arbitrary number $n \geq 2$ of players
- a routine for checking whether an allocation x is part of the corresponding set
- a routine for drawing the corresponding set for 3 or 4 players (see chapter 6 of this vignette)

Note that both for checking whether an allocation x belongs to a set solution concept or not and for drawing the set solution (in the case of 3 or 4 players) the corresponding vertices are internally always computed first.

4.1 Available set solution concepts

4.1.1 The imputation set

Given a TU game v with n players we call a vector $x \in \mathbb{R}^n$ **efficient** if

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

Frequently the set of efficient vectors is also called the set of preimputations, see e.g. the book by Peleg and Sudhölter [3], p. 20. In case $x \in \mathbb{R}^n$ is also **individually rational** (from the point of view of every player i), i.e.

$$v(\{i\}) \leq x_i$$

for all $i = 1, \ldots, n$, then we call x an imputation. Formally, we can specify the so-called imputation set $I(v)$ as

$$I(v) = \{x \in \mathbb{R}^n | \sum_{i=1}^{n} x_i = v(N) \wedge v(\{i\}) \leq x_i\}$$

According to our game property definitions from section 3 the imputation set is empty unless v is either essential or degenerate. In the latter case $I(v)$ consists of a single point. For further details on imputations see e.g. the book by Peleg and Sudhölter [3], p.20, the book by Maschler, Solan and Zamir [18], p. 674–677, or the book by Osborne and Rubinstein [20], p. 278.

In order to computate the vertices of the imputation set we provide the function `imputationsetVertices`. The rows of the matrix returned are the vertices of the imputation set. In case the imputation set is empty an empty matrix is returned. In order to check whether an allocation x is an imputation there is the function `belongsToImputationset`.

```
library(CoopGame)
v <-c(0,0,0,1,1,0,1)
imputationsetVertices(v)
##      [,1] [,2] [,3]
## [1,]    0    0    1
## [2,]    0    1    0
## [3,]    1    0    0
belongsToImputationset(c(0.7,0.3,0),v)
## [1] TRUE
```

4.1.2 The core

Apart from the set of imputations the most prominent set solution concept is the **core**. The concept was established in the Ph.D. thesis by Gillies [59] from 1953 and a seminal article by Aumann [60] from 1961. The idea of the core is to consider only those imputations x which are also **coalitionally rational**, i.e.

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

for all nonempty coalitions $S \subseteq N$. More information on the core can e.g. be found in the book by Maschler, Solan and Zamir [18], pp. 686–747, the book by Osborne and Rubinstein [20], pp. 257–275, or the book by Peleg and Sudhölter [3], pp. 27–49.

The function `coreVertices` lists the vertices of the core as rows of a matrix. In case the core is empty an empty matrix is returned. The function `belongsToCore` checks whether an allocation x is contained in the core or not.

```
library(CoopGame)
v <-c(0,0,0,1,1,0,3)
coreVertices(v)
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]    1    0    2
## [3,]    0    1    2
## [4,]    0    2    1
## [5,]    1    2    0
belongsToCore(c(1.7,1.3,0),v)
## [1] TRUE
```

4.1.3 The core cover

The core cover was originally suggested by Tijs and Lipperts in [61] as a core catcher. Given a TU game v with n players let m denote the minimal rights vector and M the utopia payoff vector. The core cover of the game v consists of all imputations x satisfying

$$m(i) \leq x_i \leq M(i)$$

for all $i = 1, \ldots, n$. One can show that the core is always a subset of the core cover, see e.g. the book by Branzei, Dimitrov and Tijs [15], p. 21, or the book by Chakravarty, Mitra and Sarkar [16], pp. 45–46, for more details. Also note that the core cover is nonempty iff the game v is quasi-balanced (see chapter 3 of this vignette).

We provide functions `coreCoreVertices` listing the vertices of the core cover as rows of a matrix and `belongsToCoreCover` for checking whether an allocation x is contained in the core cover or not.

4.1.4 The reasonable set

The reasonable set was originally suggested by Milnor in [62] as another core catcher. The reasonable set of a TU game v consists of all imputations x satisfying

$$v(\{i\}) \leq x_i \leq \max_{i:i\in S}(v(S) - v(S\backslash\{i\})$$

for all $i = 1, \ldots, n$. One can show that the core cover is always a subset of the reasonable set.
We refer to the book by Branzei, Dimitrov and Tijs [15], p. 21, the book by Chakravarty, Mitra and Sarkar [16], pp. 43–46, and the article by Gerard-Varet and Zamir [63] for more details concerning the reasonable set.

We provide functions `reasonableSetVertices` listing the vertices of the reasonable set as rows of a matrix and `belongsToReasonableSet` for checking whether an allocation x is contained in the reasonable set or not.

4.1.5 The Weber Set

The Weber Set was originally suggested by Weber in [64] as another core catcher. The Weber Set is the convex hull of the marginal vectors. Thus one can easily see that the Weber Set is contained in the imputation set for superadditive games. We provide the Weber Set only for game vectors v which are both nonnegative and monotonic. One can show that the core is always a subset of the Weber Set. We refer to the book by Peters [19], chapter 18, for further details regarding the Weber Set.

We provide functions `weberSetVertices` listing the vertices of the Weber Set as rows of a matrix and `belongsToWeberSet` for checking whether an allocation x is contained in the Weber Set or not.

4.2 Using `rcdd` for computations

All five set-based solution concepts yield convex polyhedra as results. We wish to stress that we never coded any vertex computations ourselves, but relied on the available R packages `rcdd` [65] and `geometry` [66]. We wish to give particular praise to the library `cdd` for polyhedral computations by Komei Fukuda [67]. In turn, the availability of a package like `rcdd` [65] as an R-interface of the powerful library `cdd` underlines the power of the `R` ecosystem.

The imputation set, the core, the core cover and the reasonable set are by their definitions implicitly specified as so-called H-representations, i.e. intersections of halfspaces, of convex polyhedra. Computing the so-called V-representations, i.e. representing the polyhedron as the convex hull of its vertices, is done via the package `rcdd`. The Weber set is already defined as the convex hull of the marginal vectors. Hence we only need to use `rcdd` to remove any redundant points from a list of potential Weber set vertices. This feature of removing redundant points is also used to determine whether a given allocation x belongs to one of the five set solution concepts in question or not. For further information on convex polyhedra we simply refer to the books by Rockafellar [68] and Ziegler [69].

5 Point Solution Concepts

The package `CoopGame` prides itself with implementations of a very large amount of point valued solution concepts. Calling a specific point valued solution concept always

works the same way, i.e. `nameOfPointValuedSolutionConcept(v)` where v stands for the game vector of a cooperative game.

This chapter is divided into two parts: In the first part we will give an extremely brief and concise overview of the point valued solution concepts available. In the second part we will guide the readers through three examples of point solutions of TU games and another two examples of power index computations.

5.1 Overview of point solution concepts in `CoopGame`

We distinguish the point solution concepts we implemented by solution concepts for general cooperative games and power indices for simple games. We tried to document our implementations of these solution concepts in detail and give at least one reference for each individual function. Every time there is more than one reference we checked carefully that definitions were not contradictory.

5.1.1 Available point solution concepts for general cooperative games

The following 19 point solution concepts for general cooperative games are available in `CoopGame`:

- the **centroid of the core** provided by the function `centroidCore`
- the **centroid of the core cover** provided by the function `centroidCoreCover`
- the **centroid of the imputation set** provided by the function `centroidImputationSet`
- the **centroid of the reasonable set** provided by the function `centroidReasonableSet`
- the **centroid of the Weber set** provided by the function `centroidWeberSet`
- the **disruption nucleolus** (see the article by Littlechild and Vaidya [28]) provided by the function `disruptionNucleolus`
- the **Gately point** (see the original paper by Gately [29], our recent article [30] for implementation details, the article by Littlechild and Vaidya [28] or the book by Narahari [21], pp. 455–456) provided by the function `gatelyValue`
- the **modiclus** (aka modified nucleolus, see the one of the two original papers by Sudhölter [70], [71] or the book by Peleg and Sudhölter [3], pp. 147–182) provided by the function `modiclus`
- the **normalized Banzhaf value** (see e.g. one of the articles by Gambarelli [72] or by Stach [73]) provided by the function `normalizedBanzhafValue`
- the **nucleolus** (see the papers by Schmeidler [25], by Kohlberg [74], by Kopelowitz [75], by Megiddo [76] or the book by Peleg and Sudhölter [3], pp. 82–86) provided by the function `nucleolus`
- the **per capita nucleolus** (see the original paper by Young [27]) provided by the function `perCapitaNucleolus`
- the **prenucleolus** (see chapter 6 of the book by Peleg and Sudhölter [3]) provided by the function `prenucleolus`
- the **proportional nucleolus** (see the original paper by Young, Okada and Hashimoto [77]) provided by the function `proportionalNucleolus`
- the **(normalized) Public Good value** (see the original paper by Holler and Li [34]) provided by the function `publicGoodValue`
- the **(normalized) Public Help value Chi** (see the original paper by Bertini and Stach [35]) provided by the function `publicHelpChiValue`
- the **(normalized) Public Help value Theta** (see the original paper by Bertini and Stach [35]) provided by the function `publicHelpValue`
- the **Shapley value** (see the original paper by Shapley [37], the article by Bertini [78] or chapter 18 of the book by Maschler, Solan and Zamir [18]) provided by the function `shapleyValue`

- the **simplified modiclus** (see the original article by Tarashnina [79]) provided by the function `simplifiedModiclus`
- the **tau-value** (aka Tijs value, see the original paper by Tijs [55], the article by Stach [80] or the book by Branzei, Dimitrov and Tijs [15], p. 31) provided by the function `tauValue`

Note that the above 19 solution concepts are all efficient, i.e. the sum of the components of the solution equals the value of the grand coalition $v(N)$.

In addition, `CoopGame` also provides five solution concepts for general cooperative games which are not efficient:

- the **absolute Public Good value** (see the original paper by Holler and Li [34]) provided by the function `absolutePublicGoodValue`
- the **absolute Public Help value Chi** (see the original paper by Bertini and Stach [35]) provided by the function `absolutePublicHelpChiValue`
- the **absolute Public Help value Theta** (see the original paper by Bertini and Stach [35]) provided by the function `absolutePublicHelpValue`
- the **Banzhaf value** (see the book by Peters [19], pp. 367–370) provided by the function `banzhafValue`
- the **raw Banzhaf value** (see the book by Chakravarty, Mitra and Sarkar [16], pp. 118–119) provided by the function `rawBanzhafValue`

5.1.2 Available power indices for simple games

The following seven power indices for simple games are available in `CoopGame`:

- the **Deegan-Packel index** (see the original article by Deegan and Packel [31] or the German book by Holler and Illing [24], pp. 323–324) provided by the function `deeganPackelIndex`
- the **Johnston index** (see the original paper by Johnston [81] or the book by Chakravarty, Mitra and Sarkar [16], p. 124) provided by the function `johnstonIndex`
- the **normalized Banzhaf value** (see the article by Stach [82] or the book by Chakravarty, Mitra and Sarkar [16], pp. 118–119) provided by the function `normalizedBanzhafIndex`
- the **Public Good index** (see the original papers by Holler [32], by Holler and Packel
  [83] or the later article by Holler [84]) provided by the function `publicGoodIndex`
- the **Public Help index Chi** (see the original papers by Bertini and Stach [35] and by Stach [85]) provided by the function `publicHelpChiIndex`
- the **Public Help index Theta** (see the original paper by Bertini, Gambarelli and Stach [36] or one of the later articles by Bertini and Stach [35] or Stach [85]) provided by the function `publicHelpIndex`
- the **Shapley-Shubik index** (see the original paper by Shapley and Shubik [86], the article by Stach [87] from 2011 or the book by Peters [19], pp. 156–159) provided by the function `shapleyShubikIndex`

Note that the above seven power indices are all efficient, i.e. the sum of the components of the solution equals 1.

In addition, `CoopGame` also provides nine power indices for simple games which are not efficient:

- the **Barua Chakravarty Sarkar index** (see the original paper by Barua, Chakravarty and Sarkar [88] or the book by Chakravarty, Mitra and Sarkar [16],

pp. 120–123) provided by the function `baruaChakravartySarkarIndex`
- the **Coleman Power index of a Collectivity to Act** (see the original article by Coleman [89], the paper by Stach [90] or the survey article by de Keijzer [91], p. 18) provided by the function `colemanCollectivityPowerIndex`
- the **Coleman Initiative Power index** (see the original article by Coleman [89], the paper by Stach [90], the survey article by de Keijzer [91], p. 18, or the book by Chakravarty, Mitra and Sarkar [16], pp. 120–123) provided by the function `colemanInitiativePowerIndex`
- the **Coleman Preventive Power index** (see the original article by Coleman [89], the paper by Stach [90], the survey article by de Keijzer [91], p. 18, or the book by Chakravarty, Mitra and Sarkar [16], pp. 120–123) provided by the function `colemanPreventivePowerIndex`
- the **König-Bräuninger index** (see one of the original articles by König and Bräuninger [92] and by Nevison, Zicht and Schöpke [93] or the later paper by Bertini and Stach [35]) provided by the function `koenigBraeuningerIndex`
- the **Nevison index** (see the original paper by Nevison [94]) provided by the function `nevisonIndex`
- the **non-normalized Banzhaf index** (see the paper by Stach [82] from 2011 or the book by Chakravarty, Mitra and Sarkar [16], pp. 118–119) provided by the function `nonNormalizedBanzhafIndex`
- the **Rae index** (see the original article by Rae [95] or the book by Chakravarty, Mitra and Sarkar [16], p. 119–120) provided by the function `raeIndex`
- the **raw Banzhaf index** (see the book by Chakravarty, Mitra and Sarkar [16], p. 118–119) provided by the function `rawBanzhafIndex`

In the context of power indices this vignette is not the place to discuss any details concerning the concepts of I-Power and P-Power. Hence we simply wish to refer to the original paper by Felsenthal, Machover and Zwicker [96] and the book by Felsenthal and Machover [97] for details. The two small overview articles [98] and [99] by Felsenthal and Machover from 2011 may serve as a very first introduction to these concepts.

5.2 A few remarks on the implementation of the nucleolus and its derivatives

We wish to stress that in our implementations of the nucleolus and its various derivatives, i.e. the prenucleolus, the per capita nucleolus, the proportional nucleolus, the disruption nucleolus, the modiclus and the simplified modiclus, we followed the paper by Guajardo and Jörnsten [26] very closely as to avoid the common mistakes in computing the nucleolus described in this article. In particular, we made sure to incorporate the information on the dual values in the solution process as described in the paper by Guajardo and Jörnsten [26]. Our algorithmic approach for finding the nucleolus and its derivatives is nothing particularly fancy. We simply solve a series of linear programs following the technical report by Kopelowitz [75] from 1967. Our solver for the individual linear programs is the revised dual simplex solver provided by the function `lpcdd` from the package `rcdd` [65]. We finally note that an earlier yet unpublished version of `CoopGame` employed the package `glpkAPI` [100], a low-level interface to the GLPK library [101] by Makhorin, for the solution of linear programs. We refer to the M.Sc. thesis of the second author [102] from 2017 for details of this unpublished approach. Later studies of the first author showed that the revised dual simplex solver provided by the function `lpcdd` from the package `rcdd` [65] was much better suited to compute the nucleolus and its derivatives and lead to reimplementation before the first submission of `CoopGame` to CRAN.

5.3 Using the point solution concepts in `CoopGame` by example

We would now like to guide our readers through three classic examples of point solutions for cooperative games.

5.3.1 The Shapley value for the inheritance problem due to Ibn Ezra (1146)

Our first example is taken from a paper by Robert Aumann from 2010 on "Some non-superadditive games, and their Shapley values, in the Talmud" [103]. As the first in a series of problems, Aumann investigates the inheritance problem due to Ibn Ezra (1146) (see [104]), i.e. an inconsistent will, in that paper.

A man with four sons dies. He leaves an estate worth 120 units of money. He bequeathes 120 units of money to his first son, 60 units of money to his second son, 40 units of money to his third sum and 30 units of money to his youngest son. In order to analyze the problem of how to divide the estate, let us first introduce the estate $E = 120$ and the four claims $c_1 = 120$, $c_2 = 60$, $c_3 = 40$ and $c_4 = 30$. We can now define a TU game v as follows:

$$v = \begin{cases} \min(E, \max_{i \in S} c_i) & \text{if } |S| < 4, \\ E & \text{else,} \end{cases}$$

```
library(CoopGame)
Aumann2010Example<-c(120,60,40,30,120,120,120,60,60,40,120,120,120,60,120)
shapleyValue(Aumann2010Example)
## [1] 80.83333 20.83333 10.83333  7.50000
```

We confirm that the Shapley value of v coincides with Ibn Ezra's solution: The first son get $80\frac{5}{6}$, the second son $20\frac{5}{6}$, the third son $10\frac{5}{6}$ and the youngest son $7\frac{1}{2}$ units of money. For more details, we refer to the article by Aumann [103] and the 1982 paper by O'Neill [46].

5.3.2 The nucleolus for bankruptcy problems from the Babylonian Talmud

Our second set of examples revisits bankruptcy problems from the Babylonian Talmud from section 2.1, see the papers by Aumann and Maschler [44] and Aumann [45]. We solve the problem of the three widows from the Babylonian Talmud for $E = 100, 200$, and 300 using our function `nucleolus`:

```
library(CoopGame)
v100 = bankruptcyGameVector(n=3,d=c(100,200,300),E=100)
nucleolus(v100)
## [1] 33.33333 33.33333 33.33333
v200 = bankruptcyGameVector(n=3,d=c(100,200,300),E=200)
nucleolus(v200)
## [1] 50 75 75
v300 = bankruptcyGameVector(n=3,d=c(100,200,300),E=300)
nucleolus(v300)
## [1]  50 100 150
```

We confirm that the nucleolus coincides with the solutions from the Babylonian Talmud in all three cases. For more details, e.g. on the principle of equal division of the contested sum we refer to the article by Aumann and Maschler [44] and chapter 4 of the book by Gura and Maschler [43].

5.3.3 Solving the TVA problem

We investigate the classic TVA problem introduced in section 2 of this vignette (see the original article by Ransmeier [52] on the Tennessee Valley Authority from 1942 for the history of the problem and the article by Straffin and Heaney [51] and the book by Driessen [14], p. 98, for computational results).

```
library(CoopGame)
TVACosts=c(163520,140826,250096,301607,378821,367370,412584)
(v <- costSharingGameVector(n=3, TVACosts))
## [1]      0      0      0   2739  34795  23552 141858
TVACosts[1:3] - gatelyValue(v)
## [1] 117475.54  99157.29 195951.16
TVACosts[1:3] - shapleyValue(v)
## [1] 117829.0 100756.5 193998.5
TVACosts[1:3] - nucleolus(v)
## [1] 116234  93540 202810
TVACosts[1:3] - tauValue(v)
## [1] 117475.54  99157.29 195951.16
```

We confirm the results from the literature. Note that for the TVA problem the results for the tau-value and the Gately point need to coincide as the game is semiconvex, see [58].

5.4 Using the power indices in CoopGame by example

We would now like to guide our readers through two examples of applications of power indices.

5.4.1 Examples on the failure of the donation property

Bertini, Freixas, Gambarelli and Stach point out in their article [105] that the normalized Banzhaf index, the Deegan-Packel index, the Public Good index, the Johnston index and the Public Help index Theta lack the so-called donation property. We revisit the examples from table 1 on page 9 of that paper and along the way point out that the donation property also does not hold for the Public Help index Chi. We start with the 5-player game from the article by Bertini, Freixas, Gambarelli and Stach [105]:

```
library(CoopGame)
v<-weightedVotingGameVector(n=5, w=c(6,4,1,1,1), q=9)
normalizedBanzhafIndex(v)
## [1] 0.47368421 0.36842105 0.05263158 0.05263158 0.05263158
deeganPackelIndex(v)
## [1] 0.375 0.250 0.125 0.125 0.125
publicGoodIndex(v)
## [1] 0.3333333 0.1666667 0.1666667 0.1666667 0.1666667
johnstonIndex(v)
## [1] 0.52777778 0.38888889 0.02777778 0.02777778 0.02777778
publicHelpIndex(v)
## [1] 0.28125 0.25000 0.15625 0.15625 0.15625
publicHelpChiIndex(v)
## [1] 0.3234568 0.3003086 0.1254115 0.1254115 0.1254115
#
# Now player 1 donates one vote to player 2
v<-weightedVotingGameVector(n=5, w=c(5,5,1,1,1), q=9)
```

```
normalizedBanzhafIndex(v)
## [1] 0.5 0.5 0.0 0.0 0.0
deeganPackelIndex(v)
## [1] 0.5 0.5 0.0 0.0 0.0
publicGoodIndex(v)
## [1] 0.5 0.5 0.0 0.0 0.0
johnstonIndex(v)
## [1] 0.5 0.5 0.0 0.0 0.0
publicHelpIndex(v)
## [1] 0.2857143 0.2857143 0.1428571 0.1428571 0.1428571
publicHelpChiIndex(v)
## [1] 0.3309524 0.3309524 0.1126984 0.1126984 0.1126984
```

Now for the example with 10 players from the article by Bertini, Freixas, Gambarelli and Stach [105]:

```
library(CoopGame)
v<-weightedVotingGameVector(n=10, w=c(9,8,7,0,1,1,1,1,1,1), q=23)
normalizedBanzhafIndex(v)
## [1] 0.326633166 0.326633166 0.316582915 0.000000000 0.005025126
## [6] 0.005025126 0.005025126 0.005025126 0.005025126
deeganPackelIndex(v)
## [1] 0.2291667 0.2291667 0.1666667 0.0000000 0.0625000 0.0625000
## [7] 0.0625000 0.0625000 0.0625000 0.0625000
publicGoodIndex(v)
## [1] 0.18181818 0.18181818 0.09090909 0.00000000 0.09090909
## [6] 0.09090909 0.09090909 0.09090909 0.09090909
johnstonIndex(v)
## [1] 0.332692308 0.332692308 0.323076923 0.000000000 0.001923077
## [6] 0.001923077 0.001923077 0.001923077 0.001923077
publicHelpIndex(v)
## [1] 0.15312132 0.15312132 0.15076561 0.07656066 0.07773852
## [6] 0.07773852 0.07773852 0.07773852 0.07773852
publicHelpChiIndex(v)
## [1] 0.16914603 0.16914603 0.16780487 0.06991541 0.07066461
## [6] 0.07066461 0.07066461 0.07066461 0.07066461
#
# Now player 1 donates one vote to player 4
v<-weightedVotingGameVector(n=10, w=c(8,8,7,1,1,1,1,1,1,1), q=23)
normalizedBanzhafIndex(v)
## [1] 0.32908163 0.32908163 0.32397959 0.00255102 0.00255102
## [6] 0.00255102 0.00255102 0.00255102 0.00255102
deeganPackelIndex(v)
## [1] 0.22222222 0.22222222 0.16666667 0.05555556 0.05555556
## [6] 0.05555556 0.05555556 0.05555556 0.05555556
publicGoodIndex(v)
## [1] 0.16666667 0.16666667 0.08333333 0.08333333 0.08333333
## [6] 0.08333333 0.08333333 0.08333333 0.08333333
johnstonIndex(v)
## [1] 0.3329026701 0.3329026701 0.3281653747 0.0008613264 0.0008613264
## [6] 0.0008613264 0.0008613264 0.0008613264 0.0008613264
publicHelpIndex(v)
```

```
## [1] 0.15338882 0.15338882 0.15219976 0.07728894 0.07728894 0.07728894
## [7] 0.07728894 0.07728894 0.07728894 0.07728894
publicHelpChiIndex(v)
## [1] 0.16941222 0.16941222 0.16881669 0.07033698 0.07033698 0.07033698
## [7] 0.07033698 0.07033698 0.07033698 0.07033698
```

5.4.2 Voting power in the European Parliament (2004 – 2009)

The article by Aguirre and Quintas [106] provides an example of voting power in the European Parliament as of the election of June 2004. In the following analysis the seats are allocated to eight supranational political groups. There were 732 members in that parliament, i.e. a majority of 367 votes was needed in order to pass a law. According to Aguirre and Quintas [106] the European Democrats (EPP-ED) had 277 members, the Party of European Socialists (PES) 198 members, the European Liberal Demcrat and Reform Party (ELDR) 68 members, the European Greens / European Free Alliance (Greens/EFA) 40 members, the European United Left / Nordic Green Left (EUL/NGL) 39 members, the Union for a Europe of Nations (UEN) 27 members, Europe of Democracies and Diversities (EDD) 15 members, whereas the rest of 68 parliamentary members were not part of any of the seven aforementioned supranational political groups.

In the following we confirm the results from table 3 of the paper by Aguirre and Quintas [106]:

```
library(CoopGame)
v<-weightedVotingGameVector(n=8, w=c(277,198,68,40,39,27,15,68), q=367)
shapleyShubikIndex(v)
## [1] 0.426190476 0.178571429 0.111904762 0.059523810 0.059523810
## [6] 0.045238095 0.007142857 0.111904762
normalizedBanzhafIndex(v)
## [1] 0.427272727 0.154545455 0.118181818 0.063636364 0.063636364
## [6] 0.045454545 0.009090909 0.118181818
johnstonIndex(v)
## [1] 0.621621622 0.129129129 0.077477477 0.033483483 0.033483483
## [6] 0.023273273 0.004054054 0.077477477
deeganPackelIndex(v)
## [1] 0.2222222 0.1066667 0.1488889 0.1211111 0.1211111 0.1011111
## [7] 0.0300000 0.1488889
publicGoodIndex(v)
## [1] 0.18518519 0.11111111 0.14814815 0.12962963 0.12962963 0.11111111
## [7] 0.03703704 0.14814815
```

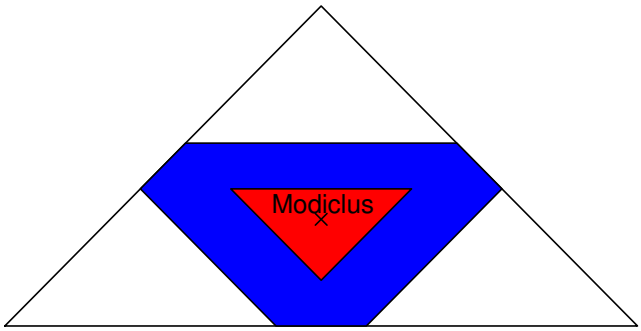
6 Visualization of Solution Concepts for Games with 3 and 4 Players

The package `CoopGame` offers the possibility to visualize both set valued and point valued solution concepts for cooperative games with 3 or 4 players using barycentric coordinates. We strongly hope that our visualization routines will be helpful for colleagues teaching cooperative game theory and wish to acknowledge that for the conversion of Cartesian coordinates to barycentric coordinates `CoopGame` makes use of the package `geometry` [66].

6.1 An example: Generating the CoopGame Logo

Running the following lines of R Code users can generate the CoopGame Logo on the first page of this vignette. It draws the reasonable set of the given game (in blue) as a subset of the imputation set and the core of the game (in red) as a subset of the reasonable set. The modiclus is displayed in black as a point in the core.

```
library(CoopGame)
v0=c(6,8,10,18,20,22,31)
drawImputationset(v0, label=FALSE)
drawReasonableSet(v0, colour="blue", holdOn=TRUE)
drawCore(v0, holdOn=TRUE, colour="red")
drawModiclus(v0, holdOn=TRUE, colour="black")
```



6.2 Some general remarks on the drawing routines in CoopGame

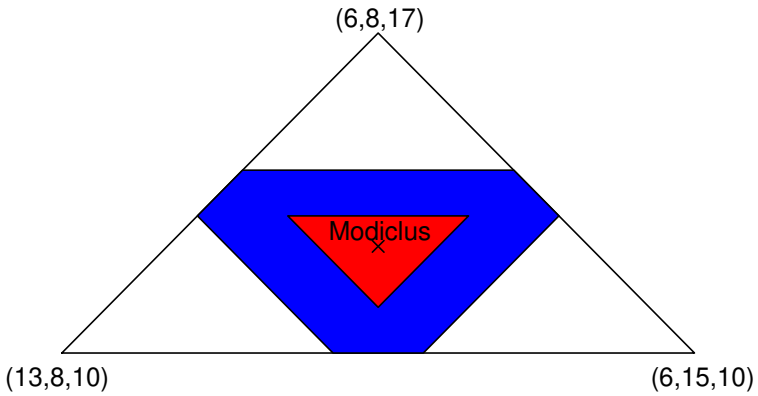
The package CoopGame provides drawing routines for all set solution concepts from chapter 4 for the case of 3 or 4 players. In particular, there are routines drawImputationSet, drawCore, drawCoreCover, drawReasonableSet and drawWeberset. Note that core, core cover, reasonable set and Weber set are all visualized as parts of the imputation set. Note also that the Weber set is only guaranteed to be a subset of the imputation set for superadditive games whereas the other three set solution concepts are always subsets of the imputation set. Also, any efficient point solution method, i.e. any point solution method for which the components of the solution vector are guaranteed to add up to the value of the grand coalition, come with their own drawing routines.

As for the individual drawing routines users can add to an existing plot by setting the parameter holdOn to TRUE. The parameter holdOn is set to FALSE by default and this feature should be handled with a little care by the user. We strongly advise against plots with too many set valued and/or point valued solutions.

In addition, users can also set the `colour` of the polyhedron or point they want to draw and use `name` for naming an individual point. Only for the function `drawImputationset` the colour is fixed to white as any polyhedra or points are always displayed as parts of the imputation set.

Also, only for the function `drawImputationset` there is the default `label=TRUE`. In any plots the vertices of the imputation set are by default labelled by their coordinates. In other words: Unless users start with `drawImputationset(gameVector, label=FALSE)` and then carefully add to their plots with `holdOn=TRUE` the points of the imputation set will by default be labelled. For all four drawing routines `drawCore`, `drawCoreCover`, `drawReasonableSet` and `drawWeberset` the parameter `label` is set to `FALSE` by default, i.e. by default the coordinates of the vertices of these sets are not displayed. The following code yields a variant of the `CoopGame` logo with the three corner points of the imputation set labelled.

```
library(CoopGame)
v0=c(6,8,10,18,20,22,31)
drawReasonableSet(v0, colour="blue")
drawCore(v0, holdOn=TRUE, colour="red")
drawModiclus(v0, holdOn=TRUE, colour="black")
```



Placing labels in the perfect places by default is tricky and users may not always be perfectly satisfied with the results they obtain when plotting set solution concepts with `label = TRUE`. So our advice to users intending to use `CoopGame` for publication ready graphics of the core, the core cover, the reasonable set and/or the Weber set with all points labelled by their coordinates is to export the plots without these labels and then postprocess the plots themselves putting labels of individual points in the desired places.

For drawing any point valued solution concepts `label` is actived by default and a proper `name` for the point valued solution is set by default.

6.3 The power of `rgl`

The R package `rgl` [107] offers the possibility to visualize solution concepts for cooperative games with 4 players in three dimensions. Users are invited to uncomment the following four lines of code, play around with the following example and rotate the nice 3d-plot displayed by `rgl`.

```
#library(CoopGame)
#A=c(0,0,0,0,8,8,8,8,8,8,12,12,12,12,22)
#drawWeberset(A, colour ="grey")
#drawCore(A, colour ="red", holdOn = TRUE)
```

7 Outlook to Future Developments and Acknowledgements

7.1 Future developments and ideas for software for cooperative games

As previously mentioned the package `CoopGame` does not allow for partitions of the player set or for communication structures (see e.g. the book by Slikker and van den Nouweland [8]). The first author of this vignette is about to finish two smaller R packages by the names of `PartitionGames` and `CommunicationGames` for cooperative games with partitions of the player set and for cooperative games on undirected graphs, respectively. For cooperative games with partitions of the player set we refer the readers to the three articles by Owen [108], by Malawski [109] and by Stach [73]. For cooperative games on undirected graphs we refer to the three papers by Myerson [110], by Borm, Owen and Tijs [111] and by van den Brink, van der Laan and Pruzhansky [112].

Once `PartitionGames` and `CommunicationGames` will have been successfully established, cooperative games on directed networks might be the subject of a future R package. We refer to the articles by Gilles, Owen and van den Brink [113], by Gilles and van den Brink [114], and by van den Brink [115] on permission structures as well as to the article by Gilles and van den Brink on measuring domination in directed networks [116]. Chapters 5 and 6 of the book by Gilles [17] provide an excellent insight into this exciting field where public domain implementations of established approaches are still scarce. In the context of hierarchical structures modelled by a directed graph, the works by Faigle and Kern [117] and by Algaba, van den Brink and Dietz [118] investigating cooperative game theory solutions under precedence constraints are also worth mentioning. So is the very recent article on the Shapley value and games with hierarchies by Algaba and van den Brink [119] which appeared in the Handbook of the Shapley Value [120] in 2020. There also exist studies into structures taking into account both communication and hierarchy properties, see the article on network structures with hierarchy and communication by Algaba, van den Brink and Dietz [121].

Cooperative games on multigraphs offer another of many possible lines of future investigations, see e.g. the article by Forlicz, Mercik, Stach and Ramsey [122].

A classical application area for cooperative game theory on graphs is the study of indirect control power in corporate networks, see e.g. the articles by Bertini, Mercik and Stach [123], by Karos and Peters [124] or by Mercik and Stach [125]. We mention a recent article by Staudacher, Olsson and Stach [126] which employs both R and the package `CoopGame` for measuring indirect control.

Currently, the first author of this vignette could think of various other ideas for software

connected to the field of cooperative game theory beyond coalition structures, graphs or multigraphs. We already mentioned that `CoopGame` is far from efficient for computing power indices. Software packages for efficient and truly powerful computation of power indices appear very worthwhile. In this context, we wish to mention three approaches. The first is the paradigm of dynamic programming, see e.g. the article by Kurz [127] from 2016 and the 2021 article by Staudacher, Stach, Kóczy, Filipp, Kramer, Noffke, Olsson, Pichler and Singer [128] which both focus on the special case of weighted voting games. The second is a recent approach based on relational algebra called quasi-ordered binary decision diagrams (QOBDDs), see the papers by Berghammer, Bolus, Rusinowska and de Swart [129], by Berghammer and Bolus [130], by Bolus [131] as well as the doctoral dissertation by Bolus [132]. Finally, the idea of generating functions appears to be of paramount importance, see e.g. the articles by Algaba, Bilbao, Fernandez Garcia and Lopez [133], by Algaba, Bilbao, Fernandez Garcia [134], by Bilbao, Fernandez, Jimenez Losada and Lopez [135] and by Alonso-Meijide and Bowles [136].

Cooperative games in a continuous setting are very intriguing. We mention the work by Aumann and Shapley on values of non-atomic games [137], Owen's multilinear extensions of cooperative games [138], the 1994 paper by Algaba, Bilbao, Fernandez and Jimenez on the Lovasz extension (see the article by Lovasz [139] from 1983) of market games [140] and Neyman's work on values of games with infinitely many players [141] as exemplary representatives of this exciting field. A publicly available software package for cooperative game theory in a continuous setting combining both symbolic and numeric computation appears both challenging and attractive.

7.2 Acknowledgements for version 0.2.1

The R package `CoopGame` started as a software project in the Computer Science M.Sc. program at Kempten University in March 2015 under the supervision of the first author of this vignette. Anna Merkle, Fatma Tokay and Kübra Tokay got the ball rolling with their remarkable efforts. In a subsequent project Alexandra Tiukkel, Michael März and Johannes Anwander, the second author of this vignette, made excellent progress. Later, Daniel Gebele, Franz Müller and Nicole Cyl contributed to `CoopGame` as parts of their B.Sc. theses. In his M.Sc. thesis [102] Johannes Anwander moved `CoopGame` significantly closer to the shape of its first release on CRAN. So this is also the place for the first author to express his deep gratitude to his coauthor for all his assistance in making `CoopGame` available to the general public via CRAN.

The wonderful conference SING14, 14th European Meeting on Game Theory 2018, in Bayreuth in July 2018 provided the first author with a very first opportunity to present small parts of an unfinished version of `CoopGame` to fellow researchers and colleagues in the field of cooperative game theory. The package `CoopGame` benefited from detailed discussions with Izabella Stach (AGH Kraków), Gianfranco Gambarelli (University of Bergamo) and Tamás Solymosi (Corvinus University Budapest) during that conference. Tamás also suggested the term weakly constant-sum game and provided the first author with a copy of the famous 1967 technical report by Kopelowitz [75] on computing the nucleolus. This is also the place to thank a large number of other colleagues, too numerous to name them all, for expressing their interest in `CoopGame` and for making suggestions during that conference. Finally, this is also the opportunity to apologize to any colleagues in the field who might have been waiting for the release of `CoopGame` since July 2018. The process of getting the final details right took overly long.

The final thanks of the first author go to his Kempten colleague Stefan Rieck in the Faculty of Computer Science for having hosted `CoopGame` on his Open Project Server for four years and, in particular, for having provided a subversion server for `CoopGame` until

its ultimate release on CRAN.

7.3 Acknowledgements for version 0.2.2

Version 0.2.2 of `CoopGame` is only a very minor update to version 0.2.1. It solves an internal problem in the automatic generation of this vignette, no longer requires users of `CoopGame` to install the library `rgl` and fixes very few (but certainly not all) bugs. Nevertheless, the package maintainer wishes to thank many of his students and his colleagues – far too numerous to name them all – for their feedback on `CoopGame` and their suggestions for improvements and additional functionality (none of which have been incorporated in version 0.2.2). Special thanks go to Encarnación Algaba (Universidad de Sevilla) who provided the package maintainer with precious suggestions for future software development as well as with references to the literature for improving section 7.1 of this vignette in March 2020. Finally, the first author wishes to express his deep gratitude to his friend and coauthor Izabella Stach (AGH Kraków) for having advertised `CoopGame` within her research community immediately after its initial release in March 2019, for having introduced him to many esteemed colleagues in the scientific community and for the humour she constantly provides in their research cooperation even in situations when she feels that more efficient software than `CoopGame` is needed.

Literature

1. von Neumann, J., and Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press; 1944.

2. van Damme, E.E.C.: On the state of the art in game theory: An interview with Robert Aumann. Games and Economic Behavior, 1998;1–2:181–210.

3. Peleg, B., and Sudhölter, P.: Introduction to the Theory of Cooperative Games. Springer; 2007.

4. Peleg, B.: Working paper 247, Institute of Mathematical Economics, University of Bielefeld. 1996.

5. Peleg, B.: Economic Letters. 1997;55:305–8.

6. Aumann, R.J., and Myerson, R.B.: Endogenous formation of links between players and of coalitions: An application of the Shapley value. In: Roth, A.E., editor. The Shapley Value. Essays in in honor of Lloyd S. Shapley. Cambridge University Press; 1988. pp. 175–91.

7. Thrall, R.M., and Lucas, W.F.: N-person games in partition function form. Naval Research Logistics Quarterly. 1963;10:281–98.

8. Slikker, M., and van den Nouweland, A.: Social and Economic Networks in Cooperative Game Theory. Springer; 2001.

9. Aumann, R.J., and Dreze, J.H.: Cooperative games with coalition structures. International Journal of game theory. 1974;3:217–37.

10. Aumann, R.J., and Maschler, M.: The bargaining set for cooperative games. Advances in game theory. 1964;52:443–76.

11. Aumann, R.J., Peleg, B., and Rabinowitz, P.: A method for computing the kernel of n-person games. Mathematics of Computation. 1965;19:531–51.

12. Aumann, R.J.: Markets with a continuum of traders. Econometrica: Journal of the Econometric Society. 1964;39–50.

13. Aumann, R.J., and Maschler, M.: Von Neumann-Morgenstern solutions to cooperative games without side payments. Bulletin of the American Mathematical Society. 1960;66:173–9.

14. Driessen, T.: Cooperative Games, Solutions and Applications. Springer; 1998.

15. Branzei, R., Dimitrov, D., and Tijs, S.: Models in Cooperative Game Theory. Springer; 2008.

16. Chakravarty, S.R., Mitra, M., and Sarkar, P.: A Course on Cooperative Game Theory. Cambridge University Press; 2015.

17. Gilles, R.P.: The Cooperative Game Theory of Networks and Hierarchies. Springer; 2015.

18. Maschler, M., Solan, E., and Zamir, S.: Game theory. Cambridge University Press; 2013.

19. Peters, H.: Game theory: A Multi-leveled approach. Springer; 2015.

20. Osborne, M.J., and Rubinstein, A.: A Course in Game Theory. MIT Press; 1994.

21. Narahari, Y.: Game theory and mechanism design. World Scientific; 2014.

22. Straffin, P.D.: Game theory and strategy. The Mathematical Association of America; 1996.

23. Wiese, H.: Kooperative Spieltheorie. In German. Oldenbourg-Verlag; 2005.

24. Holler, M.J., and Illing, G.: Einführung in die Spieltheorie. In German. Springer; 2006.

25. Schmeidler, D.: The nucleolus of a characteristic function game. SIAM Journal on applied mathematics. 1969;17:1163–70.

26. Guajardo, M., and Jörnsten, K.: Common mistakes in computing the nucleolus. European Journal of Operational Research. 2015;241:931–5.

27. Young, H.P.: Monotonic Solutions of cooperative games. Int Journal of Game Theory. 1985;14:65–72.

28. Littlechild, S.C., and Vaidya, K.G.: The propensity to disrupt and the disruption nucleolus of a characteristic function game. Int Journal of Game Theory. 1976;5:151–61.

29. Gately, D.: Sharing the Gains from Regional Cooperation. International Economic Review. 1974;15:195–208.

30. Staudacher, J., and Anwander, J.: Conditions for the uniqueness of the Gately point for cooperative games. 10 pages. arXiv preprint, arXiv:1901.01485. 2019.

31. Deegan, J., and Packel, E.W.: A new index of power for simple n-person games. Int Journal of Game Theory. 1978;7:113–23.

32. Holler, M.J.: Forming coalitions and measuring voting power. Political Studies. 1982;30:262–71.

33. Bertini, C.: Minimal winning coalition. Encyclopedia of Power, SAGE Publications. 2011;422–3.

34. Holler, M.J., and Li, X.: From public good index to public value. An axiomatic approach and generalization. Control and Cybernetics. 1995;24:257–70.

35. Bertini, C., and Stach, I.: On Public Values and Power Indices. Decision Making in Manufacturing and Services. 2015;9:9–25.

36. Bertini, C., Gambarelli, G. and Stach, I.: A public help index. In: Braham, M., and Steffen, F., editor. Power, freedom, and voting: Essays in Honour of Manfred J. Holler. Springer; 2008. pp. 83–98.

37. Shapley, L.S.: A value for n-person games. Contributions to the Theory of Games. 1953;2:307–17.

38. Saavedra-Nieves, A.: GameTheoryAllocation: Tools for Calculating Allocations in Game Theory. 2016. https://CRAN.R-project.org/package=GameTheoryAllocation.

39. Cano-Berlanga, S., Giménez-Gómez, J.M., and Vilella, C.: Enjoying cooperative games: The R package GameTheory. Applied Mathematics and Computation. 2017;305:381–93.

40. Prieto, D.: coopProductGame: Cooperative Aspects of Linear Production Programming Problems. 2018. https://CRAN.R-project.org/package=coopProductGame.

41. Owen, G.: On the core of linear production games. Mathematical programming. 1975;9:358–70.

42. Gebele, D., and Staudacher, J.: EvolutionaryGames: Important Concepts of Evolutionary Game Theory. 2017. https://CRAN.R-project.org/package=EvolutionaryGame s.

43. Gura, E.Y., and Maschler, M.: Insights into Game Theory. Cambridge University Press; 2008.

44. Aumann, R.J., and Maschler, M.: Game Theoretic Analysis of a Bankruptcy Problem from the Talmud. Journal of Economic Theory. 1985;36:195–213.

45. Aumann, R.J.: Game Theory in the Talmud. 13 pages. Research Bulletin Series on Jewish Law and Economics. 2002.

46. O'Neill, B.: A problem of rights arbitration from the Talmud. Mathematical Social Sciences. 1982;2:345–71.

47. Otten, G.J.: Characterizations of a Game Theoretical Cost Allocation Method. Zeitschrift für Operations Research. 1996;38:175–85.

48. Young, H.P.: Cost allocation. In: Aumann, R.J., and Hart, S., editor. Handbook of game theory, with economic applications. Volume 2. North-Holland; 1994. pp. 1193–235.

49. Parrachino, I., Zara, S., and Patrone, F.: Cooperative Game Theory and its Application to natural, environmental and water resource issues: 1. Basic Theory. 30 pages. World Bank Policy Research Working Paper 4072. 2006.

50. Tijs, S., and Driessen, T.: Game Theory and Cost Allocation Problems. Management Science. 1996;32:1015–28.

51. Straffin, P.D., and Heaney, J.P.: Game Theory and the Tennessee Valley Authority. Int Journal of Game Theory. 1981;10:35–43.

52. Ransmeier, J.S.: The Tennessee Valley Authority; a case study in the economics of multiple purpose stream planning. 1942.

53. Peleg, B.: Game-theoretic analysis of voting in committees. Handbook of social choice and welfare. 2002;1:395–423.

54. Jackson, M.O.: Social and Economic Networks. Princeton University Press; 2008.

55. Tijs, S.: Bounds for the core of a game and the t-value. In: Moeschlin, O., and Pallaschke, D., editor. Game Theory and Mathematical Economics. North-Holland; 1981. pp. 123–32.

56. Bondareva, O.N.: Some applications of linear programming methods to the theory of cooperative games. Problemy kibernetiki. 1963;10:119–39.

57. Shapley, L.S.: On Balanced Sets and Cores. Naval Research Logistics Quarterly. 1967;14:453–60.

58. Driessen, T., and Tijs, S.: The τ-value, the core and semiconvex games. International Journal of Game Theory. 1985;14:229–47.

59. Gillies, D.B.: Some Theorems on n-person Games. Ph.D. Thesis, Princeton University Press; 1953.

60. Aumann, R.J.: The core of a cooperative game without side payments. Transactions of the American Mathematical Society. 1961;98:539–52.

61. Tijs, S.H., and Lipperts, F.A.S.: The hypercube and the core cover of n-person cooperative games. Cahiers du Centre d'Études de Researche Opérationelle. 1982;24:27–37.

62. Milnor, J.W.: Reasonable Outcomes for N-person Games. Rand Corporation; 1953.

63. Gerard-Varet, L.A., and Zamir, S.: Remarks on the reasonable set of outcomes in a general coalition function form game. International Journal of Game Theory. 1987;16:123–43.

64. Weber, R.J.: Probabilistic values for games. In: Roth, A.E., editor. The Shapley Value. Essays in in honor of Lloyd S. Shapley. Cambridge University Press; 1988. pp. 101–19.

65. Geyer, C.J., Meeden, G.D.: rcdd: Computational Geometry. 2017. https://CRAN.R-project.org/package=rcdd.

66. Habel, K., Grasman, R., Gramacy, R.B., Mozharovskyi, P., and Sterratt, D.C.: geometry: Mesh Generation and Surface Tesselation. 2019. https://CRAN.R-project.org/package=geometry.

67. Fukuda, K.: cddlib Reference Manual. 2005. ftp://ftp.math.ethz.ch/users/fukudak/cdd/cddlibman/cddlibman.html.

68. Rockafellar, R.T.: Convex analysis. Princeton University Press; 2015.

69. Ziegler, G.M.: Lectures on polytopes. Springer; 2012.

70. Sudhölter, P.: The Modified Nucleolus as Canonical Representation of Weighted Majority Games. Mathematics of Operations Research. 1996;21:734–56.

71. Sudhölter, P.: The Modified Nucleolus. Properties and Axiomatizations. Int Journal of Game Theory. 1997;26:147–82.

72. Gambarelli, G.: Banzhaf value. Encyclopedia of Power, SAGE Publications. 2011;53–4.

73. Stach, I.: Sub-Coalitional Approach to Values. In: Transactions on Computational Collective Intelligence XXVII. Springer; 2017. pp. 74–86.

74. Kohlberg, E.: On the nucleolus of a characteristic function game. SIAM Journal on Applied Mathematics. 1971;20:62–6.

75. Kopelowitz, A.: Computation of the kernels of simple games and the nucleolus of n-person games. Department of Mathematics, The Hebrew University of Jerusalem, 45 pages; 1967.

76. Megiddo, N.: On the nonmonotonicity of the bargaining set, the kernel and the nucleolus of a game. SIAM Journal on Applied Mathematics. 1974;27:355–8.

77. Young, H.P., Okada, N., and Hashimoto, T.: Cost allocation in water resources development. Water Resources Research. 1982;18:463–75.

78. Bertini, C.: Shapley value. Encyclopedia of Power, SAGE Publications. 2011;600–3.

79. Tarashnina, S.: The simplified modified nucleolus of a cooperative TU-game. TOP. 2011;19:150–66.

80. Stach, I.: Tijs value. Encyclopedia of Power, SAGE Publications. 2011;667–70.

81. Johnston, R.J.: On the measurement of power: Some reactions to Laver. Environment and Planning A. 1978;10:907–14.

82. Bertini, C., and Stach, I.: Banzhaf voting power measure. Encyclopedia of Power, SAGE Publications. 2011;54–5.

83. Holler, M.J., and Packel, E.W.: Power, luck and the right index. Zeitschrift für Nationalökonomie. 1983;43:21–9.

84. Holler, M.: Public Goods index. Encyclopedia of Power, SAGE Publications. 2011;541–2.

85. Stach, I.: Power Measures and Public Goods. In: Nguyen, N.T., and Kowalczyk, R., editor. Transactions on Computational Collective Intelligence XXIII. Springer; 2016. pp. 99–110.

86. Shapley, L.S., and Shubik, M.: A method for evaluating the distribution of power in a committee system. American political science review. 1954;48:787–92.

87. Stach, I.: Shapley-Shubik index. Encyclopedia of Power, SAGE Publications. 2011;603–6.

88. Barua, R., Chakravarty, S.R., and Sarkar, P.: Measuring p-power of voting. Journal of Economic Theory and Social Development. 2012;1:81–91.

89. Coleman, J.S.: Control of collectivities and the power of a collectivity to act. In: Liberman B, editor. Social choice. Gordon and Breach; 1971. pp. 269–300.

90. Bertini, C., and Stach, I.: Coleman index. Encyclopedia of Power, SAGE Publications. 2011;117–9.

91. de Keijzer, B.: A survey on the computation of power indices. 69 pages. Delft University of Technology. 2008.

92. König, T., and Bräuninger, T.: The inclusiveness of European decision rules. Journal of Theoretical Politics. 1998;10:125–42.

93. Nevison, C.H., Zicht, B., and Schoepke, S.: A naive approach to the Banzhaf index of power. Behavioral Science. 1978;23:130–1.

94. Nevison, H.: Structural power and satisfaction in simple games. In: Applied game theory. Springer; 1979. pp. 39–57.

95. Rae, D. W.: Decision-rules and individual values in constitutional choice. American Political Science Review. 1969;63:40–56.

96. Felsenthal, D.S., Machover, M., and Zwicker, W.: The bicameral postulates and indices of a priori voting power. Theory and Decision. 1998;44:83–116.

97. Felsenthal, D.S., and Machover, M.: The measurement of voting power. Edward Elgar Publishing; 1998.

98. Felsenthal, D.S., and Machover, M.: I-Power. Encyclopedia of Power, SAGE Publications. 2011;354–5.

99. Felsenthal, D.S., and Machover, M.: P-Power. Encyclopedia of Power, SAGE Publications. 2011;528–9.

100. Gelius-Dietrich, G.: glpkAPI: R Interface to C API of GLPK. 2015. https://CRAN.R-project.org/package=glpkAPI.

101. Makhorin, A.: GLPK (GNU linear programming kit). 2008. http://www.gnu.org/s/glpk/glpk.html.

102. Anwander, J.: Untersuchungen zur kooperativen Spieltheorie und Erweiterung des R-Pakets CoopGame. In German. Master Thesis. Hochschule Kempten; 2017.

103. Aumann, R.J.: Some non-superadditive games, and their Shapley values, in the Talmud. International Journal of Game Theory. 2010;39:3–10.

104. Ibn, EZRA: A (1146) Sefar ha-Mispar (The Book of the Number, in Hebrew). Verona (German translation: Silberberg, M). 1895.

105. Bertini, C., Freixas, J., Gambarelli, G., and Stach, I.: Comparing power indices. International Game Theory Review. 2013;15:1340004.

106. Aguirre, J.F., and Quintas, L.G.: Computing power indices in weighted majority games. In: XI congreso argentino de ciencias de la computación. 2005.

107. Adler, D., Murdoch, D., and others: rgl: 3D visualization device system (OpenGL). 2014. https://CRAN.R-project.org/package=rgl.

108. Owen, G.: Values of games with a priori unions. In: Mathematical economics and game theory. Springer; 1977. pp. 76–88.

109. Malawski, M.: Counting power indices for games with a priori unions. In: Essays in Cooperative Games. Springer; 2004. pp. 125–40.

110. Myerson, R.B.: Graphs and cooperation in games. Mathematics of operations research. 1977;2:225–9.

111. Borm, P., Owen, G., and Tijs, S.: On the position value for communication situations. SIAM Journal on Discrete Mathematics. 1992;5:305–20.

112. van den Brink, R., and van der Laan, G., and Pruzhansky, V.: Harsanyi power solutions for graph-restricted games. International journal of game theory. 2011;40:87–110.

113. Gilles, R.P., Owen, G., and van den Brink, R.: Games with permission structures: the conjunctive approach. International Journal of Game Theory. 1992;20:277–93.

114. van den Brink, R., and Gilles, R.P.: Axiomatizations of the conjunctive permission value for games with permission structures. Games and Economic Behavior. 1996;12:113–26.

115. Van Den Brink, R.: An axiomatization of the disjunctive permission value for games with a permission structure. International Journal of Game Theory. 1997;26:27–43.

116. van den Brink, R., and Gilles, R.P.: Measuring domination in directed networks. Social networks. 2000;22:141–57.

117. Faigle, U., and Kern, W.: The Shapley value for cooperative games under precedence constraints. International Journal of Game Theory. 1992;21:249–66.

118. Algaba, E., van den Brink, R., and Dietz, C.: Power measures and solutions for games under precedence constraints. Journal of Optimization Theory and Applications. 2017;172:1008–22.

119. Algaba, E., and van den Brink, R.: The Shapley Value and Games with Hierarchies. In: Handbook of the Shapley value. Chapman; Hall/CRC; 2020. pp. 49–74.

120. Algaba, E., Fragnelli, V., and Sánchez-Soriano, J.: Handbook of the Shapley value. CRC Press; 2020.

121. Algaba, E., van den Brink, R., and Dietz, C.: Network structures with hierarchy and communication. Journal of Optimization Theory and Applications. 2018;179:265–82.

122. Forlicz, S., Mercik, J., Stach, I., and Ramsey, D.: The Shapley value for multigraphs. In: International conference on computational collective intelligence. Springer; 2018. pp. 213–21.

123. Bertini, C., Mercik, J., and Stach, I.: Indirect control and power. Operations Research and Decisions. 2016;26:7–30.

124. Karos, D. and Peters, H.: Indirect control and power in mutual control structures. Games and Economic Behavior. 2015;92:150–65.

125. Mercik, J. and Stach, I.: On Measurement of Control in Corporate Structures. In: Transactions on Computational Collective Intelligence XXXI, LNCS, vol. 11290. Springer; 2018. pp. 64–79.

126. Staudacher, J., Olsson, L., and Stach, I.: Implicit power indices for measuring indirect control in corporate structures. In: Transactions on Computational Collective Intelligence, LNCS, vol. 13010, to appear. Springer; 2021.

127. Kurz, S.: Computing the Power Distribution in the IMF. 10 pages. arXiv preprint, arXiv:1603.01443; 2016.

128. Staudacher, J., Kóczy, L.A., Stach, I., Filipp, J., Kramer, M., Noffke, T., Olsson, L., Pichler, J., and Singer, T.: Computing power indices for weighted voting games via dynamic programming. Operations Research and Decisions. 2021;31:137–60.

129. Berghammer, R., Bolus, S., Rusinowska, A., and De Swart, H.: A relation-algebraic approach to simple games. European Journal of Operational Research. 2011;210:68–80.

130. Berghammer, R., and Bolus, S.: On the use of binary decision diagrams for solving problems on simple games. European Journal of Operational Research. 2012;222:529–41.

131. Bolus, S.: Power indices of simple games and vector-weighted majority games by means of binary decision diagrams. European Journal of Operational Research. 2011;210:258–72.

132. Bolus, S.: A QOBDD-based approach to simple games. PhD thesis. Christian-Albrechts Universität Kiel; 2012.

133. Algaba, E., Bilbao, J.M., Fernández Garcia, J.R., and López, J.J.: Computing power indices in weighted multiple majority games. Mathematical Social Sciences. 2003;46:63–80.

134. Algaba, E., Bilbao, J.M., and Fernández Garcia, J.R.: The distribution of power in the European Constitution. European Journal of Operational Research. 2007;176:1752–66.

135. Bilbao, J.M., Fernandez, J.R., Jiménez Losada, A., and Lopez, J.J.: Generating functions for computing power indices efficiently. Top. 2000;8:191–213.

136. Alonso-Meijide, J.M., and Bowles, C.: Generating functions for coalitional power indices: An application to the IMF. Annals of Operations Research. 2005;137:21–44.

137. Aumann, R.J., and Shapley, L.S.: Values of non-atomic games. Princeton University Press; 2015.

138. Owen, G.: Multilinear extensions of games. Management Science. 1972;18:64–79.

139. Lovász, L.: Submodular functions and convexity. In: Mathematical programming the state of the art. Springer; 1983. pp. 235–57.

140. Algaba, E., Bilbao, J.M., Fernández, J.R., and Jiménez, A.: The Lovász extension of market games. In: Essays in cooperative games. Springer; 2004. pp. 229–38.

141. Neyman, A.: Values of games with infinitely many players. Handbook of game theory with economic applications. 2002;3:2121–67.