Sage Reference Manual: Game Theory

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

CHAPTER

ONE

CO-OPERATIVE GAMES WITH FINITE PLAYERS

This module implements a class for a characteristic function cooperative game. Methods to calculate the Shapley value (a fair way of sharing common resources: see [CEW2011]) as well as test properties of the game (monotonicity, superadditivity) are also included.

AUTHORS:

- James Campbell and Vince Knight (06-2014): Original version

class sage.game_theory.cooperative_game.**CooperativeGame**(*characteristic_function*)
Bases: sage.structure.sage_object.SageObject

An object representing a co-operative game. Primarily used to compute the Shapley value, but can also provide other information.

INPUT:

- characteristic_function – a dictionary containing all possible sets of players:
  - key - each set must be entered as a tuple.
  - value - a real number representing each set of players contribution

EXAMPLES:

The type of game that is currently implemented is referred to as a Characteristic function game. This is a game on a set of players Ω that is defined by a value function $v : C \to \mathbf{R}$ where $C = 2^{\Omega}$ is the set of all coalitions of players. Let $N := |\Omega|$. An example of such a game is shown below:

$$v(c) = \begin{cases} 0 & \text{if } c = \emptyset, \\ 6 & \text{if } c = \{1\}, \\ 12 & \text{if } c = \{2\}, \\ 42 & \text{if } c = \{3\}, \\ 12 & \text{if } c = \{1, 2\}, \\ 42 & \text{if } c = \{1, 3\}, \\ 42 & \text{if } c = \{2, 3\}, \\ 42 & \text{if } c = \{1, 2, 3\}. \end{cases}$$

The function v can be thought of as as a record of contribution of individuals and coalitions of individuals. Of interest, becomes how to fairly share the value of the grand coalition (Ω)? This class allows for such an answer to be formulated by calculating the Shapley value of the game.

Basic examples of how to implement a co-operative game. These functions will be used repeatedly in other examples.

```
sage: integer_function = {(): 0,
....:                     (1,): 6,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 12,
....:                     (1, 3,): 42,
....:                     (2, 3,): 42,
....:                     (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
```

We can also use strings instead of numbers.

```
sage: letter_function = {(): 0,
....:                    ('A',): 6,
....:                    ('B',): 12,
....:                    ('C',): 42,
....:                    ('A', 'B',): 12,
....:                    ('A', 'C',): 42,
....:                    ('B', 'C',): 42,
....:                    ('A', 'B', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
```

Please note that keys should be tuples. `'1, 2, 3'` is not a valid key, neither is `123`. The correct input would be `(1, 2, 3)`. Similarly, for coalitions containing a single element the bracket notation (which tells Sage that it is a tuple) must be used. So `(1)`, `(1,)` are correct however simply inputting `1` is not.

Characteristic function games can be of various types.

A characteristic function game $G = (N, v)$ is monotone if it satisfies $v(C_2) \geq v(C_1)$ for all $C_1 \subseteq C_2$. A characteristic function game $G = (N, v)$ is superadditive if it satisfies $v(C_1 \cup C_2) \geq v(C_1) + v(C_2)$ for all $C_1, C_2 \subseteq 2^{\Omega}$ such that $C_1 \cap C_2 = \emptyset$.

We can test if a game is monotonic or superadditive.

```
sage: letter_game.is_monotone()
True
sage: letter_game.is_superadditive()
False
```

Instances have a basic representation that will display basic information about the game:

```
sage: letter_game
A 3 player co-operative game
```

It can be shown that the "fair" payoff vector, referred to as the Shapley value is given by the following formula:

$$\phi_i(G) = \frac{1}{N!} \sum_{\pi \in \Pi_n} \Delta_{\pi}^{G}(i),$$

where the summation is over the permutations of the players and the marginal contributions of a player for a given permutation is given as:

$$\Delta_{\pi}^{G}(i) = v\big(S_{\pi}(i) \cup \{i\}\big) - v\big(S_{\pi}(i)\big)$$

where $S_{\pi}(i)$ is the set of predecessors of i in π, i.e. $S_{\pi}(i) = \{j \mid \pi(i) > \pi(j)\}$ (or the number of inversions of the form (i, j)).

This payoff vector is "fair" in that it has a collection of properties referred to as: efficiency, symmetry, additivity and Null player. Some of these properties are considered in this documentation (and tests are implemented in the class) but for a good overview see [CEW2011].

Note ([MSZ2013]) that an equivalent formula for the Shapley value is given by:

$$\phi_i(G) = \sum_{S \subseteq \Omega} \sum_{p \in S} \frac{(|S|-1)!(N-|S|)!}{N!}\big(v(S) - v(S \setminus \{p\})\big) = \sum_{S \subseteq \Omega} \sum_{p \in S} \frac{1}{|S|\binom{N}{|S|}}\big(v(S) - v(S \setminus \{p\})\big).$$

This later formulation is implemented in Sage and requires $2^N - 1$ calculations instead of $N!$.

To compute the Shapley value in Sage is simple:

```
sage: letter_game.shapley_value()
{'A': 2, 'C': 35, 'B': 5}
```

The following example implements a (trivial) 10 player characteristic function game with $v(c) = |c|$ for all $c \in 2^\Omega$.

```
sage: def simple_characteristic_function(N):
....:     return {tuple(coalition) : len(coalition)
....:                   for coalition in subsets(range(N))}
sage: g = CooperativeGame(simple_characteristic_function(10))
sage: g.shapley_value()
{0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1}
```

For very large games it might be worth taking advantage of the particular problem structure to calculate the Shapley value and there are also various approximation approaches to obtaining the Shapley value of a game (see [SWJ2008] for one such example). Implementing these would be a worthwhile development For more information about the computational complexity of calculating the Shapley value see [XP1994].

We can test 3 basic properties of any payoff vector λ. The Shapley value (described above) is known to be the unique payoff vector that satisfies these and 1 other property not implemented here (additivity). They are:

- Efficiency - $\sum_{i=1}^{N} \lambda_i = v(\Omega)$ In other words, no value of the total coalition is lost.
- The nullplayer property - If there exists an i such that $v(C \cup i) = v(C)$ for all $C \in 2^\Omega$ then, $\lambda_i = 0$. In other words: if a player does not contribute to any coalition then that player should receive no payoff.
- Symmetry property - If $v(C \cup i) = v(C \cup j)$ for all $C \in 2^\Omega \setminus \{i, j\}$, then $x_i = x_j$. If players contribute symmetrically then they should get the same payoff:

```
sage: payoff_vector = letter_game.shapley_value()
sage: letter_game.is_efficient(payoff_vector)
True
sage: letter_game.nullplayer(payoff_vector)
True
sage: letter_game.is_symmetric(payoff_vector)
True
```

Any payoff vector can be passed to the game and these properties can once again be tested:

```
sage: payoff_vector = {'A': 0, 'C': 35, 'B': 3}
sage: letter_game.is_efficient(payoff_vector)
False
sage: letter_game.nullplayer(payoff_vector)
True
sage: letter_game.is_symmetric(payoff_vector)
True
```

TESTS:

Check that the order within a key does not affect other functions:

```
sage: letter_function = {(): 0,
....:                    ('A',): 6,
....:                    ('B',): 12,
....:                    ('C',): 42,
....:                    ('A', 'B',): 12,
....:                    ('C', 'A',): 42,
....:                    ('B', 'C',): 42,
....:                    ('B', 'A', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
sage: letter_game.shapley_value()
{'A': 2, 'C': 35, 'B': 5}
sage: letter_game.is_monotone()
True
sage: letter_game.is_superadditive()
False
sage: letter_game.is_efficient({'A': 2, 'C': 35, 'B': 5})
True
sage: letter_game.nullplayer({'A': 2, 'C': 35, 'B': 5})
True
sage: letter_game.is_symmetric({'A': 2, 'C': 35, 'B': 5})
True
```

Any payoff vector can be passed to the game and these properties can once again be tested.

```
sage: letter_game.is_efficient({'A': 0, 'C': 35, 'B': 3})
False
sage: letter_game.nullplayer({'A': 0, 'C': 35, 'B': 3})
True
sage: letter_game.is_symmetric({'A': 0, 'C': 35, 'B': 3})
True
```

REFERENCES:

is_efficient (*payoff_vector*)

Return `True` if `payoff_vector` is efficient.

A payoff vector v is efficient if $\sum_{i=1}^{N} \lambda_i = v(\Omega)$; in other words, no value of the total coalition is lost.

INPUT:

- `payoff_vector` – a dictionary where the key is the player and the value is their payoff

EXAMPLES:

An efficient payoff vector:

```
sage: letter_function = {(): 0,
....:                    ('A',): 6,
....:                    ('B',): 12,
....:                    ('C',): 42,
....:                    ('A', 'B',): 12,
....:                    ('A', 'C',): 42,
....:                    ('B', 'C',): 42,
....:                    ('A', 'B', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
sage: letter_game.is_efficient({'A': 14, 'B': 14, 'C': 14})
True

sage: letter_function = {(): 0,
```

```
....:                     ('A',): 6,
....:                     ('B',): 12,
....:                     ('C',): 42,
....:                     ('A', 'B',): 12,
....:                     ('A', 'C',): 42,
....:                     ('B', 'C',): 42,
....:                     ('A', 'B', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
sage: letter_game.is_efficient({'A': 10, 'B': 14, 'C': 14})
False
```

A longer example:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.is_efficient({1: 20, 2: 20, 3: 5, 4: 20})
True
```

is_monotone()

Return `True` if `self` is monotonic.

A game $G = (N, v)$ is monotonic if it satisfies $v(C_2) \geq v(C_1)$ for all $C_1 \subseteq C_2$.

EXAMPLES:

A simple game that is monotone:

```
sage: integer_function = {(): 0,
....:                     (1,): 6,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 12,
....:                     (1, 3,): 42,
....:                     (2, 3,): 42,
....:                     (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
sage: integer_game.is_monotone()
True
```

An example when the game is not monotone:

```
sage: integer_function = {(): 0,
....:                     (1,): 6,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 10,
```

```
....:                   (1, 3,): 42,
....:                   (2, 3,): 42,
....:                   (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
sage: integer_game.is_monotone()
False
```

An example on a longer game:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.is_monotone()
True
```

is_superadditive()

Return `True` if `self` is superadditive.

A characteristic function game $G = (N, v)$ is superadditive if it satisfies $v(C_1 \cup C_2) \geq v(C_1) + v(C_2)$ for all $C_1, C_2 \subseteq 2^{\Omega}$ such that $C_1 \cap C_2 = \emptyset$.

EXAMPLES:

An example that is not superadditive:

```
sage: integer_function = {(): 0,
....:                     (1,): 6,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 12,
....:                     (1, 3,): 42,
....:                     (2, 3,): 42,
....:                     (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
sage: integer_game.is_superadditive()
False
```

An example that is superadditive:

```
sage: A_function = {(): 0,
....:                (1,): 6,
....:                (2,): 12,
....:                (3,): 42,
....:                (1, 2,): 18,
....:                (1, 3,): 48,
....:                (2, 3,): 55,
```

```
....:                (1, 2, 3,): 80}
sage: A_game = CooperativeGame(A_function)
sage: A_game.is_superadditive()
True
```

An example with a longer game that is superadditive:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.is_superadditive()
True
```

An example with a longer game that is not:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 55,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 85}
sage: long_game = CooperativeGame(long_function)
sage: long_game.is_superadditive()
False
```

is_symmetric(*payoff_vector*)

Return `True` if `payoff_vector` possesses the symmetry property.

A payoff vector possesses the symmetry property if $v(C \cup i) = v(C \cup j)$ for all $C \in 2^{\Omega} \setminus \{i, j\}$, then $x_i = x_j$.

INPUT:

- `payoff_vector` – a dictionary where the key is the player and the value is their payoff

EXAMPLES:

A payoff pector that has the symmetry property:

```
sage: letter_function = {(): 0,
....:                    ('A',): 6,
....:                    ('B',): 12,
....:                    ('C',): 42,
....:                    ('A', 'B',): 12,
....:                    ('A', 'C',): 42,
....:                    ('B', 'C',): 42,
....:                    ('A', 'B', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
sage: letter_game.is_symmetric({'A': 5, 'B': 14, 'C': 20})
True
```

A payoff vector that returns `False`:

```
sage: integer_function = {(): 0,
....:                     (1,): 12,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 12,
....:                     (1, 3,): 42,
....:                     (2, 3,): 42,
....:                     (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
sage: integer_game.is_symmetric({1: 2, 2: 5, 3: 35})
False
```

A longer example for symmetry:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.is_symmetric({1: 20, 2: 20, 3: 5, 4: 20})
True
```

nullplayer (*payoff_vector*)

Return `True` if `payoff_vector` possesses the nullplayer property.

A payoff vector v has the nullplayer property if there exists an i such that $v(C \cup i) = v(C)$ for all $C \in 2^{\Omega}$ then, $\lambda_i = 0$. In other words: if a player does not contribute to any coalition then that player should receive no payoff.

INPUT:

- `payoff_vector` – a dictionary where the key is the player and the value is their payoff

EXAMPLES:

A payoff vector that returns `True`:

```
sage: letter_function = {(): 0,
....:                    ('A',): 0,
....:                    ('B',): 12,
....:                    ('C',): 42,
....:                    ('A', 'B',): 12,
....:                    ('A', 'C',): 42,
....:                    ('B', 'C',): 42,
....:                    ('A', 'B', 'C',): 42}
sage: letter_game = CooperativeGame(letter_function)
sage: letter_game.nullplayer({'A': 0, 'B': 14, 'C': 14})
True
```

A payoff vector that returns `False`:

```
sage: A_function = {(): 0,
....:               (1,): 0,
....:               (2,): 12,
....:               (3,): 42,
....:               (1, 2,): 12,
....:               (1, 3,): 42,
....:               (2, 3,): 55,
....:               (1, 2, 3,): 55}
sage: A_game = CooperativeGame(A_function)
sage: A_game.nullplayer({1: 10, 2: 10, 3: 25})
False
```

A longer example for nullplayer:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.nullplayer({1: 20, 2: 20, 3: 5, 4: 20})
True
```

TESTS:

Checks that the function is going through all players:

```
sage: A_function = {(): 0,
....:               (1,): 42,
....:               (2,): 12,
....:               (3,): 0,
....:               (1, 2,): 55,
```

```
....:                (1, 3,): 42,
....:                (2, 3,): 12,
....:                (1, 2, 3,): 55}
sage: A_game = CooperativeGame(A_function)
sage: A_game.nullplayer({1: 10, 2: 10, 3: 25})
False
```

shapley_value()

Return the Shapley value for `self`.

The Shapley value is the "fair" payoff vector and is computed by the following formula:

$$\phi_i(G) = \sum_{S \subseteq \Omega} \sum_{p \in S} \frac{1}{|S| \binom{N}{|S|}} \big(v(S) - v(S \setminus \{p\})\big).$$

EXAMPLES:

A typical example of computing the Shapley value:

```
sage: integer_function = {(): 0,
....:                     (1,): 6,
....:                     (2,): 12,
....:                     (3,): 42,
....:                     (1, 2,): 12,
....:                     (1, 3,): 42,
....:                     (2, 3,): 42,
....:                     (1, 2, 3,): 42}
sage: integer_game = CooperativeGame(integer_function)
sage: integer_game.player_list
(1, 2, 3)
sage: integer_game.shapley_value()
{1: 2, 2: 5, 3: 35}
```

A longer example of the Shapley value:

```
sage: long_function = {(): 0,
....:                  (1,): 0,
....:                  (2,): 0,
....:                  (3,): 0,
....:                  (4,): 0,
....:                  (1, 2): 0,
....:                  (1, 3): 0,
....:                  (1, 4): 0,
....:                  (2, 3): 0,
....:                  (2, 4): 0,
....:                  (3, 4): 0,
....:                  (1, 2, 3): 0,
....:                  (1, 2, 4): 45,
....:                  (1, 3, 4): 40,
....:                  (2, 3, 4): 0,
....:                  (1, 2, 3, 4): 65}
sage: long_game = CooperativeGame(long_function)
sage: long_game.shapley_value()
{1: 70/3, 2: 10, 3: 25/3, 4: 70/3}
```

CHAPTER

TWO

INDICES AND TABLES

- Index
- Module Index
- Search Page

BIBLIOGRAPHY

[CEW2011] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan & Claypool Publishers, (2011). ISBN 9781608456529, doi:10.2200/S00355ED1V01Y201107AIM016.

[MSZ2013] Michael Maschler, Solan Eilon, and Zamir Shmuel. *Game Theory*. Cambridge: Cambridge University Press, (2013). ISBN 9781107005488.

[XP1994] Deng Xiaotie, and Christos Papadimitriou. *On the complexity of cooperative solution concepts.* Mathematics of Operations Research 19.2 (1994): 257-266.

[SWJ2008] Fatima Shaheen, Michael Wooldridge, and Nicholas Jennings. *A linear approximation method for the Shapley value.* Artificial Intelligence 172.14 (2008): 1673-1699.

PYTHON MODULE INDEX

g

INDEX