Санкт-Петербургский государственный университет
Кафедра математической теории игр и статистических решений

Тарасов Антон Алексеевич

Выпускная квалификационная работа бакалавра

Эксцессоподобные решения в интервальных играх

Направление 010302

Прикладная математика, фундаментальная информатика и основы программирования

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Панкратова Я. Б.

Санкт-Петербург

2020

Содержание

Введение

Современную теорию игр можно разделить на два раздела – на теорию некооперативных (бесколиционных) игр и на теорию кооперативных (коалиционных) игр. В некооперативных играх основной упор ставится на изучение стратегий, способов достижения желаемого игроками равновесного состояния. Игроки в таких играх играют каждый сам за себя.

В отличие от бесколиционных игр в кооперативных играх в первую очередь рассматриваются проблемы групп игроков. В данных играх у игроков есть возможность объединяться в коалиции для достижения большей выгоды. При этом предполагается, что если игроки создали коалицию, то коалиции известно, что необходимо сделать, чтобы гарантированно получить свой выигрыш.

Кооперативные игры также разделяются на два основных раздела - на игры с трансферабельной полезностью (ТП игры) и на игры с нетрансферабельной полезностью. Под кооперативными ТП играми, то есть играми с побочными платежами, понимаются игры, в которых полезность измеряется в универсальных, общепринятых для всех участников игры единицах и передается от игрока к игроку без потерь и трансформаций. Иные игры называются играми с нетрансферабельной полезностью.

В данной работе рассматриваются кооперативные интервальные игры с трансферабельной полезностью и их решения, основанные на эксцессах коалиций. Такие игры подходят для описания ситуаций, в которых люди или компании, рассматривающие возможность сотрудничества, не могут определить достижимые выигрыши коалиции, зная с уверенностью только их нижнюю и верхнюю границы. Кооперативные интервальные игры были введены в статьях (Branzei, Dimitrov and Tijs, 2003)[1] и в

(Branzei, Dimitrov, Pickl and Tijs, 2004)[2] в контексте ситуаций банкротства и подробнее изучены (Alparslan Gök, Branzei and Tijs, 2008, Branzei, Tijsand Alparslan Gök, 2008)[3]. Каждая интервальная игра определяется двумя классическими кооперативными играми - нижней и верхней - чьи значения характеристических функций являются границами выигрыша коалиции. Решения для интервальных игр определяются также в интервальной форме. Значение ТП игры дает значение интервала для соответствующего класса интервальных игр, если значение верхней игры преобладает над значением нижней игры.

В данной работе рассматривается новое решение кооперативных интервальных игр - SM-ядро. Впервые SM-ядро для кооперативных игр было введено в статье «The simplified modified nucleolus of a cooperative TU-games» (2011) Тарашниной С.И. [4]. Данное решение интересно тем, что учитывает конструктивную и блокирующую силы коалиции. Подробнее об этом написано в статье «Constructive and Blocking Powers in Some Applications» Тарашниной С. И., Смирновой Н. В.[6]. По этой причине данный подход рассмотрен и для интервальных кооперативных игр.

Постановка задачи

Целью данной работы является рассмотрение понятия интервальных игр, изучение алгоритма построения N-ядра и SM-ядра для кооперативных игр, интервального N-ядра для интервальных игр и построения интервального SM-ядра для интервальных игр. Для выполнения поставленных целей необходимо решить ряд задач:

1. Рассмотреть понятие интервальных игр.
2. Рассмотреть понятие интервального N-ядра и его свойства.
3. Построить интервальное SM-ядро и изучить его свойства.
4. Рассмотреть примеры нахождения N-ядра и SM-ядра для игр трех и четырех лиц.
5. Реализовать компьютерную программу для нахождения интервальных N-ядра и SM-ядра.

Обзор литературы

При написании данной работы была использована научная, учебно-методическая литература, а также публикации из научных изданий.

Основные определения и понятия о кооперативных играх, их решениях и свойствах этих решений были взяты из книги «Теория игр», авторов Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. [7], а также из книги «Кооперативные игры: решения и аксиомы», Печерский С. Л. и Яновская Е. Б. [8].

Основным источником для изучения интервальных игр и их решений являлась публикация «The Nucleolus and the τ-value of Interval Games», Elena B. Yanovskaya [10]. Также были прочитаны и изучены по этой теме работы «Shapley-like values for interval bankruptcy games, Economics Bulletin», R. Branzei, D. Dimitrov and S. Tijs [2], «How to cope with division problems under interval uncertainty of claims», R. Branzei, D. Dimitrov, S. Pickl and S. Tijs [1].

Для изучения алгоритма построения SM-ядра была использована публикация «The simplified modified nucleolus of a cooperative TU-game», Тарашнина С. И. [4]. В статье был рассмотрен алгоритм построения, а также свойства нового решения. Для детального рассмотрения свойств SM-ядра для кооперативных игр была прочитана статья «Properties of solutions of cooperative games with transferable utilities» Тарашнина С. И., Смирнова Н. В. [5]. В статье «The nucleolus is not aggregate monotonic on the domain of convex games» [11] было доказано отсутствие свойства агрегатной монотонности у N-ядра, что сделало нахождение интервального N-ядра нетривиальным решением.

Глава 1. Кооперативные игры

1.1. Классическая кооперативная игра

Для начала рассмотрим понятие кооперативной игры с трансферабельными полезностями для дальнейшего перехода к интервальным играм.

Определение 1.1. [8] Пара (N, v) называется *кооперативной игрой с трансферабельными полезностями*, если

N — конечное непустое множество игроков;

$v\colon 2^N \to R$ — характеристическая функция, ставящая в соответствие каждой коалиции $S \subseteq N$ ее выигрыш $v(S)$, такая что $v(\varnothing) = 0$.

Под G_N будем понимать семейство кооперативных игр.

Определение 1.2. Коалицией называется любое непустое подмножество $S \subseteq N$.

Определение 1.3. [7] Вектор $\alpha = (\alpha_1, \ldots, \alpha_n)$, удовлетворяющий условиям:

$$\alpha_i \geqslant v(\{i\}),\ i \in N,$$
$$\sum_{i=1}^{n} \alpha_i = v(N),$$

где $v(\{i\})$ — значение характеристической функции для коалиции $\{i\}$, называется *дележом.*

Определение 1.4. [4] *Множеством допустимых векторов выигрышей* в игре (N, v) называется множество:

$$X(N, v) = \{x \in R^N : \sum_{i \in N} x_i \leq v(N)\}.$$

Определение 1.5.[4] *Множеством эффективно-рациональных векто-*

ров выигрышей в игре (N, v) называется множество:

$$Y(N, v) = \{x \in R^N : \sum_{i \in N} x_i = v(N)\}.$$

Определение 1.6. [9] *Множеством дележей* в игре (N, v) называется множество эффективно-рациональных выигрышей, удовлетворяющих условию индивидуальной рациональности:

$$I(N, v) = \{x \in Y(N, v) : x_i \geq v(i)\}.$$

Определение 1.7. [4] Решением кооперативной игры называется отображение $\phi : G_N \to R^N$, ставящее в соответствие любой игре из множества кооперативных игр G_N дележ или множество дележей и удовлетворяющее $\phi(N, v) \subseteq X(N, v)$.

Определение 1.8. [8] Кооперативная игра называется *супераддитивной*, если для любых коалиций $S, T \in 2^N$, таких что $S \cap T = \varnothing$, верно неравенство $v(S \cup T) \geq v(S) + v(T)$.

Определение 1.9.[4] *Эксцессом* кооперативной игры (N, v) любой коалиции $S \subseteq N$ для любого $x \in Y(N, v)$ будем называть

$$e(x, v, S) = v(S) - x(S),$$

где $x(S) = \sum_{i \in S} x_i$, $S \subseteq N$.

Эксцесс в данном случае – это мера неудовлетворенности игроков из коалиции S выигрышем $v(S)$.

Определение 1.10. [8] Набор коалиций $T \subset N$ называется *сбалансированным*, если существуют такие $\lambda_S > 0, S \subset T$, что для любого $i \in N$ выполняется $\sum_{\substack{T \subset S \\ i \in S}} \lambda_S = 1$.

Определение 1.11. [12] Сбалансированный набор коалиций называется *минимальным*, если не существует его подмножества, являющегося сбалансированным.

1.2. Решения кооперативных игр

Рассмотрим некоторые решения кооперативных игр, основанные на эксцессах коалиций. В дальнейшем данные решения будут введены для интервальных игр.

Определение 1.12 [4] *N-ядром* игры (N, v) называется множество векторов $x \in I(N, v)$ таких, что:

$$Nu(N, v) = \{x \in I(N, v) : \theta(e(x, v, S)_{S \subseteq N}) \preceq_{lex} \theta(e(y, v, S)_{S \subseteq N}) \; \forall y \in I(N, v)\},$$

где $\theta(e(x, v, S))_{S \subseteq N}$ - вектор эксцессов, расположенных в порядке невозрастания.

Определение 1.13 [4] *Двойственной* характеристической функцией к игре (N, v) называется $v^*(N)$ такая, что выполняется:

$$v^*(S) = v(N) - v(N \backslash S).$$

Определение 1.14 [4] Эксцессом двойственной игры (N, v^*) является:

$$e(x, v^*, S) = v^*(S) - x(S).$$

Определение 1.15. Эксцессом суммы (sum-excess) для игры (N, v) и для двойственной ей игры (N, v^*) называется [4]:

$$\overline{e}(x, v, S) = \frac{1}{2} e(x, v, S) + \frac{1}{2} e(x, v^*, S).$$

Определение 1.16 [4] *The simplified modified nucleolus* или *SM-ядром* игры (N, v) называется множество:

$$\mu(N, v) = \{x \in Y(N, v) : \theta(\overline{e}(x, v, S)_{S \subseteq N}) \preceq_{lex} \theta(\overline{e}(y, v, S)_{S \subseteq N}) \; \forall y \in Y(N, v)\},$$

где $\theta(\overline{e}(x, v, S)_{S \subseteq N})$ — вектор, чьи компоненты являются эксцессами суммы, расположенными в невозрастающем порядке.

В отличие от N-ядра, где рассматриваются обычные эксцессы, SM-ядро строится на сумме эксцессов, характеризующих конструктивную и блокирующую силы коалиции, что делает данное решение в некоторых играх более справедливым чем N-ядро.

1.3. Свойства N-ядра и SM-ядра

Поскольку в дальнейшем будут доказаны некоторые свойства для интервальных версий данных решений, рассмотрим свойства их классических версий.

Определение 1.17. [13] Решение ϕ является *симметричным*, если любые два игрока $i, j \in N$ в игре (N, v) имеют равные возможности, то есть если $v(S \cup \{i\}) = v(S \cup \{j\})$ для всех $S \subset N$, $i, j \notin S$, то $x_i = x_j$ для любых $x \in \phi$.

Определение 1.18. Решение ϕ обладает *свойством болвана* для каждой игры (N, v), если для любого игрока $i \in N$ такого, что $v(S \cup \{i\}) = v(S)$, $i \notin S$, выигрыш $x_i \in \phi$ равен нулю.

Определение 1.19. [4] Решение ϕ является *анонимным*, если для каждой игры $(N, v) \in G_N$ и для каждого биективного отображения $\pi : N \to N$ выполняется $\phi(N, \pi v) = \pi(\phi(N, v))$ (где $\pi v(\pi(S)) = v(S)$, $S \subseteq N$).

Определение 1.20. [13] Решение ϕ является *ковариантным* по стратегической эквивалентности, если для любых двух игр $(N, v_1), (N, v_2) \in G_N$, где $v_2 = \alpha v_1 = \beta$ для некоторой $\alpha > 0, \beta \in R^N$ выполняется $\phi(N, v_2) = \alpha\phi(N, v_1) + \beta$.

Определение 1.21. Решение ϕ является *однозначным*, если $|\phi(N, v)| = 1$ для любой игры $(N, v) \in G_N$.

Определение 1.22. Решение ϕ удовлетворяет свойству непустоты, если $\phi(N, v) = \varnothing$ для любой игры $(N, v) \in G_N$.

Определение 1.23. [13] Решение ϕ является *парето-оптимальным*, если $\phi(N, v) \subseteq Y(N, v)$ для любой игры $(N, v) \in G_N$.

N-ядро для любой игры (N, v) обладает следующими [14]: Анонимность, Симметричность, Ковариантнось, Индивидуальная рациональ-

ность, Парето-оптимальность (эффективность), также оно обладает свойством болвана и является непрерывным и стандартным для игры двух лиц.

Согласно [4], SM-ядро для любой игры (N, v) обладает следующими свойствами: Анонимность, Ковариантность, Парето-оптимальность (эффективность), Однозначность и Непустота.

Глава 2. Интервальные игры

2.1. Основные понятия интервальных игр

Изучение решений интервальных игр начинается с расширения решений классических кооперативных игр на интервальные.

Определение 2.1. [10] *Кооперативной интервальной игрой* в коалиционной форме называется упорядоченная пара (N, w), где $N = \{1, \ldots, n\}$ – конечное множество игроков, и $w\colon 2^N \to I(R)$ – характеристическая функция, такая что $w(\varnothing) = [0, 0]$, где $I(R)$ множество всех непустых замкнутых ограниченных интервалов в R. для любой коалиции $S \in 2^N$, выигрышный интервал $w(S)$ коалиции S в интервальной игре $N = \{1, \ldots, n\}$ имеет форму $[\underline{w}(S), \overline{w}(S)]$, где $\underline{w}(S), \overline{w}(S) : 2^N \to R$ удовлетворяют неравенству $\underline{w}(S) \leq \overline{w}(S)$ для любой коалиции $S \in 2^N$.

Обозначим за IG_N *семейство всех интервальных игр* с множеством игроков N.

В случае, когда все $w(S)$ являются вырожденными интервалами, т.е $\underline{w}(S) = \overline{w}(S)$ для любой коалиции $S \in 2^N$, тогда интервальную игру (N, w) можно сопоставить с классической кооперативной игрой (N, v), где $v(S) = \underline{w}(S) = \overline{w}(S)$ для всех коалиций $S \in 2^N$.[10]

Интервальная игра может быть представлена в виде двух классических ТП-игр: *нижней граничной игры* $(N, \underline{w})$ и *верхней граничной игры* $(N, \overline{w})$.

Для дальнейших рассуждений необходимо ввести некоторые операции над интервалами [15]:

Допустим $\alpha \in R^+$, $I, J \in I(R)$, где $I = [\underline{I}, \overline{I}]$, а $J = [\underline{J}, \overline{J}]$.

1. $|I| = \underline{I} - \overline{I}$.

2. $I + J = [\underline{I} + \underline{J}, \overline{I} + \overline{J}]$.

3. $\alpha I = [\alpha\underline{I}, \alpha\overline{I}]$.

4. $I - J = [\underline{I} - \underline{J}, \overline{I} - \overline{J}]$, если $|I| \geq |J|$.

5. $I \succeq J$ (I слабо лучше J), если $\underline{I} \geq \underline{J}$ и $\overline{I} \geq \overline{J}$.

Определение 2.2. [16] Интервальную игру $(N, w) \in IG_N$ будем называть *супераддитивной*, если для любых коалиций $S, T \in 2^N$, таких что $S \cap T = \varnothing$, верно неравенство

$$w(S \cup T) \succeq w(S) + w(T).$$

Рассмотрим понятие однозначного решения, поскольку именно на нем в дальнейшем будет строиться расширение классических эксцессоподобных решений на семейство интервальных игр.

Определение 2.3. [10] Однозначное решение (значение) ϕ для класса IG_N интервальных игр представляет собой отображение, присваивающее каждой интервальной игре $(N, w) \in IG_N$ пару векторов выигрышей $\phi(N, w) = (x, y) \in R^N \times R^N$, удовлетворяющих условиям $x \in X(N, \underline{w}), y \in X(N, \underline{w})$ и $x \leq y$.

Может показаться, что решением является лишь пара векторов выигрышей, но все таки в данном определении подразумеваются границы интервалов решений.

2.2. Интервальное N-ядро

Интерес для расширения решений, основанных на эксцессах коалициях, заключается в самом эксцессе, поскольку эксцесс является мерой неудовлетворенности коалиции решением. Минимизация данной неудовлетворенности дает интересные решения.

Рассмотрим понятие интервального N-ядра. Поскольку возникает проблема при определении интервального N-ядра следующим образом:

$$INu(N,w) = (x,y), \qquad x = Nu(N,\underline{w}), y = Nu(N,\overline{w}), \qquad x \leq y.$$

Последнее условие не удовлетворяется, поэтому N-ядро определяется с помощью следующих множеств.

Для любых векторов выигрышей $x \in X(N,\underline{w})$, $y \in X(N,\overline{w})$ определим следующие множества [10]:

$$X^y = \{x \in X(N,\underline{w}) | x \leq y\}, \qquad Y^x = \{y \in X(N,\overline{w} | y \geq x\}.$$

N-ядра игр $(N,\underline{w})$, $(N,\overline{w})$ на множествах X^y, Y^x, то есть максимумы лексикографических минимумов отношений векторов эксцессов

$$e(x,\underline{w}) = \{\underline{w}(S) - x(S)\}_{S \subset N}, \qquad e(x,\overline{w}) = \{\overline{w}(S) - x(S)\}_{S \subset N}$$

на областях X^y, Y^x соответственно, определим как $x^y = Nu(N,\underline{w},X^y)$, $y^x = Nu(N,\overline{w},Y^x)$ соответственно. Таким образом, имеются отображения $\underline{N} : X(N,\underline{w}) \to X(N,\underline{w})$, $\overline{N} : X(N,\overline{w}) \to X(N,\overline{w})$ определяемые как

$$\underline{N}(x) = x^{y^x}, \qquad \overline{N}(y) = y^{x^y}.$$

Допустим, x^* - фиксированная точка отображения $\underline{N}$, тогда по определению отображений $\underline{N}$, $\overline{N}$

$$x^* = Nu(N,\underline{w},X^{y^{x^*}}), \qquad y^{x^*} = Nu(N,\overline{w},Y^{x^*})$$

Определение 2.4. [10] Набор фиксированных точек $\{(x^*, y^*)\}$ отображений $\underline{N}$, $\overline{N}$, таких что $y^* = y^{x^*}$, называется *набором интервальных N-ядер* (INuS) для игры (N, w).

Определение 2.5. [10] Интервальным N-ядром (INu) интервальной игры (N, w) будем называть пару (x^*, y^*) векторов выигрышей $x^* \in X(N, \underline{w})$, $y^* \in X(N, \overline{w})$ таких, что $x^* \leq y^*$ и

$$\epsilon(x^*, y^*) \preceq_{lex} \epsilon(x, y) \qquad x \in X(N, \underline{w}), y \in X(N, \overline{w}), \quad x \leq y,$$

где $\epsilon(x, y) \in R^{2^{n+1}-4}$ - вектор эксцессов $e(x, \underline{w}, S)$, $e(x, \overline{w}, T)$, $S, T \subseteq N$, $S, T \neq N, \varnothing$, отсортированных в порядке невозрастания.

Теорема 2.1. [10] На множестве выпуклых интервальных игр существует единственное интервальное N-ядро.

Для всех векторов $x \in X(N, \underline{w})$, $y \in X(N, \overline{w})$ и $\alpha \in R$ определим

$$B_0(x, y) = \{i \in N | x_i = y_i\},$$

$$\underline{B}_\alpha(x) = \{S \subset N | \underline{w}(S) - x(S) \geq \alpha\},$$

$$\overline{B}_\alpha(y) = \{S \subset N | \overline{w}(S) - y(S) \geq \alpha\}.$$

Теорема 2.2. [10] Даны выпуклая интервальная игра (N, w) и пара векторов выигрышей (x^*, y^*), таких что выигрыши $x^* \leq y^*$ нижней и верхней граничных игр соответственно, наборы $\underline{B}_\alpha(x^*)$, $\overline{B}_\alpha(y^*)$ пусты или сбалансированны для всех α, тогда $(x^*, y^*) = IPN(N, w)$.

Если $(x^*, y^*) = INu(N, w)$, тогда для любого $\alpha \in R$ наборы $\underline{B}_\alpha(x^*) \cup B_0(x^*, y^*)$, $\overline{B}_\alpha(y^*) \cup B_0(x^*, y^*)$ являются пустыми или слабо сбалансированными с положительными весами для коалиций из $\underline{B}_\alpha(x^*)$, $\overline{B}_\alpha(y^*)$ соответственно.

2.3. SM-ядро для интервальных игр

Определение 2.6. Интервальным SM-ядром (simplified modified interval nucleolus) супераддитивной интервальной игры (N, w) будем называть пару (x^*, y^*) векторов выигрышей $x^* \in X(N, \underline{w})$, $y^* \in X(N, \overline{w})$ таких, что $x^* \leq y^*$ и

$$\overline{\epsilon}(x^*, y^*) \preceq_{lex} \overline{\epsilon}(x, y), \quad \forall x \in X(N, \underline{w}), y \in X(N, \overline{w}), \quad x \leq y \qquad (1)$$

где $\overline{\epsilon}(x, y) \in R^{2^{n+1}-4}$ - вектор эксцессов суммы $\overline{e}(x, \underline{w}, S)$, $\overline{e}(x, \overline{w}, T)$, $S, T \subseteq N$, $S, T \neq N, \varnothing$, отсортированных в порядке невозрастания.

Теорема 2.3. Существует единственное интервальное SM-ядро на множестве супераддитивных выпуклых интервальных игр.

Доказательство. Доказательство аналогично доказательству единственности интервального N-ядра. [10]

Допустим, что (x_1, y_1) - решение задачи (1) без условия $x \leq y$,то есть $\overline{\epsilon}(x^*, y^*) \preceq_{lex} \overline{\epsilon}(x, y), \quad \forall x \in X(N, \underline{w}), y \in X(N, \overline{w})$.

Тогда $x_1 = \mu(N, \underline{w}), y_1 = \mu(N, \overline{w})$, и существует решение задачи (1). Единственность решения следует из выпуклости области $\{(x, y) | x \in Y(N, \underline{w}), y \in Y(N, \overline{w}), x \leq y\}$.

Теорема 2.4. SM-ядро интервальной игры (N, w) совпадает с интервальным N-ядром игры (N, u), где $u(S) = \frac{1}{2}w(S) + \frac{1}{2}w^*(S)$.

Доказательство. Рассмотрим эксцессы для функции $u(S)$:

$$e(x, \underline{u}, S) = \underline{u}(S) - x(S) = \frac{1}{2}\underline{w}(S) + \frac{1}{2}\underline{w}^*(S) - x(S) =$$
$$= (\frac{1}{2}\underline{w}(S) - \frac{1}{2}x(S)) + \frac{1}{2}\underline{w}^*(S) - \frac{1}{2}x(S)) =$$
$$= \frac{1}{2}e(x, \underline{w}, S) + \frac{1}{2}e(x, \underline{w}, S) = \overline{e}(x, \underline{w}, S)$$

Аналогично для $e(x, \overline{u}, S)$.

2.4. Свойства интервальных N-ядра и SM-ядра

Определение 2.7. Будем говорить, что однозначное решение ϕ называется *анонимным*, если для каждой игры $(N, w) \in IG_N$ и для каждого биективного отображения (перестановки игроков) $\pi : N \to N$ выполняется $\phi(N, \pi w) = \pi(\phi(N, w))$ (где $\pi w(\pi(S)) = w(S)$, $S \subseteq N$).

Утверждение 2.1. Интервальное N-ядро удовлетворяет свойству анонимности для любой игры $(N, w) \in IG_N$.

Доказательство. Следует из определения N-ядра. Рассмотрим две игры (N, w) и $(N, \pi w)$, где $\pi : N \to N$. Эксцессы для данных игр будут выглядить следующим образом:

$$e(x_1, \underline{w}, S) = \underline{w}(S) - x_1(S)$$

$$e(y_1, \overline{w}, S) = \overline{w}(S) - y_1(S)$$

$$e(x_2, \pi\underline{w}, S) = \pi\underline{w}(S) - x_2(S)$$

$$e(y_2, \pi\overline{w}, S) = \pi\overline{w}(S) - y_2(S)$$

Поскольку, очевидно, что $e(x, \underline{w}, S) = e(x, \pi\underline{w}, \pi(S))$, вектора эксцессов будут содержать одинаковые элементы, а их упорядочивание по невозрастанию полностью нивелирует эффект от перестановки игроков.

Утверждение 2.2. Интервальное SM-ядро удовлетворяет свойству анонимности для любой игры $(N, w) \in IG_N$.

Доказательство. Аналогично доказательству анонимности N-ядра.

Определение 2.8. Будем говорить, что однозначное решение ϕ называется *положительно однородным*, если для любого $\alpha \in R^+$ и игры $(N, w) \in IG_N$, игра $(N, \alpha w) \in IG_N$ и $\phi(N, \alpha w) = \alpha\phi(N, w)$.

Утверждение 2.3. Интервальное N-ядро удовлетворяет свойству положительной однородности.

Доказательство. Рассмотрим две игры (N, w), $(N, \alpha w)$, где $\alpha \in R^+$. Экцессы для данных игр будут выглядить так:

$$e(x_1, \underline{w}, S) = \underline{w}(S) - x_1(S)$$

$$e(y_1, \overline{w}, S) = \overline{w}(S) - y_1(S)$$

$$e(x_2, \alpha\underline{w}, S) = \alpha\underline{w}(S) - x_2(S)$$

$$e(y_2, \alpha\overline{w}, S) = \alpha\overline{w}(S) - y_2(S)$$

Тогда векторы эксцессов суммы $\epsilon_{\alpha\underline{w}}(x, y) = \alpha\epsilon_{\underline{w}}(x, y)$, $\epsilon_{\alpha\overline{w}}(x, y) = \alpha\epsilon_{\overline{w}}(x, y)$ и векторы доходов $x_2(S) = \alpha x_1(S)$, $y_2(S) = \alpha y_1(S)$. Таким образом, $\phi(N, \alpha w) = \alpha\phi(N, w)\ \forall\alpha \in R^+$.

Утверждение 2.4. Интервальное SM-ядро удовлетворяет свойству положительной однородности.

Доказательство. Аналогично доказательству положительной однородности N-ядра.

Глава 3. Программная реализация нахождения N-ядра и SM-ядра

3.1. Минимальные сбалансированные коалиции

Одним из методов нахождения интервального N-ядра согласно Теореме 2.4 является поиск лексикографического минимума эксцессов решений, найденных в минимально сбалансированных наборах коалиций. Обычно таким способом решают вручную. У данного метода есть множество проблем, и в случае интервальных игр их количество увеличивается многократно. Важно вспомнить, что даже при нахождении классического N-ядра существуют следующие трудности и особенности:

- Отсутствие решений в некоторых минимально сбалансированных наборов коалиций
- Бесконечное множество решений в некоторых минимально сбалансированных наборах коалиций
- Минимально сбалансированных коалиций может быть крайне много.

Рассмотрим последний пункт подробнее. В рамках данной работы в целях нахождения минимально сбалансированных наборов коалиций была написана программа для их поиска. Опишем алгоритм ее работы.

1. Поиск всевозможных наборов коалиций
2. Удаление наборов коалиций, не содержащих всех игроков
3. Удаление несбалансированных наборов коалиций и наборов коалиций, сбалансированных не минимально

Код программы написан на языке программирования Python. Подробнее с ним ознакомиться можно в Приложении 1.

Для кооперативной игры 4 лиц существует 41 минимально сбалансированный набор коалиций (Таблица 1). 22 из них содержат четыре коалиции, 12 содержат три коалиции, 7 содержат две коалиции. Для интеравальной игры в худшем случае придется перебрать 1681 объединенных наборов коалиций верхней и нижней граничных игр. Это крайне много. Такой метод тяжело реализовать программно.

$\{2,3,4\}, \{1\}$	$\{1,3,4\}, \{2\}$
$\{1,2\}, \{3,4\}$	$\{3\}, \{1,2,4\}$
$\{2,4\}, \{1,3\}$	$\{1,4\}, \{2,3\}$
$\{1,2,3\}, \{4\}$	$\{3,4\}, \{2\}, \{1\}$
$\{3\}, \{2,4\}, \{1\}$	$\{2,3\}, \{4\}, \{1\}$
$\{3\}, \{2\}, \{1,4\}$	$\{4\}, \{1,3\}, \{2\}$
$\{3\}, \{1,2\}, \{4\}$	$\{1,2\}, \{1,3,4\}, \{2,3,4\}$
$\{1,3\}, \{1,2,4\}, \{2,3,4\}$	$\{1,3,4\}, \{2,3\}, \{1,2,4\}$
$\{1,4\}, \{1,2,3\}, \{2,3,4\}$	$\{2,4\}, \{1,2,3\}, \{1,3,4\}$
$\{3,4\}, \{1,2,3\}, \{1,2,4\}$	$\{3\}, \{4\}, \{2\}, \{1\}$
$\{2,4\}, \{2,3\}, \{3,4\}, \{1\}$	$\{2,4\}, \{2,3\}, \{1,3,4\}, \{1\}$
$\{3,4\}, \{2,3\}, \{1,2,4\}, \{1\}$	$\{2,4\}, \{1,2,3\}, \{3,4\}, \{1\}$
$\{1,4\}, \{1,3\}, \{2\}, \{3,4\}$	$\{1,4\}, \{1,3\}, \{2\}, \{2,3,4\}$
$\{1,2,4\}, \{1,3\}, \{2\}, \{3,4\}$	$\{1,4\}, \{1,2,3\}, \{2\}, \{3,4\}$
$\{3\}, \{1,2\}, \{2,4\}, \{1,4\}$	$\{3\}, \{1,2\}, \{2,3,4\}, \{1,4\}$
$\{3\}, \{1,2\}, \{1,3,4\}, \{2,4\}$	$\{1,2\}, \{2,3\}, \{1,3\}, \{4\}$
$\{1,2\}, \{1,3\}, \{4\}, \{2,3,4\}$	$\{1,4\}, \{1,2\}, \{1,3\}, \{2,3,4\}$
$\{1,2\}, \{1,3,4\}, \{2,3\}, \{4\}$	$\{1,2\}, \{1,3,4\}, \{2,3\}, \{2,4\}$
$\{3\}, \{2,4\}, \{1,2,3\}, \{1,4\}$	$\{1,2,4\}, \{2,3\}, \{1,3\}, \{4\}$
$\{3,4\}, \{2,3\}, \{1,3\}, \{1,2,4\}$	$\{1,4\}, \{2,4\}, \{1,2,3\}, \{3,4\}$
$\{1,3,4\}, \{1,2,3\}, \{1,2,4\}, \{2,3,4\}$	

Таблица 1: Минимально сбалансированные коалиции для игры четырех лиц.

3.2. Усовершенствованный метод перебора

Одним из самых простых способов нахождения эксцессоподобного решения интервальной игры является метод перебора. В рамках данной работы был реализован именно данный метод, но с изменениями.

Поскольку перебор 8 переменных занимает крайне много времени, были придуманы следующие усовершествования для ускорения перебора:

- Перебор начинается с шага $\epsilon\alpha$ (стандартно $\epsilon = 1$, $\alpha = 1$).
- Если при следующей итерации после получения лексикографически минимального среди векторов эксцессов не был найден новый таковой, то $\alpha = \alpha/10$. И с уменьшенным шагом рассматривается область, пример которой представлен на Рисунке 1.

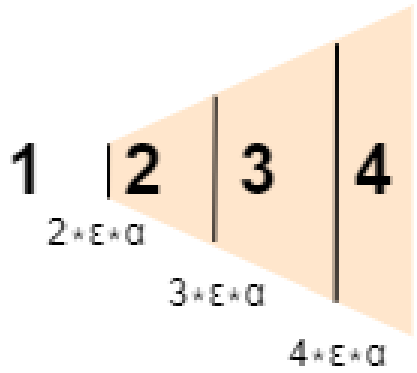

Рис. 1: Область пересчета с уменьшенным шагом, если последнее решение было получено на изменении переменной игрока 2 в игре четырех лиц.

- Перебор переменных происходит для нижней и верхней граничных игр почти отдельно. На каждую итерацию нижней граничной игры происходит полный перебор верхней.
- Полный перебор верхней граничной игры происходит, только если N-ядро верхней граничной игры в сочетании с новыми значениями

нижней граничной игры не дает нового решения, или не удовлетворяется условие $x_i \leq y_i$.

Реализация данного метода на языке программирования Python представлена в Приложении 2.

3.3. Метод минимизации лексикографической разницы

Идея данного метода заключается в минимизации функции лексикографической разницы вектора эксцессов коалиций предыдущего решения с вектором эксцессов текущего. Под лексикографической разницей $f_{lexmin}(\epsilon, \epsilon')$ в случае минимума в данной работе понимается следующее:

$$f_{lexmin}(\epsilon, \epsilon') = \sum_{i=0}^{(\epsilon_i - \epsilon'_i)(\epsilon_{i+1} - \epsilon'_{i+1}) < 0} (\epsilon_i - \epsilon'_i),$$

где ϵ_i, ϵ'_i – компоненты векторов ϵ и ϵ' соответственно.

Рассмотрим задачу подробнее:

$\epsilon(x, y)$ – вектор эксцессов коалиций интервальной игры размера $2^{n+1} - 4$, упорядоченный по невозрастанию, от решения x нижней граничной игры и решения y - верхней.

x, y – решения нижней и верхней граничных игр, удовлетворяющие условиям:

$$x_i \leq y_i \qquad i = 1 \dots n$$

$$x_i \geq 0 \quad i = 1 \dots n, \qquad y_i \geq 0 \quad i = 1 \dots n,$$
$$\sum_{i=1}^{n} x_i \leq \underline{w}(N), \qquad \sum_{i=1}^{n} y_i \leq \overline{w}(N),$$

где x_i, y_i – компоненты векторов x и y соответственно.

ϵ_k – лексикографически минимальный вектор эксцессов, упорядоченный по невозрастанию, на k-ом шаге.

На 0 шаге:

$f_0 = 0$.

$\epsilon_0 = \epsilon(0_{1\times n}, 0_{1\times n})$

На 1 шаге:

$$\text{minimize}(f_0 + f_{lexmin}(\epsilon(x, y), \epsilon_0))$$

Минимизируем фукнцию по x, y. В результате получаем x^1, y^1 – аргументы функции, где она достигает минимума.

$$f_1 = \begin{cases} f_0 + f_{lexmin}(\epsilon(x_1, y_1), \epsilon_0) & \text{если } f_{lexmin}(\epsilon(x_1, y_1), \epsilon_0) \leq 0 \\ f_0 & \text{если } f_{lexmin}(\epsilon(x_1, y_1), \epsilon_0) > 0 \end{cases},$$

$$\epsilon_1 = \begin{cases} \epsilon(x^1, y^1) & \text{если } f_{lexmin}(\epsilon(x^1, y^1), \epsilon_0) \leq 0 \\ \epsilon_0 & \text{если } f_{lexmin}(\epsilon(x^1, y^1), \epsilon_0) > 0 \end{cases},$$

На k-ом шаге:

$$\text{minimize}(f_{k-1} + f_{lexmin}(\epsilon(x, y), \epsilon_0))$$

Минимизируем фукнцию по x, y. В результате получаем x^k, y^k – аргументы функции, где она достигает минимума.

$$f_k = \begin{cases} f_{k-1} + f_{lexmin}(\epsilon(x^k, y^k), \epsilon_{k-1}) & \text{если } f_{lexmin}(\epsilon(x^k, y^k), \epsilon_{k-1}) \leq 0 \\ f_{k-1} & \text{если } f_{lexmin}(\epsilon(x^k, y^k), \epsilon_{k-1}) > 0 \end{cases},$$

$$\epsilon_k = \begin{cases} \epsilon(x^k, y^k) & \text{если } f_{lexmin}(\epsilon(x^k, y^k), \epsilon_{k-1}) \leq 0 \\ \epsilon_{k-1} & \text{если } f_{lexmin}(\epsilon(x^k, y^k), \epsilon_{k-1}) > 0 \end{cases},$$

Критерий останова: Алгоритм следует закончить на m-ом шаге при $f_{m-1} - f_m = \theta$, где θ – заданная точность, или же после достижения определенного количества шагов. Решением будет пара векторов x^m, y^m.

Неплохим методом для решения такой задачи показался метод последовательного программирования наименьших квадратов, поскольку он последовательно решает задачи квадратичного программирования, аппроксимирующие основную задачу оптимизации [17].

Код программы на языке программирования Python представлен в Приложении 3.

Глава 4. Примеры

4.1. Нахождение N-ядра

Рассмотрим следующую игру:

S	$\underline{w}(S)$	$\overline{w}(S)$
1	0	0
2	0	0
3	0	0
1,2	2	2
1,3	0	0
2,3	2	2
N	6	8

Таблица 2: Значения характеристических функций $\underline{w}$, $\overline{w}$ для примера

Эксцессы для обеих игр будут выглядить следующим образом:

$e(x, v, \{1\}) = -x(\{1\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

$e(x, v, \{2\}) = -x(\{2\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

$e(x, v, \{3\}) = -x(\{3\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

$e(x, v, \{1, 2\}) = 2 - x(\{1\}) - x(\{2\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

$e(x, v, \{1, 3\}) = -x(\{1\}) - x(\{3\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

$e(x, v, \{2, 3\}) = 2 - x(\{2\}) - x(\{3\}) \quad \forall v \in \{\underline{w}, \overline{w}\}$

N-ядром для нижней игры $(N, \underline{w})$ будет: $x^* = (2, 2, 2)$

N-ядром для верхней игры $(N, \overline{w})$ будет: $y^* = (\frac{8}{3}, \frac{8}{3}, \frac{8}{3})$

Эти решения дадут нам интервальное N-ядро $\big((2, 2, 2), (\frac{8}{3}, \frac{8}{3}, \frac{8}{3})\big)$, поскольку $x^* \leq y^*$ и

$$\epsilon(x^*, y^*) \preceq_{lex} \epsilon(x, y) \qquad x \in X(N, \underline{w}), y \in X(N, \overline{w}), \quad x \leq y,$$

где $\epsilon(x, y) \in R^{2^{n+1}-4}$ - вектор эксцессов $e(x, \underline{w}, S)$, $e(x, \overline{w}, T)$, $S, T \subseteq N$, $S, T \neq N, \varnothing$, отсортированных в порядке невозрастания.

Результат работы программ:

Метод перебора ($eps = 1, alpha = 1, iters = 3$):

$$((2, 2, 2), (2.66, 2.67, 2.67))$$

Метод минимизации лексикографической разницы:

$$((1.9999602, 2.0000796, 1.9999602), (2.62476, 2.75048, 2.62476))$$

Но не всегда будет работать подобный способ, поскольку в области выпуклых игр как минимум с четырьмя игроками N-ядро классической кооперативной игры не является агрегатно-монотонным [11]. Например, в статье [10] рассматривался следующий пример:
N-ядром для нижней игры $(N, \underline{w})$ будет: $(2, 2, 2, 4)$
N-ядром для верхней игры $(N, \overline{w})$ будет: $(3, 3, 3, 3)$
Интервальным N-ядром для игры (N, w) будет: $\big((2, 2, 2, 4), (\frac{8}{3}, \frac{8}{3}, \frac{8}{3}, 4)\big)$

Результат работы программ:

Метод перебора ($eps = 1, alpha = 1, iters = 3$):

$$((2, 2, 2, 4), (2.66, 2.67, 2.67, 4))$$

Метод минимизации лексикографической разницы:

$$((2.00003837, 2.00010006, 2.00003964, 3.99982194),$$

$$(2.65153455, 2.69633946, 2.65153457, 4.00059142))$$

S	$\underline{w}(S)$	$\overline{w}(S)$
1	0	0
2	0	0
3	0	0
4	0	0
1,2	2	2
1,3	0	0
1,4	2	2
2,3	2	2
2,4	2	2
3,4	2	2
1,2,3	4	4
1,2,4	6	6
1,3,4	6	6
2,3,4	6	6
N	10	12

Таблица 3: Пример, основанный на статье (Hokari, 2000) [11]

4.2. Нахождение SM-ядра

Найдем интервальное SM-ядро для предыдущего примера игры для трех игроков.

Эксцессы суммы для нижней граничной игры $(N, \underline{w})$ будут выглядить следующим образом:

$\overline{e}(x, \underline{w}, \{1\}) = 2 - x(\{1\})$

$\overline{e}(x, \underline{w}, \{2\}) = 3 - x(\{2\})$

$\overline{e}(x, \underline{w}, \{3\}) = 2 - x(\{3\})$

$\overline{e}(x, \underline{w}, \{1,2\}) = 4 - x(\{1\}) - x(\{2\})$

$\overline{e}(x, \underline{w}, \{1,3\}) = 3 - x(\{1\}) - x(\{3\})$

$\overline{e}(x, \underline{w}, \{2,3\}) = 4 - x(\{2\}) - x(\{3\})$

S	$\underline{w}(S)$	$\overline{w}(S)$	$\underline{w}^*(S)$	$\overline{w}^*(S)$
1	0	0	4	6
2	0	0	6	8
3	0	0	4	6
1,2	2	2	6	8
1,3	0	0	6	8
2,3	2	2	6	8
N	6	8	6	8

Таблица 4: Значения характеристических функций для примера из Таблицы 2

SM-ядром для нижней игры $(N, \underline{w})$ будет: $x^* = (\frac{5}{3}, \frac{8}{3}, \frac{5}{3})$

SM-ядром для верхней игры $(N, \overline{w})$ будет: $y^* = (\frac{7}{3}, \frac{10}{3}, \frac{7}{3})$

Тогда интервальное SM-ядро будет выглядить так: $\big((\frac{5}{3}, \frac{8}{3}, \frac{5}{3}), (\frac{7}{3}, \frac{10}{3}, \frac{7}{3})\big)$, поскольку

$$\overline{\epsilon}(x^*, y^*) \preceq_{lex} \overline{\epsilon}(x, y), \quad \forall x \in X(N, \underline{w}), y \in X(N, \overline{w}), \quad x \leq y$$

где $\overline{\epsilon}(x, y) \in R^{2^{n+1}-4}$ - вектор эксцессов суммы $\overline{e}(x, \underline{w}, S)$, $\overline{e}(x, \overline{w}, T)$, $S, T \subseteq N$, $S, T \neq N, \varnothing$, отсортированных в порядке невозрастания.

Результат работы программ:

Метод перебора $(eps = 1, alpha = 1, iters = 3)$:

$$((1.71, 2.64, 1.65), (2.33, 3.33, 2.34))$$

Метод минимизации лексикографической разницы:

$$((1.66666667, 2.66666667, 1.66666667),$$

$$(2.33333333, 3.33333333, 2.33333333))$$

Аналогичным образом решается пример из Таблицы 3.

SM-ядром для нижней игры $(N, \underline{w})$ будет: $x^* = (2.2, 2.4, 2.2, 3.2)$

S	$\underline{w}(S)$	$\overline{w}(S)$	$\underline{w}^*(S)$	$\overline{w}^*(S)$
1	0	0	4	6
2	0	0	4	6
3	0	0	4	6
4	0	0	6	8
1,2	2	2	8	10
1,3	0	0	8	10
1,4	2	2	8	10
2,3	2	2	8	10
2,4	2	2	10	12
3,4	2	2	8	10
1,2,3	4	4	10	12
1,2,4	6	6	10	12
1,3,4	6	6	10	12
2,3,4	6	6	10	12
N	10	12	10	12

Таблица 5: Пример, основанный на статье (Hokari, 2000) [11]

SM-ядром для верхней игры $(N, \overline{w})$ будет: $y^* = (\frac{8}{3}, 3, \frac{8}{3}, \frac{11}{3})$

Тогда интервальное SM-ядро будет выглядить так:

$\big((2.2, 2.4, 2.2, 3.2), (\frac{8}{3}, 3, \frac{8}{3}, \frac{11}{3})\big)$, поскольку

$$\overline{\epsilon}(x^*, y^*) \preceq_{lex} \overline{\epsilon}(x, y), \quad \forall x \in X(N, \underline{w}), y \in X(N, \overline{w}), \quad x \leq y$$

где $\overline{\epsilon}(x, y) \in R^{2^{n+1}-4}$ - вектор эксцессов суммы $\overline{e}(x, \underline{w}, S)$, $\overline{e}(x, \overline{w}, T)$, $S, T \subseteq N$, $S, T \neq N, \varnothing$, отсортированных в порядке невозрастания.

Результат работы программ:

Метод перебора $(eps = 1, alpha = 1, iters = 3)$:

$$((2.2, 2.4, 2.2, 3.2), (2.66, 3.01, 2.66, 3.67))$$

Метод минимизации лексикографической разницы:

$$((1.98490067, 2.40130489, 1.98265413, 3.40656088),$$

$$(2.36885265, 3.02417585, 2.36883547, 4.07486509))$$

Заключение

В данной работе были выполнены все поставленные задачи. Были изучены основные понятия и определения интервальных кооперативных игр, а так же рассмотрена теория для расширения SM-ядра на множество интервальных игр и его построение на нескольких примерах. Были изучены свойства N-ядра и SM-ядра, и некоторые из них были распространены на интервальные игры. Было также реализовано несколько методов программного нахождения N-ядра и SM-ядра на языке программирования Python.

В дальнейшем планируется изучение свойств N-ядра и SM-ядра на множестве интервальных игр, улучшение программных методов нахождения этих решених, путем переноса на другой более быстрый язык программирования и усовершенствования их алгоритмов работы.

Список литературы

1. R. Branzei, D. Dimitrov and S.Tijs. Shapley-like values for interval bankruptcy games // Economics Bulletin 3, 2003, P.1-8.

2. R. Branzei, D. Dimitrov, S. Pickl and S. Tijs. How to cope with division problems under interval uncertainty of claims // International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 12, 2004, P.191-200.

3. Alparslan Gök, S. Z. Branzei, R. and S. Tijs . Cores and stable sets for interval-valued games. Preprint no. 113, Institute of Applied Mathematics, METU and Tilburg University, Center for Economic Research. The Netherlands, CentER, DP 63, Tilburg, 2008.

4. Tarashnina S. I., The simplified modified nucleolus of a cooperative TU-game //Top, 2011, T.19, C. 150-166.

5. Smirnova N. V., Tarashnina S. I., Properties of Solutions of Cooperative Games with Transferable Utilities //Russian Mathematics, 2016, T.60, No. 6, C. 63-74.

6. Tarashnina S. I., Smirnova N. V., Constructive and Blocking Powers in Some Applications //Contributions to Game Theory and Management, 2017, T.10, C. 339-349.

7. Петросян Л. А., Зенкевич Н. А., Шевкопляс Е. В. Теория игр: учебник СПб.: БХВ-Петербург, 2012. С. 159, 187-191.

8. Печерский С. Л., Яновская Е. Б. Кооперативные игры: решения и

аксиомы. М.: Европейский университет в Санкт-Петербурге, 2004. с.32-47.

9. Смирнова Н. В., Тарашнина С. И., Об одном обобщении N-ядра в кооперативных играх//Дискретный анализ и исследование операций, 2011, Т.18, в.4, С. 77-93

10. Elena B. Yanovskaya, The Nucleolus and the τ-value of Interval Games // Contributions to Game Theory and Management, 2010, Volume 3, P.421–430.

11. Hokari, T. The nucleolus is not aggregate monotonic on the domain of convex games // International Journal of Game Theory, 29, 2000, P.133-137.

12. Сбалансированные игры, http://xity.narod.ru/game/balance.pdf

13. Е. А. Лежнина. Свойство подтверждения и аксиоматизация наименьшего ядра // Вестник Санкт-Петербургского Университета, Сер. 10. 2010. Вып. 1.

14. Prenucleolus, https://compscicenter.ru/media/courses/2019-spring/spb-game-theory/slides/game_theory_lecture_290419.pdf

15. S.Z. Alparslan Gök. On the interval Shapley value, Optimization, 2014, P.748

16. Jan Bok and Milan Hladik. Selection-based Approach to Cooperative Interval Games, 2018.

17. Sequential quadratic programming,
https://en.wikipedia.org/wiki/Sequential_quadratic_programming

Приложение 1

Листинг 1: Поиск минимально сбалансированных наборов коалиций

```
from functools import reduce

from scipy.optimize import linprog

# !!! GAME EXAMPLE FOR ALL PROGRAMMES, PAYOFF OF GRAND COALITION
    SHOULD BE LAST !!!

lower_game = {
    "1": 0,
    "2": 0,
    "12": 2,
    "3": 0,
    "13": 0,
    "23": 2,
    "123": 4,
    "4": 0,
    "14": 2,
    "24": 2,
    "124": 6,
    "34": 2,
    "134": 6,
    "234": 6,
    "1234": 10
}

upper_game = {
    "1": 0,
    "2": 0,
    "12": 2,
    "3": 0,
    "13": 0,
```

```
    "23": 2,
    "123": 4,
    "4": 0,
    "14": 2,
    "24": 2,
    "124": 6,
    "34": 2,
    "134": 6,
    "234": 6,
    "1234": 12
}


def get_players_from_str_coalition(coalition: str):
    return list(map(lambda x: int(x), coalition))


def get_A_row(coalition, players_num: int):
    num_coalition = get_players_from_str_coalition(coalition)
    result = [0] * players_num
    for player in num_coalition:
        result[player - 1] = -1
    return result


def get_A_eq_row(coalition_set: set, player: str):
    result = [0] * len(coalition_set)
    for i, coalition in enumerate(coalition_set):
        if player in coalition:
            result[i] = 1
    return result


def is_balanced_coalition_set(coalition_set: set, players: set):
```

```
65      c = [1] * len(coalition_set)
66      A_eq = [get_A_eq_row(coalition_set, player) for player in players
            ]
67      b_eq = [1] * len(players)
68      result = linprog(c, A_eq=A_eq, b_eq=b_eq, method="revised simplex
            ", bounds=[(0, 1)] * len(coalition_set))
69      return result["success"] and 0 not in result["x"]
70
71
72  def get_balanced_coalitions(game: dict):
73      coalitions = list(game.keys())[:-1]
74      players = set(map(lambda x: x, filter(lambda x: len(x) == 1, game
            .keys())))
75
76      coalition_sets = [{coalition} for coalition in coalitions]
77      print(coalition_sets)
78      checking_coalition_sets = coalition_sets.copy()
79      while len(checking_coalition_sets[0]) != (2 ** len(players) - 2):
80          print(list(checking_coalition_sets))
81          new_checking_coalition_sets = list()
82          for coalition_set in checking_coalition_sets:
83              for coalition in coalitions:
84                  new_coalition_set = coalition_set | {coalition}
85                  if new_coalition_set not in coalition_sets:
86                      coalition_sets.append(new_coalition_set)
87                      new_checking_coalition_sets.append(
                            new_coalition_set)
88          checking_coalition_sets = new_checking_coalition_sets.copy()
89      print(coalition_sets)
90
91      player_full_coalition_sets = list()
92      for coalition_set in coalition_sets:
93          if reduce(lambda x, y: x | set(y), coalition_set, set()) ==
                players:
```

```
94              player_full_coalition_sets.append(coalition_set)
95      print(player_full_coalition_sets)
96
97      balanced_coalition_sets = list()
98      for coalition_set in player_full_coalition_sets:
99          if is_balanced_coalition_set(coalition_set, players):
100             balanced_coalition_sets.append(coalition_set)
101     print(balanced_coalition_sets)
102
103     weakly_balanced_coalition_sets = list()
104     for coalition_set in balanced_coalition_sets:
105         is_weakly_balanced = True
106         for another_coalition_set in balanced_coalition_sets:
107             if coalition_set != another_coalition_set and
                    coalition_set.issuperset(another_coalition_set):
108                 is_weakly_balanced = False
109                 break
110         if is_weakly_balanced:
111             weakly_balanced_coalition_sets.append(coalition_set)
112
113     return weakly_balanced_coalition_sets
```

Приложение 2

Листинг 2: Усовершенствованный метод перебора

```
import numpy as np


def get_dual_game(game: dict):
    coalitions = list(game.keys())
    dual_game = dict()
    for coalition in coalitions[:-1]:
        dual_game[coalition] = game[coalitions[-1]] - game["".join(
            sorted(set(coalitions[-1]) ^ set(coalition)))]
    dual_game[coalitions[-1]] = game[coalitions[-1]]
    return dual_game


def get_sum_game(game, dual_game):
    sum_game = dict()
    for coalition in game.keys():
        sum_game[coalition] = 0.5 * game[coalition] + 0.5 * dual_game
            [coalition]
    return sum_game

def find_nucleolus(lower_game: dict, upper_game: dict, x: list = None
    , y: list = None, eps: float = 1, alpha: float = 1, iters: int =
    3):
    players = get_players(lower_game)
    if x is None:
        x = [0]*len(players)
    if y is None:
        y = [0]*len(players)

    def lower_recursive_while(x: list, y: list, player: int = 1, eps:
        float = 1,
```

```
27                                  alpha: float = 1, iters: int = 3,
28                                  min_excess:list = None, solution:list=
                                        None, local_alpha = None,
                                        local_iters = None,
29                                  local_winner = 1, old_x: list = None,
                                        upper_nucleolus: list = None):
30          super_global_min_excess_counter = 0
31          global_min_excess_counter = 0
32          local_min_excess_counter = 0
33          if old_x is None:
34              if player == 1:
35                  old_x = x.copy()
36              else:
37                  old_x = [0]*len(players)
38          if local_alpha is None:
39              local_alpha = alpha
40          if local_iters is None:
41              local_iters = iters
42          if solution is None:
43              solution = [0]*len(players)*2
44          while sum(x) <= list(lower_game.values())[-1]:
45              if player == len(players):
46
47                  if upper_nucleolus is None or not np.all(np.array(x)
                        <= np.array(upper_nucleolus)) or min_excess is
                        None:
48                      solution, min_excess,
                            new_local_min_excess_counter =
                            upper_recursive_while(
49                          x,y, 1,eps, alpha, iters, min_excess,
                                solution)
50                      if solution[:len(players)] == [0]*len(players):
51                          upper_nucleolus = solution[len(players):].
                                copy()
```

```
52                    local_min_excess_counter +=
                          new_local_min_excess_counter
53                    global_min_excess_counter +=
                          new_local_min_excess_counter
54                    super_global_min_excess_counter +=
                          new_local_min_excess_counter
55                elif upper_nucleolus is not None and np.all(np.array(
                      x) <= np.array(upper_nucleolus)):
56                    excess = get_sorted_interval_excess(lower_game,
                          upper_game, x, upper_nucleolus)
57                    if excess < min_excess:
58                        min_excess = excess.copy()
59                        solution = x.copy() + upper_nucleolus.copy()
60                        local_min_excess_counter += 1
61                        global_min_excess_counter += 1
62                        super_global_min_excess_counter += 1
63                        print("excess", min_excess)
64                        print("lower solution", solution, "player",
                              player)
65                        print("alpha", local_alpha, "iters",
                              local_iters)
66                        print()
67            else:
68                solution, min_excess, new_local_min_excess_counter,
                      upper_nucleolus = lower_recursive_while(
69                    x,y, player + 1, eps, alpha, iters, min_excess,
                          solution, local_alpha, local_iters,
                          local_winner,
70                    old_x, upper_nucleolus)
71                local_min_excess_counter +=
                      new_local_min_excess_counter
72                global_min_excess_counter +=
                      new_local_min_excess_counter
73                super_global_min_excess_counter +=
```

```
                    new_local_min_excess_counter
74
75              if local_min_excess_counter == 0:
76                  if global_min_excess_counter != 0:
77                      local_winner = player
78                      if local_iters > 1:
79                          x[player - 1:] = [max(solution[:len(players)
                                ][player - 1] -  (player-local_winner)*eps
                                 * local_alpha, 0) for player in players[
                                player-1:]]
80                          old_x = x.copy()
81                          local_alpha /=10
82                          local_iters -=1
83                          global_min_excess_counter = 0
84                      else:
85                          break
86                  elif old_x[player - 1] + (player-local_winner+2)*eps*
                        local_alpha*10 <= x[player - 1]:
87                      local_alpha = min(alpha, local_alpha * 10)
88                      local_iters = min(iters, local_iters + 1)
89
90              local_min_excess_counter = 0
91              x[player:] = old_x[player:].copy()
92              x[player - 1] = round(x[player - 1]+eps * local_alpha,
                    iters)
93          if player == 1:
94              return solution
95          else:
96              return solution, min_excess,
                    super_global_min_excess_counter, upper_nucleolus
97
98      def upper_recursive_while(x: list, y: list, player: int = 1, eps:
            float = 1, alpha: float = 1, iters: int = 3,
99                                min_excess: list = None, solution: list
```

```
                                    = None, local_alpha = None,
                                   local_iters = None,
100                               local_winner = 1, old_y:list = None):
101
102         super_global_min_excess_counter = 0
103         global_min_excess_counter = 0
104         local_min_excess_counter = 0
105         if old_y is None:
106             old_y = x.copy()
107         if local_alpha is None:
108             local_alpha = alpha
109         if local_iters is None:
110             local_iters = iters
111         if max(y) == 0:
112             y = x.copy()
113         while sum(y) <= list(upper_game.values())[-1] \
114                 and x[player - 1] <= y[player - 1]:
115             excess = get_sorted_interval_excess(lower_game,
                        upper_game, x, y)
116             if min_excess is None or excess < min_excess:
117                 min_excess = excess.copy()
118                 solution = x.copy()+y.copy()
119
120                 local_min_excess_counter += 1
121                 global_min_excess_counter += 1
122                 super_global_min_excess_counter += 1
123                 print("excess", min_excess)
124                 print("upper solution", solution, "player", player)
125                 print("alpha", local_alpha, "iters", local_iters)
126                 print()
127
128             if player < len(players):
129                 solution, min_excess, new_local_min_excess_counter =
                        upper_recursive_while(
```

```
130                     x, y, player + 1, eps, alpha, iters, min_excess,
                            solution, local_alpha, local_iters,
                            local_winner, old_y)
131                 local_min_excess_counter +=
                        new_local_min_excess_counter
132                 global_min_excess_counter +=
                        new_local_min_excess_counter
133                 super_global_min_excess_counter +=
                        new_local_min_excess_counter
134
135             if local_min_excess_counter == 0:
136                 if global_min_excess_counter != 0:
137                     local_winner = player
138                     if local_iters >1:
139                         y[player - 1:] = [max(solution[len(players)
                                :][player - 1] - (player-local_winner+1)*
                                eps * local_alpha, 0, x[player - 1]) for
                                player in players[player-1:]]
140                         old_y = y.copy()
141                         local_alpha /=10
142                         local_iters -= 1
143                         global_min_excess_counter = 0
144                     else:
145                         break
146                 elif old_y[player-1]+(player-local_winner+2)*eps*
                        local_alpha*10<=y[player - 1]:
147                     local_alpha = min(alpha, local_alpha * 10)
148                     local_iters = min(iters, local_iters + 1)
149
150             local_min_excess_counter = 0
151             y[player:] = old_y[player:].copy()
152             y[player - 1] = round(y[player - 1]+eps * local_alpha,
                    iters)
153         return solution, min_excess, super_global_min_excess_counter
```

Приложение 3

Листинг 3: Метод минимизации лексикографической разницы

```
import numpy as np
from scipy import optimize


def get_dual_game(game: dict):
    coalitions = list(game.keys())
    dual_game = dict()
    for coalition in coalitions[:-1]:
        dual_game[coalition] = game[coalitions[-1]] - game["".join(
            sorted(set(coalitions[-1]) ^ set(coalition)))]
    dual_game[coalitions[-1]] = game[coalitions[-1]]
    return dual_game


def get_sum_game(game, dual_game):
    sum_game = dict()
    for coalition in game.keys():
        sum_game[coalition] = 0.5 * game[coalition] + 0.5 * dual_game
            [coalition]
    return sum_game


def get_excess_vector(x, coalitions, problem):
    result = []
    x_coalition = []
    for coalition in coalitions:
        sum = 0
        for player in coalition:
            sum += x[player - 1]
        x_coalition.append(sum)
    for value, payoff in zip(list(problem.values())[:-1], x_coalition
```

```
        [:-1]):
30          result.append(value - payoff)
31      return result
32
33
34  def find_nucleolus(lower_game: dict, upper_game: dict):
35      players = list(map(lambda x: int(x), filter(lambda x: len(x) ==
            1, lower_game.keys())))
36      coalitions = list(map(lambda coalition: list(map(lambda player:
            int(player), coalition)), lower_game.keys()))
37      constraint_matrix = [[0] * (i - 1) + [-1] + [0] * (len(players) -
            i) + [0] * (i - 1) + [1] + [0] * (
38              len(players) - i)
39                           for i in players] \
40                          + [[1] * len(players) + [0] * len(players)] +
                                [[0] * len(players) + [1] * len(players)]
41      lb = [0] * len(players) + [0] + [0]
42      ub = [np.inf] * len(players) + [list(lower_game.values())[-1]] +
            [list(upper_game.values())[-1]]
43      count = 0
44      ans = None
45
46      def lex_min(x, min):
47          com = np.array(x) - np.array(min)
48          sum = 0
49          for i in com:
50              if i <= 0:
51                  sum += i
52              else:
53                  break
54          if sum == 0:
55              for i in com:
56                  if i >= 0:
57                      sum += i
```

```
58                  else:
59                      break
60          return sum
61
62      def get_func(lower_game: dict, upper_game: dict, coalitions,
            players, A, lb, ub):
63          def func(x):
64              nonlocal min_excess
65              nonlocal count
66              nonlocal ans
67              lower_excess = get_excess_vector(x[:len(players)],
                    coalitions, lower_game)
68              upper_excess = get_excess_vector(x[len(players):],
                    coalitions, upper_game)
69              general_excess = sorted(lower_excess + upper_excess,
                    reverse=True)
70              if min_excess is not None:
71                  sum = lex_min(general_excess, min_excess)
72                  if sum < 0:
73                      min_excess = general_excess
74                      ans = x
75                      count = count + sum
76                      result = count
77                  else:
78                      result = count + sum
79              else:
80                  min_excess = general_excess
81                  result = count
82              return result
83
84          return func
85
86      func = get_func(lower_game, upper_game, coalitions, players,
            constraint_matrix, lb, ub)
```

```
87
88      min_excess = sorted(get_excess_vector([0] * len(players),
            coalitions, lower_game) \
89                          + get_excess_vector([0] * len(players),
                                coalitions, upper_game), reverse=True)
90
91      constraint = optimize.LinearConstraint(constraint_matrix, lb, ub)
92
93      res = optimize.minimize(func, np.array([0] * len(players) * 2),
            constraints=constraint,
94                              bounds=[(0, None)] * 2 * len(players),
                                method="SLSQP",
95                              options={'maxiter': 10000, 'ftol': 1e-10,
                                    'disp': False,
96                                         'eps': 0.0001})
97      return res["x"]
```