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

Хамзина Альфия Жалиловна

Магистерская диссертация

Эксцессоподобные решения в играх с разрешенной структурой

Направление 010400
Прикладная математика и информатика

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Тарашнина С. И.

Санкт-Петербург
2019

Содержание

Введение

Ситуации, в которых игроки объединяются для получения большего выигрыша, могут быть представлены в виде кооперативной игры с трансферабельными полезностями (ТП-игры). Однако в реальной жизни не всегда существуют возможности для кооперации по следующим причинам:

- законодательные ограничения,
- ограничения на максимальное (минимальное) число участников в коалиции,
- ограничения, вызванные отсутствием коммуникаций между участниками процесса,
- ограничения, вызванные тем, что игроки должны получить разрешение своих руководителей, чтобы вступить в кооперацию.

Классическая кооперативная игра игнорирует данные причины, поэтому возникает необходимость использовать более сложные математические модели, такие как игры с разрешенной структурой.

Организационная структура многих предприятий имеет иерархию (руководители - подчиненные), которую можно представить в виде направленного графа, поэтому рассмотрение игр с разрешенной структурой и нахождения в них решений является актуальной задачей.

В данной работе рассматривается случай, когда игроки должны получить разрешение на кооперацию от других более сильных игроков. Другими словами, в игре присутствует иерархическая структура, представляемая в виде ориентированного графа на множестве игроков. В зависимости от ограничения на кооперативные возможности игроков рассматриваются два способа построения игры с разрешенной структурой: конъюнктивный и дизъюнктивный.

Решения классической кооперативной игры, такие как C-ядро, вектор Шепли [1] и α-N-ядро , $\alpha \in [0, 1]$, [3] имеют место и в играх с разрешенной структурой. Учитывая особенности построения игр заданного класса, будут существовать два решения — решение в конъюнктивном и дизъюнктивном подходах.

В магистерской диссертации в качестве решения игры с разрешенной структурой рассматривается α-N-ядро, $\alpha \in [0, 1]$, предложенное в работах [2], [4]. Данное решение было выбрано из-за его интересных свойств: оно учитывает как конструктивную, так и блокирующую силу коалиций в игре.

Целью настоящей работы является программная реализация алгоритма, позволяющего вычислять α-N-ядро, $\alpha \in [0, 1]$, в игре с разрешенной структурой для любого конечного числа игроков. Для этого необходимо решить следующие задачи:

1) изучить два способа построения игр с разрешенной структурой — конъюнктивный и дизъюнктивный;

2) программно реализовать построение ограниченных игр (N, r^c_v) и (N, r^d_v);

3) предложить способ построения двойственной дизъюнктивно-ограниченной игры $(N, r^d_{v^*})$;

4) проверить выполнение свойств для игры с разрешенной структурой (N, v^α, D);

5) программно реализовать построение кооперативной игры (N, v^α), ограниченных игр $(N, r^c_{v^\alpha})$ и $(N, r^d_{v^\alpha})$;

6) изучить алгоритм построения пред-N-ядра в дизъюнктивно-оргграниченной игре $(N, r^d_{v^\alpha})$, предложенный в работе [11];

7) программно реализовать алгоритм построения α-N-ядра, $\alpha \in [0, 1]$, в игре (N, r^d_v);

8) проиллюстрировать полученные результаты на примере;

1. Классическая кооперативная игра

1.1. Основные определения и понятия

Понятие игры с разрешенной структурой тесно связано с понятием классической кооперативной игры. В данном разделе дадим основные определения и приведем результаты, касающиеся кооперативных игр n лиц с трансферабельными полезностями (ТП-игр).

Обозначим через $N = \{1, 2, \dots, n\}$ конечное множество игроков. Любое непустое подмножество S из N будем называть *коалицией.* Под *характеристической функцией игры* будем понимать вещественнозначную функцию $v : 2^N \to R^1$, такую что $v(\varnothing) = 0$. Пара (N, v) определяет кооперативную ТП-игру.

Напомним основные свойства кооперативной игры. Игра (N, v) называется *монотонной*, если выполняется неравенство

$$v(S) \leqslant v(T),\ S \subseteq T \subseteq N. \tag{1}$$

Данное свойство означает, что коалиция большей размерности получает больший выигрыш, т. е. игрокам выгодно объединяться. Игра (N, v) называется *выпуклой (вогнутой)*, если выполняется неравенство вида

$$v(S) + v(T) \leqslant (\geqslant) v(S \cap T) + v(S \cup T) \text{ для всех } S, T \subseteq N. \tag{2}$$

Пусть игра $(N, v) \in G^N$, где G^N — это множество всех ТП-игр с фиксированным конечным множеством игроков N. Полагая, что игроки сформировали максимальную коалицию N, рассмотрим задачу распределения величины $v(N)$ между всеми игроками.

Определение 1.1. *Множеством допустимых векторов выигрышей в игре (N, v) назовём множество*

$$X^*(N, v) = \{x \in R^n : x(N) \leqslant v(N)\}.$$

Здесь и далее будем писать $x(S) = \sum\limits_{i \in S} x_i,\ S \subseteq N.$

Определение 1.2. *Множество эффективно-рациональных векторов вы-*

игрышей в игре (N, v) есть множество

$$X^0(N, v) = \{x \in R^n : x(N) = v(N)\}.$$

Определение 1.3. *Множеством дележей $I(N, v)$ в игре (N, v) будем называть множество эффективно-рациональных векторов выигрышей, удовлетворяющих условию индивидуальной рациональности, т. е. множество векторов $x \in X^0(N, v)$, таких что $x_i \geqslant v(\{i\})$ для всех $i \in N$.*

Введем понятие решения кооперативной ТП-игры [4].

Определение 1.4. *Решением на множестве игр G^N называется отображение $f : G^N \to X^*(N, v)$, которое каждой игре $(N, v) \in G^N$ ставит в соответствие подмножество $f(N, v)$ множества $X^*(N, v)$.*

В данной работе рассматривается решение кооперативной ТП-игры α-N-ядро, $\alpha \in [0, 1]$. Данное решение выбрано из-за своей универсальности: при $\alpha = 0$ оно совпадает с анти-пред-N-ядром, при $\alpha = \frac{1}{2}$ оно представляет собой SM-ядро, при $\alpha = 1$ оно совпадает с пред-N-ядром.

1.2. N-ядро

Вычисление α-N-ядра, $\alpha \in [0, 1]$, в игре (N, v) связано с вычислением пред-N-ядра в игре (N, v^α). Поэтому в данном разделе приведем определение пред-N-ядра, основанного на понятии эксцесса коалиции [5, 6].

Определение 1.5. *Для произвольного $x \in X^0(N, v)$ эксцессом коалиции $S \subseteq N$ будем называть величину*

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

Эксцесс коалиции S означает меру неудовлетворенности коалиции своим выигрышем, которое предписывается вектором x.

Впервые понятие N-ядра было введено Шмайдлером в 1969 году [6].

Определение 1.6. *N-ядром относительно множества $X \subset X^0(N, v)$ (обозначается как $\mathcal{N}(X)$) называется множество векторов $x \in X$:*

$$\mathcal{N}(X) = \{x \in X : \theta\left(e(x, v, S)_{S \subseteq N}\right) \preceq_{lex} \theta\left(e(y, v, S)_{S \subseteq N}\right) \text{ для всех } y \in X\},$$

где $\theta\left(e(y,v,S)_{S\subseteq N}\right)$ — *вектор эксцессов, расположенных в порядке невозрастания. Если* $X = X^0(N,v)$, *то соответствующее* $\mathcal{N}(X^0)$ *называется пред-*N*-ядром игры* (N,v) *и обозначается* $\mathcal{PN}$. *Если* $X = I(N,v)$, *то* $\mathcal{N}(X)$ *называется* N*-ядром игры* (N,v) *и обозначается* $\mathcal{N}$.

Из определения 1.6 следует простая интерпретация: N-ядро — это делёж, на котором степень неудовлетворенности всех коалиций, измеряемая величиной их эксцесса, является наименьшей. Из результатов теоремы Шмайдлера следует, что пред-N-ядро игры (N,v) всегда существует и состоит из единственной точки [7]. Этот единственный элемент обозначим через $\nu(N,v)$. Е. Колберг в 1971 году сформулировал теорему, позволяющую характеризовать пред-N-ядро с помощью сбалансированных наборов коалиций [8]. Теорема сформулирована как в работе [5].

Определение 1.7. *Сбалансированным набором коалиций называется набор* $\mathcal{B}$ *коалиций, если существуют такие положительные числа* $\lambda_S > 0$, $S \in \mathcal{B}$, *что для всех* $i \in N$

$$\sum_{S\in\mathcal{B}:S\ni i} \lambda_S = 1. \tag{3}$$

Для произвольной игры (N,v), её вектора выигрышей $x \in X^0(N,v)$ и числа $\gamma \in R^1$ обозначим через $\mathcal{B}_\gamma(x)$ следующий набор коалиций:

$$\mathcal{B}_\gamma(x) = \{S \subsetneqq N \mid e(x,v,S) \geq \gamma\}.$$

Справедлива следующая теорема.

Теорема 1 (Колберга). *Для того, чтобы* $x = \nu(N,v)$ *необходимо и достаточно, чтобы наборы* $\mathcal{B}_\gamma(x)$ *были пусты или сбалансированны для всех* γ.

Данная теорема означает, что делёж является пред-N-ядром тогда и только тогда, когда для любого вещественного числа γ набор коалиций с эксцессом больше γ является сбалансированным набором. Теорема Колберга позволяет находить пред-N-ядро в игре n лиц.

Дадим определение еще одному важному понятию.

Определение 1.8. *Кооперативная игра* (N,v) *называется сбалансированной, если для любого сбалансированного набора коалиций* $\mathcal{B}$ *имеет ме-*

сто неравенство

$$\sum_{S\in\mathcal{B}} \lambda_S \upsilon(S) \leqslant \upsilon(N).$$

Еще один способ вычисления пред-N-ядра был предложен в работе [9]. С помощью теоремы Колберга Хьюбермен доказал, что в сбалансированной игре (N, υ) существенные коалиции и максимальная коалиция N определяют пред-N-ядро.

Определение 1.9. *В игре (N, υ) коалиция S называется существенной, если не существует такого разбиения данной коалиции $S = S_1 \cup S_2 \cup \ldots \cup S_m$, $m > 2$, что*

$$\upsilon(S) \leq \sum_{j=1}^{m} \upsilon(S_j).$$

Коалиции, для которых не выполняется данное определение, называются *несущественными.* Заметим, что одноэлементные коалиции всегда являются существенными. Из определения 1.9 следует, что в игре (N, υ) для эксцесса несущественной коалиции S выполняется следующее неравенство:

$$e(S, x) \leq \sum_{j=1}^{m} e(S_j, x), \text{ для всех } x \in R^n.$$

Приведем несколько фактов для сбалансированных игр, которые будут использованы в дальнейшем. Пусть $x = \nu(N, \upsilon)$. Обозначим через $e^*(N, \upsilon)$ минимальный негативный эксцесс в игре (N, υ), т. е.

$$e^*(N, \upsilon) = \min_{\{S\subset N | S\neq\emptyset\}} -e(S, x).$$

Очевидно, что $e^*(N, \upsilon) \geqslant 0$ тогда и только тогда, когда C-ядро не пусто, а, следовательно, в игре (N, υ) существует пред-N-ядро.

Лемма 1. *Если $e^*(N, \upsilon) > 0$, тогда каждая коалиция $S \subset N$, для эксцесса которой выполняется условие*

$$-e(S, x) = e^*(N, \upsilon), \text{ где } x = \nu(N, \upsilon),$$

является существенной.

Рассмотрим некоторый набор сбалансированных коалиций $B = \{S_1, \ldots, S_k\}$,

$B \in \mathcal{B}$. Справедлива следующая лемма, сформулированная в работе Арина и Иннара [10].

Лемма 2. *Если* $e^*(N, v) \geqslant 0$, *тогда*

$$e^*(N, v) = \min_{B \in \mathcal{B}} \frac{v(N) - \sum_{S \in \mathcal{B}: S \ni i} \lambda_S v(S)}{\sum_{S \in \mathcal{B}: S \ni i} \lambda_S},$$

где λ_S *решения системы (3).*

1.3. α-N-ядро

Перейдем к определению α-N-ядра [4]. Будем рассматривать силу коалиции $S \subset N$ в игре (N, v) двойственным образом. С одной стороны коалиция S гарантированно обеспечивает себе выигрыш $v(S)$, тем самым используя конструктивную силу. С другой стороны, коалиция S может блокировать образование максимальной коалиции N в игре (N, v), обладая блокирующей силой, которая выражается величиной $v^*(S) = v(N) - v(N \setminus S)$. Т. е. $v^*(S)$ есть вклад коалиции S в максимальную коалицию N в случае объединения с коалицией $N \setminus S$. Есть и простая интерпретация — это ценность коалиции S для всего сообщества игроков в игре (N, v). Таким образом, α-N-ядро позволяет учитывать произвольные соотношения конструктивной и блокирующей сил коалиции S, $S \subset N$, т. е. с весом α коалиция использует конструктивную силу, с $1-\alpha$ — блокирующую. Перейдем к формальному определению. Рассмотрим игру $(N, v) \in G^N$. Двойственная игра (N, v^*) к данной задается по правилу

$$v^*(S) = v(N) - v(N \setminus S), \ S \subseteq N. \tag{4}$$

Определение α-N-ядра базируется на понятии α-эксцесса коалиции.

Определение 1.10. *α-эксцессом коалиции $S \subseteq N$ относительно $x \in X^0(N, v)$ для фиксированного $\alpha \in [0, 1]$ называется величина*

$$e^\alpha(x, v, S) = \alpha e(x, v, S) + (1 - \alpha) e(x, v^*, S).$$

Определение 1.11. *Для фиксированного $\alpha \in [0, 1]$ α-N-ядром относи-*

тельно множества $X^0(N, \upsilon)$ называется множество векторов $x \in X^0(N, \upsilon)$:

$$\mathcal{N}^\alpha(X^0) = \{x \in X : \theta\left(e^\alpha(x, \upsilon, S)_{S \subseteq N}\right) \preceq_{lex} \theta\left(e^\alpha(y, \upsilon, S)_{S \subseteq N}\right) \text{ для всех } y \in X^0(N, \upsilon)\},$$

где $\theta\left(e^\alpha(y, \upsilon, S)_{S \subseteq N}\right)$ — вектор эксцессов, расположенных в порядке невозрастания.

Таким образом, для нахождения α-N-ядра мы рассматриваем множество эффективно-рациональных векторов выигрышей и выбираем распределение, соответствующее минимуму отношения лексикографического порядка на множестве α-эксцессов.

В ходе исследования будут применены следующие теоремы [4].

Теорема 2. *Для любого фиксированного $\alpha \in [0, 1]$ α-N-ядро кооперативной игры (N, υ) совпадает с пред-N-ядром игры (N, υ^α), где*

$$\upsilon^\alpha(S) = \alpha\upsilon(S) + (1 - \alpha)\upsilon^*(S). \tag{5}$$

Теорема 3. *Для любого фиксированного $\alpha \in [0, 1]$ α-N-ядро кооперативной игры (N, υ) непусто и состоит из единственной точки.*

Обозначим этот единственный элемент через $\nu^\alpha(\upsilon)$.

Данные результаты позволяют находить α-N-ядро в игре (N, υ) при помощи нахождения пред-N-ядра во вспомогательной игре с характеристической функцией υ^α, что существенно облегчает задачу поиска решения.

В работе [11] рассматривается алгоритм вычисления пред-N-ядра в игре с разрешенной структурой, в которой игроки являются частью иерархии, представляемой в виде ориентированного графа.

2. Игра с разрешенной структурой

Перейдем к определению игр с разрешенной структурой и соответствующих им ограниченных игр.

2.1. Ориентированный граф

Основное отличие игры с разрешенной структурой от классической кооперативной ТП-игры заключается в том, что игрокам требуется разрешение на кооперацию от других игроков. Данное ограничение задается ориентированным графом.

Определение 2.1. *Ориентированный граф есть пара* (N, D), *где* $N \subset \mathbb{N}$ — *множество узлов,* $D \subseteq N \times N$ — *бинарное отношение на множестве* N — *ребра графа, имеющие направление.*

Ориентированный граф, или орграф, в новом классе игр называется *разрешенной структурой.* Множество узлов графа иллюстрирует игроков. *Подграфом орграфа* (N, D) будем называть ориентированный граф $(S, D(S))$, $S \subseteq N$, где $D(S) = \{(i, j) \in D \mid i, j \in S\}$. Так как множество игроков фиксировано, то в дальнейшем ориентированный граф будем обозначать D, подграф $D(S)$. Будем также считать, что орграф обладает свойством *иррефлексивности*, т. е. пара (i, i) не принадлежит D для любых $i \in D$.

Рассмотрим основные определения из теории графов. Через $S_D(i) = \{j \in N \mid (i, j) \in N\}$ обозначим *последователей узла* i, а через $P_D(i) = \{j \in N \mid (j, i) \in N\}$ *предшественников узла* i в D. В игре с разрешенной структурой элементы множества $S_D(i)$ будем называть подчиненными игрока i, а элементы множества $P_D(i)$ будем называть руководителями.

Определение 2.2. *Прямым путем от* i *до* j *в графе* D *называется последовательность узлов* $(h_1, \ldots, h_t)$, *где* $h_1 = i, h_k + 1 \in S_D(h_k)$ *для* $k = 1, \ldots, t-1$ *и* $h_t = j$.

Прямой путь $(i_1, \ldots, i_t)$, $t \geqslant 2$, называется *циклом*, если $(i_t, i_1) \in D$. Орграф D называется *ациклическим*, если он не содержит циклов, т. е. граф D имеет хотя бы один узел i_0 такой, что $P_D(i_0) = \emptyset$. Такие узлы будем называть *корневыми вершинами*, а соответствующих им игроков назовем *независимыми игроками* и обозначим их как $R_D = \{i \in N \mid P_D(i) = \emptyset\}$.

Ориентированный граф D называется *слабо связным*, если для двух различных вершин графа существует по крайней мере один маршрут, соединящий их. Множество всех иррефлексивных, ацикличных и слабо связных орграфов на N обозначим через $\mathcal{D}^N$.

2.2. Ограниченная игра

Введем определение игры с разрешенной структурой [11].

Определение 2.3. *Тройка* (N, v, D)*, где* $N \subset \mathbb{N}$ *— множество игроков,* $v \in G^N$ *— характеристическая функция соответствующей ТП-игры,* D *— орграф на* N*, называется игрой с разрешенной структурой.*

В работе сделаны следующие предположения:

1) ограф $D \in \mathcal{D}^N$, т. е. является иррефлексивным, ациклическим и слабо связным.

2) игрок 1 является независимым игроком, т. е. соответсвующий узел есть корневая вершина орграфа D, $R_D = \{1\}$.

3) максимальная коалиция N является несущественной в игре (N, v, D).

В зависимости от вида графа существуют два подхода к построению игры с разрешенной структурой: *конъюнктивный* и *дизъюнктивный* [12]. Основное отличие данных способов построения заключается в наборе возможных коалиций. Игру с разрешенной структурой, построенную в конъюнктивном или дизъюнктивном подходе, будем называть *ограниченной* игрой.

Конъюнктивный подход

В конъюнктивном подходе предполагается, что игроку необходимо разрешение на кооперацию с другими игроками от всех своих руководителей. В данном подходе коалиция возможна тогда и только тогда, когда для каждого игрока рассматриваемой коалиции все его руководители также содержатся в данной коалиции. Формально, *набор конъюнктивно-возможных коалиций* задается как:

$$\Phi_D^c = \{S \subseteq N \mid P_D(i) \subseteq S \text{ для всех } i \in S\}.$$

Для каждой коалиции $S \subseteq N$ обозначим через $\sigma_D^c(S) = \bigcup_{F \in \Phi_D^c | F \subseteq S} F$ *максимальную конъюнктивно-возможную коалицию для* S.

Определение 2.4. *Конъюнктивно-ограниченная игра есть пара* $(N, r_{\upsilon,D}^c)$, *где* N *— конечное множество игроков,* $r_{\upsilon,D}^c : 2^N \to R$ *— характеристическая функция, которая определяет для* $S \subseteq N$ *в качестве выигрыша значение ее максимальной конъюнктивно-возможной коалиции, т. е.* $r_{\upsilon,D}^c(S) = \upsilon(\sigma_D^c(S))$ *для всех* $S \subseteq N$.

Дизъюнктивный подход

Дизъюнктивный подход имеет альтернативную интерпретацию: игроку требуется разрешение на кооперацию хотя бы от одного своего руководителя. В этом случае коалиция возможна, если в ней содержатся игрок по крайней мере с одним своим руководителем и независимые игроки. Формально, *набор дизъюнктивно-возможных коалиций* есть

$$\Phi_D^d = \{S \subseteq N \mid P_D(i) \cap S \neq \emptyset \text{ для всех } i \in S \setminus R_D\}.$$

Для каждой коалиции $S \subseteq N$ обозначим через $\sigma_D^d(S) = \bigcup_{F \in \Phi_D^d | F \subseteq S} F$ *максимальную дизъюнктивно-возможную коалицию для* S.

Определение 2.5. *Дизъюнктивно-ограниченнная игра есть пара* $(N, r_{\upsilon,D}^d)$, *где* N *— конечное множество игроков,* $r_{\upsilon,D}^d : 2^N \to R$ *— характеристическая функция, которая определяет для* $S \subseteq N$ *в качестве выигрыша значение максимальной дизъюнктивно-возможной коалиции, т. е.* $r_{\upsilon,D}^d(S) = \upsilon(\sigma_D^d(S))$ *для всех* $S \subseteq N$.

Из пункта 2 предположения о виде игры с разрешенной структурой следует, что $r_{\upsilon,D}^d(S) = 0$, когда $1 \notin S$. Также согласно теореме, сформулированной в работе [13], максимальная коалиция N всегда содержится в наборе Φ_D^d.

В дальнейшем будем использовать следующие обозначения:

- (N, r_υ^d) для дизъюнктивно-ограниченной игры, соответствующей игре с разрешенной структурой (N, υ, D),
- r_υ^d для характеристической функции дизъюнктивно-ограниченной игры,

- Φ^c для набора конъюнктивно-возможных коалиций,
- $\sigma^c(S)$ для максимальной конъюнктивно-возможной коалиции S,
- Φ^d для набора дизъюнктивно-возможных коалиций,
- $\sigma^d(S)$ для максимальной дизъюнктивно-возможной коалиции S.

В работе рассматривается игра с разрешенной структурой (N, v, D) и соответствующая ей дизъюнктивно-ограниченная игра (N, r_v^d).

2.3. Пример программной реализации построения ограниченных игр (N, r_v^c) и (N, r_v^d)

Для иллюстрации понятий, описанных выше, приведем пример.

Рассмотрим игру с разрешенной структурой четырех лиц (N, v, D) на множестве игроков $N = \{1, 2, 3, 4\}$, с графом $D = \{(1, 2), (1, 3), (2, 4), (3, 4)\}$, представленном на рисунке 1, с характеристической функцией $v(S)$:

$$v(\{1\}) = 1,\ v(\{2\}) = 5,\ v(\{3\}) = 6,\ v(\{4\}) = 8,$$

$$v(\{12\}) = 7,\ v(\{13\}) = 9,\ v(\{14\}) = 11,\ v(\{23\}) = 14,\ v(\{24\}) = 16,$$

$$v(\{34\}) = 17,\ v(\{123\}) = 18,\ v(\{124\}) = 21,\ v(\{134\}) = 23,\ v(\{234\}) = 24,$$

$$v(\{1234\}) = 30.$$

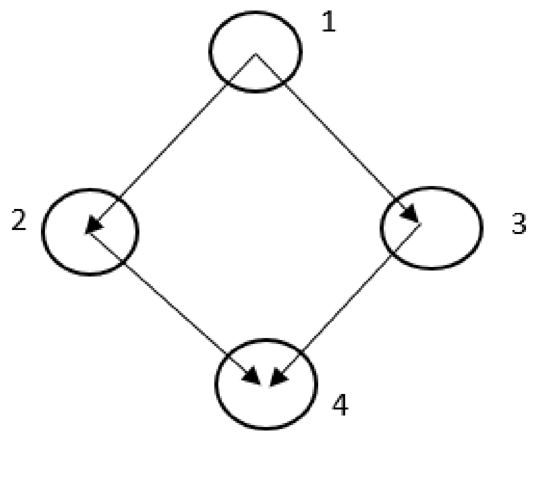


Рис. 1: Граф D игры (N, v, D)

Найдем множество последователей и предшественников для каждого узла (подчиненных и руководителей для игрока)

$$S_D(1) = \{2, 3\}, P_D(1) = \emptyset,$$

$$S_D(2) = \{4\}, P_D(2) = \{1\},$$

$$S_D(3) = \{4\}, P_D(3) = \{1\},$$

$$S_D(4) = \emptyset, P_D(4) = \{2, 3\}.$$

Независимым игроком является первый игрок.

Конъюнктивный подход

Определим набор конъюнктивно-возможных коалиций в данной игре

$$\Phi^c = \{\{1\},\ \{12\},\ \{13\},\ \{123\},\ \{1234\}\}.$$

Для каждой коалиции $S \subseteq N$ найдем максимальную конъюнктивно-возможную коалицию в наборе Φ^c

$$\sigma^c(\{1\}) = \{1\},\ \sigma^c(\{2\}) = \emptyset,\ \sigma^c(\{3\}) = \emptyset,\ \sigma^c(\{4\}) = \emptyset,$$

$$\sigma^c(\{12\}) = \{12\},\ \sigma^c(\{13\}) = \{13\},\ \sigma^c(\{14\}) = \{1\}, \sigma^c(\{23\}) = \emptyset,$$

$$\sigma^c(\{24\}) = \emptyset,\ \sigma^c(\{34\}) = \emptyset,\ \sigma^c(\{123\}) = \{123\},\ \sigma^c(\{124\}) = \{12\},$$

$$\sigma^c(\{134\}) = \{13\}, \sigma^c(\{234\}) = \emptyset,\ \sigma^c(\{1234\}) = \{1234\}.$$

Конъюнктивно-ограниченную игру (N, r_v^c) построим по определению.

$$r_v^c(\{1\}) = v(\sigma^c\{1\}) = v(\{1\}) = 1,$$

$$r_v^c(\{2\}) = v(\sigma^c\{2\}) = v(\emptyset) = 0,$$

$$r_v^c(\{3\}) = v(\sigma^c\{3\}) = v(\emptyset) = 0,$$

$$r_v^c(\{4\}) = v(\sigma^c\{4\}) = v(\emptyset) = 0,$$

$$r_v^c(\{12\}) = v(\sigma^c\{12\}) = v(\{12\}) = 7,$$

$$r_v^c(\{13\}) = v(\sigma^c\{13\}) = v(\{13\}) = 9,$$

$$r_v^c(\{14\}) = v(\sigma^c\{14\}) = v(\{1\}) = 1,$$

$$r_v^c(\{23\}) = v(\sigma^c\{23\}) = v(\emptyset) = 0,$$

$$r_v^c(\{24\}) = v(\sigma^c\{24\}) = v(\emptyset) = 0,$$

$$r_v^c(\{34\}) = v(\sigma^c\{34\}) = v(\emptyset) = 0,$$

$$r_v^c(\{123\}) = v(\sigma^c\{123\}) = v(\{123\}) = 18,$$

$$r_v^c(\{124\}) = v(\sigma^c\{124\}) = v(\{12\}) = 7,$$

$$r_v^c(\{134\}) = v(\sigma^c\{134\}) = v(\{13\}) = 9,$$

$$r_v^c(\{234\}) = v(\sigma^c\{234\}) = v(\emptyset) = 0,$$

$$r_v^c(\{1234\}) = v(\sigma^c\{1234\}) = v(\{1234\}) = 30.$$

Дизъюнктивный подход

Рассмотрим дизъюнктивный подход к построению игры с разрешенной структурой. Определим набор дизъюнктивно-возможных коалиций:

$$\Phi^d = \{\{1\},\ \{12\},\ \{13\},\ \{123\},\ \{124\},\ \{134\},\ \{1234\}\}.$$

Определим для каждой коалиции $S \subseteq N$ максимальную дизъюнктивно-возможную коалицию в наборе Φ^d

$$\sigma^d(\{1\}) = \{1\},\ \sigma^d(\{2\}) = \emptyset,\ \sigma^d(\{3\}) = \emptyset,\ \sigma^d(\{4\}) = \emptyset,$$

$$\sigma^d(\{12\}) = \{12\},\ \sigma^d(\{13\}) = \{13\},\ \sigma^d(\{14\}) = \{1\},\ \sigma^d(\{23\}) = \emptyset,$$

$$\sigma^d(\{24\}) = \emptyset,\ \sigma^d(\{34\}) = \emptyset,\ \sigma^d(\{123\}) = \{123\},\ \sigma^d(\{124\}) = \{124\},$$

$$\sigma^d(\{134\}) = \{134\},\ \sigma^d(\{234\}) = \emptyset,\ \sigma^d(\{1234\}) = \{1234\}.$$

Дизъюнктивно-ограниченную игру (N, r_v^d) построим по определению.

$$r_v^d(\{1\}) = v(\sigma^d\{1\}) = v(\emptyset) = 1,$$

$$r_v^d(\{2\}) = v(\sigma^d\{2\}) = v(\emptyset) = 0,$$

$$r_v^d(\{3\}) = v(\sigma^d\{3\}) = v(\emptyset) = 0,$$

$$r_v^d(\{4\}) = v(\sigma^d\{4\}) = v(\emptyset) = 0,$$

$$r_v^d(\{12\}) = v(\sigma^d\{12\}) = v(\{12\}) = 7,$$

$$r_v^d(\{13\}) = v(\sigma^d\{13\}) = v(\{13\}) = 9,$$

$$r_v^d(\{14\}) = v(\sigma^d\{14\}) = v(\{1\}) = 1,$$

$$r_v^d(\{23\}) = v(\sigma^d\{23\}) = v(\emptyset) = 0,$$

$$r_v^d(\{24\}) = v(\sigma^d\{24\}) = v(\emptyset) = 0,$$

$$r_v^d(\{34\}) = v(\sigma^d\{34\}) = v(\emptyset) = 0,$$

$$r_v^d(\{123\}) = v(\sigma^d\{123\}) = v(\{123\}) = 18,$$

$$r_v^d(\{124\}) = v(\sigma^d\{124\}) = v(\{124\}) = 21,$$

$$r_v^d(\{134\}) = v(\sigma^d\{134\}) = v(\{134\}) = 23,$$

$$r_v^d(\{234\}) = v(\sigma^d\{234\}) = v(\emptyset) = 0,$$

$$r_v^d(\{1234\}) = v(\sigma^d\{1234\}) = v(\{1234\}) = 30.$$

Таким образом, построены ограниченные игры (N, r_v^c) и (N, r_v^d).

В ходе исследования игр с разрешенной структурой программно реализован алгоритм построения конъюнктивно- и дизъюнктивно-ограниченных игр на языке Java. Входными данными программы являются: число игроков N, характеристическая функция ТП-игры v и ориентированный граф D. Результатом работы являются соответствующие ограниченные игры (N, r_v^c) и (N, r_v^d). Код программы представлен в Приложении 1. Приведем пример работы программы на основе игры, рассмотренной выше.

Формат входных данных — файл формата txt, представленный на рисунке 2. В начале вводится количество игроков, далее граф игры в виде пары (i, j) через пробел, далее выигрыши коалиций, расположенных в лексикографическом порядке.

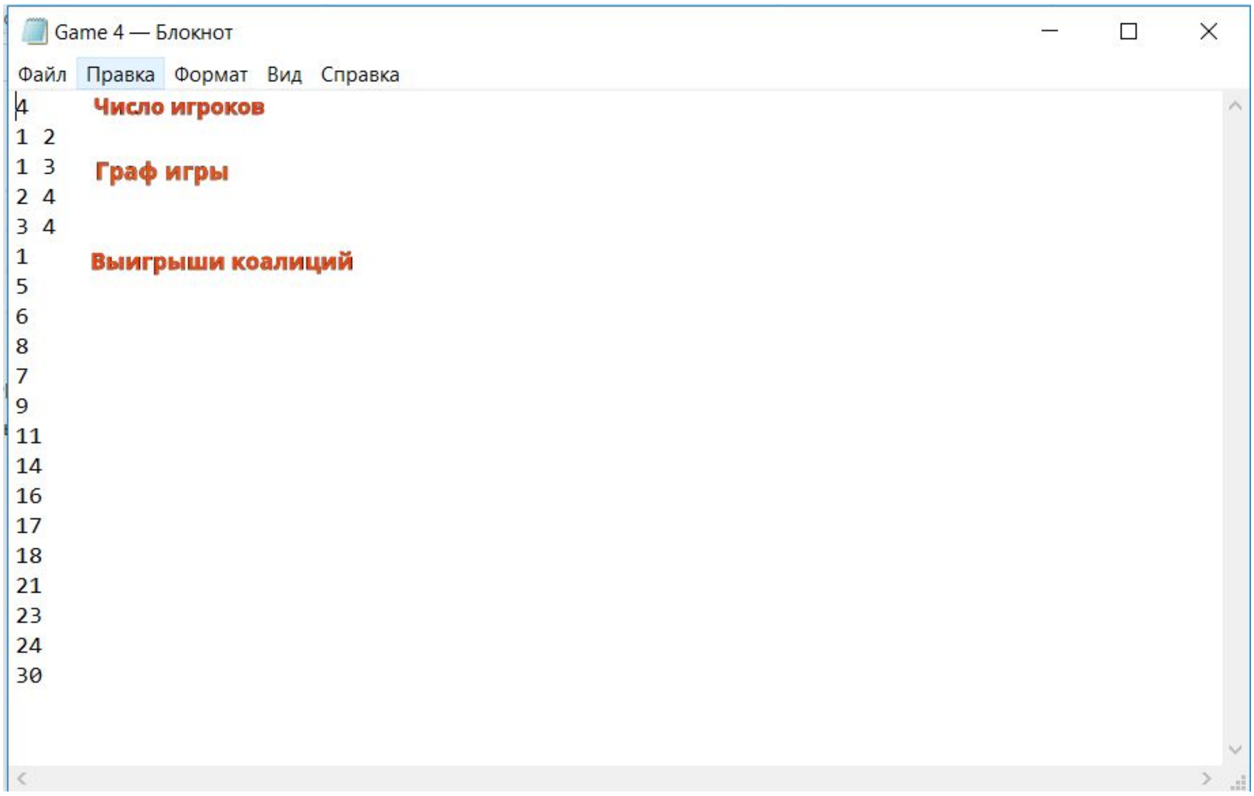


Рис. 2: Файл с входными данными

Выбор файла с данными производится из интерфейса программы посредством стандартных способов операционной системы (рис. 3).

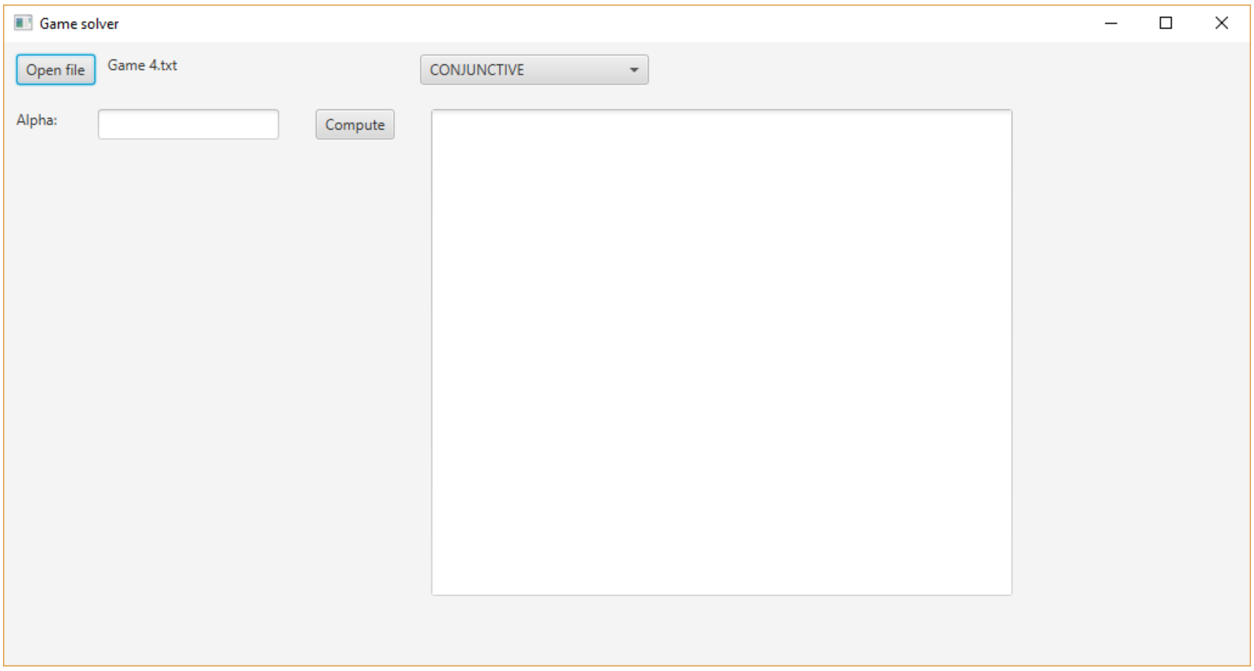


Рис. 3: Выбор файла с входными данными

В программе существует возможность выбора вычисления: построение игры с разрешенной структурой в конъюнктивном или дизъюнктивном подходах. Вычисленная игра с разрешенной структурой в конъюнктивном подходе представлена на рисунке 4.

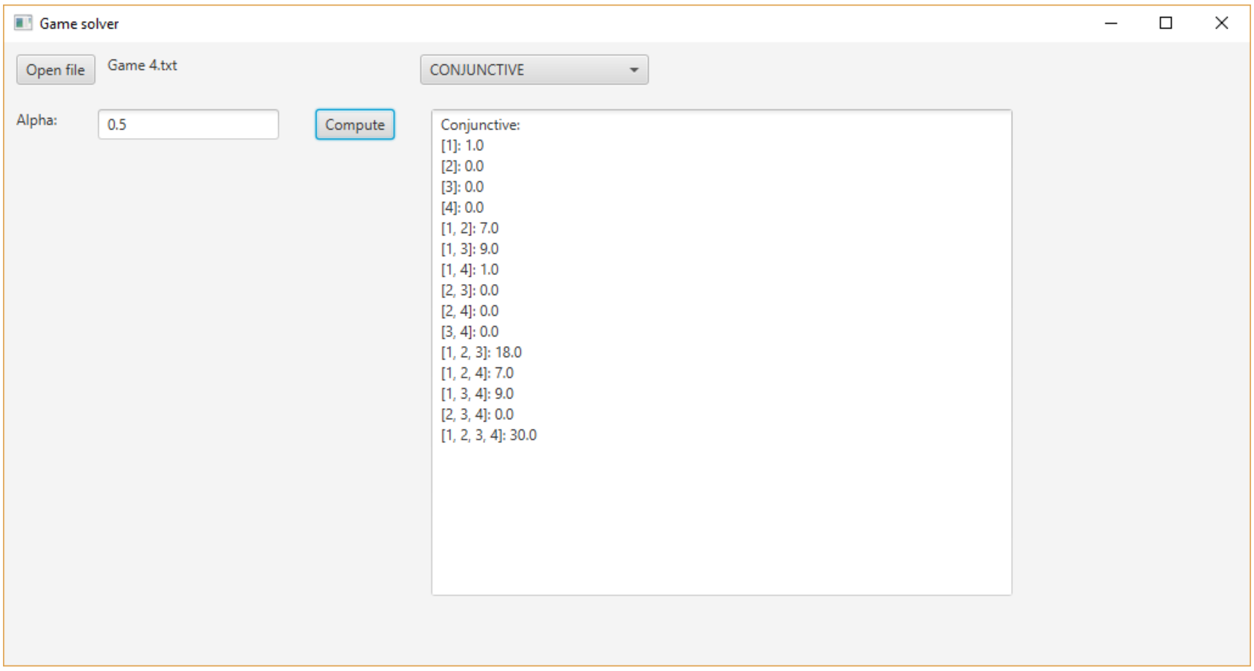


Рис. 4: Конъюнктивно-ограниченная игра (N, r_v^c)

Вычисленная игра с разрешенной структурой в дизъюнктивном подходе, представлена на рисунке 5.

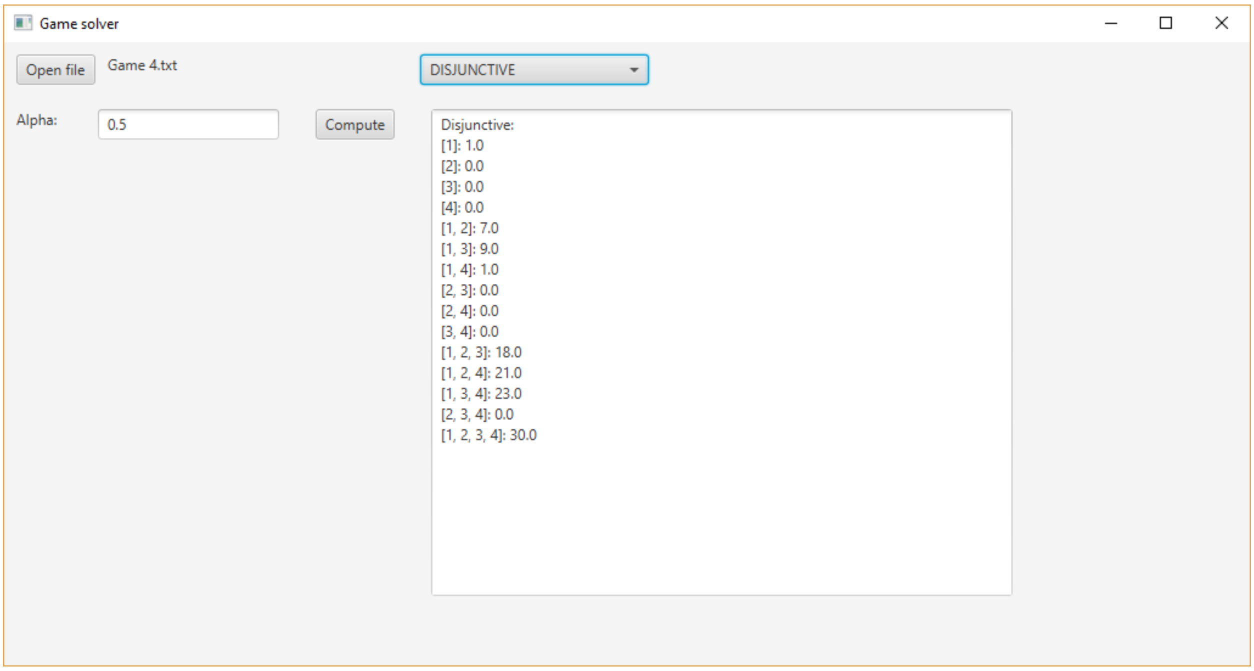


Рис. 5: Дизъюнктивно-ограниченная игра (N, r_v^d)

Таким образом, реализована программа, которая позволяет строить ограниченные игры для любого количества игроков.

3. Дизъюнктивно-ограниченная игра $(N, r^d_{\upsilon^\alpha})$

3.1. Способ построения игры $(N, r^d_{\upsilon^\alpha})$

Решения классических кооперативных игр имеют место и в играх с разрешенной структурой (N, υ, D). Для нахождения α-N-ядра в ограниченной игре (N, r^d_υ) будем опираться на теорему 2, согласно которой α-N-ядро равно пред-N-ядру в игре с характеристической функцией $\upsilon^\alpha(S)$. Поэтому рассмотрим построение игры с разрешенной структурой (N, υ^α, D).

Характеристическая функция υ^α задается формулой (5), где $\upsilon^*(S)$ вычисляется по формуле (4). Построим дизъюнктивно-ограниченную игру $(N, r^d_{\upsilon^\alpha})$, соотвествующую игре с разрешенной структурой (N, υ^α, D) двумя способами.

Первый способ

1) Построим дизъюнктивно-ограниченную игру (N, r^d_υ), соответствующую игре с разрешенной структурой (N, υ, D), по определению

$$r^d_\upsilon(S) = \upsilon(\sigma^d(S)) \text{ для всех } S \subseteq N.$$

2) Определим дизъюнктивно-ограниченную игру $(N, r^d_{\upsilon^*})$ для двойственной игры (N, υ^*) следующим образом

$$r^d_{\upsilon^*}(S) = \upsilon^*(\sigma^d(S)) = \upsilon(N) - \upsilon(N \setminus \sigma^d(S)) \text{ для всех } S \subseteq N.$$

3) Дизъюнктивно-ограниченную игру $(N, r^d_{\upsilon^\alpha})$, соответствующую игре с разрешенной структурой (N, υ^α, D), зададим как

$$r^d_{\upsilon^\alpha}(S) = \alpha \cdot r^d_\upsilon(S) + (1-\alpha) \cdot r^d_{\upsilon^*}(S), \text{ где } \alpha \in [0, 1].$$

Второй способ

1) Построим двойствую кооперативную игру (N, υ^*) к игре (N, υ).

2) Построим кооперативную игру (N, υ^α), где характеристическая функция игры вычисляется по формуле (5).

3) Построим дизъюнктивно-ограниченную игру $(N, r^d_{\upsilon^\alpha})$, соответсвующую игре с разрешенной структурой (N, υ^α, D), по определению

$$r^d_{\upsilon^\alpha}(S) = \upsilon^\alpha(\sigma^d(S)), \text{ для всех } S \subseteq N. \tag{6}$$

Утверждение 1. *Первый и второй способы построения дизъюнктивно-ограниченной игры* $(N, r^d_{\upsilon^\alpha})$ *эквивалентны.*

Доказательство. Подставляя вид характеристической функции $\upsilon^\alpha(S)$ в равенство (6), получим

$$r^d_{\upsilon^\alpha}(S) = \alpha\upsilon(\sigma^d(S)) + (1-\alpha)\upsilon^*(\sigma^d(S)) =$$

$$= \alpha \cdot r^d_\upsilon(S) + (1-\alpha) \cdot r^d_{\upsilon^*}(S), \text{ для всех } S \subseteq N.$$

При этом получаем формулу для вычисления характеристической функции $r^d_{\upsilon^\alpha}$

$$r^d_{\upsilon^\alpha} = \alpha\upsilon(\sigma^d(S)) + (1-\alpha)(\upsilon(N) - \upsilon(N \setminus \sigma^d(S)), \tag{7}$$

где $S \subseteq N,\ \alpha \in [0, 1]$. $\square$

Согласно утверждению 1 дизъюнктивно-ограниченная игра $(N, r^d_{\upsilon^\alpha})$ может быть построена путем взвешивания дизъюнктивно-ограниченных игр (N, r^d_υ) и $(N, r^d_{\upsilon^*})$ или путем построения кооперативной игры (N, υ^α) и соответствующей ей дизъюнктивно-ограниченной игры по определению 2.5. Первый способ вычисления удобно использовать в случае наличия посчитанных дизъюнктивно-ограниченных игр (N, r^d_υ) и $(N, r^d_{\upsilon^*})$, в других случаях удобнее использовать второй способ вычисления. Также результат утверждения 1 позволяет использовать формулу (7) для построения дизъюнктивно-ограниченной игры $(N, r^d_{\upsilon^\alpha})$, что делает алгоритм построения вычислительно проще, убирая промежуточные вычисления.

3.2. Пример программной реализации построения ограниченной игры $(N, r^d_{\upsilon^\alpha})$

Как было сказано ранее, согласно теореме 2 вычисление α-N-ядра в игре (N, r^d_υ) сводится к вычислению пред-N-ядра в игре $(N, r^d_{\upsilon^\alpha})$. Поэтому в ходе исследования игр с разрешенной структурой программно реализован алгоритм построения игры $(N, r^d_{\upsilon^\alpha})$ на языке Java. Входными данными программы являются: число игроков N, характеристическая функция ТП-игры υ, ориентированный граф D и число $\alpha \in [0,1]$. Результатом работы являются соответствующая дизъюнктивно-ограниченная игра $(N, r^d_{\upsilon^\alpha})$. Код программы представлен в Приложении 1.

Приведем пример работы программы. Рассмотрим игру четырех лиц, описанную в разделе 2.3. В программе реализована возможность выбора вычисления: построение двойственной игры, двойственной разрешенной игры в конъюнктивном и дизъюнктивном подходах, кооперативной игры (N, υ^α), ограниченных игр $(N, r^c_{\upsilon^\alpha})$ и $(N, r^d_{\upsilon^\alpha})$.

Результат построения кооперативной игры (N, υ^α), где $\alpha \in [0,1]$, представлен на рисунке 6.

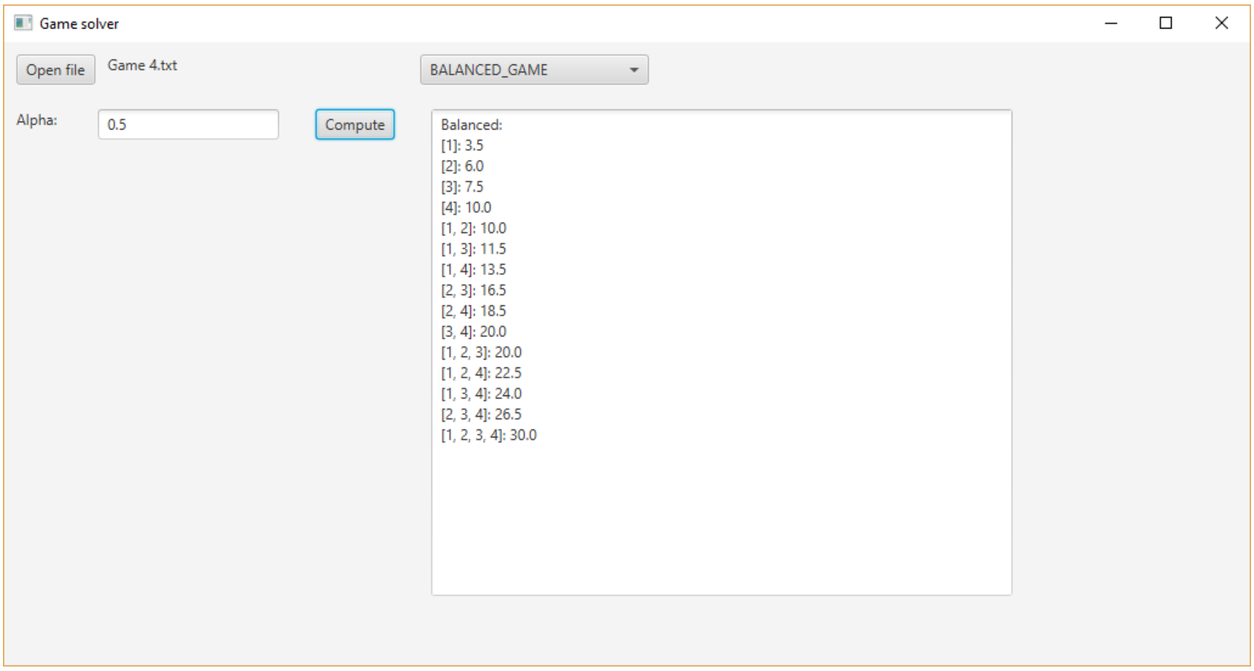


Рис. 6: Игра (N, υ^α)

Результат построения конъюнктивно-ограниченной игры $(N, r^c_{\upsilon^\alpha})$, соответствующей игре с разрешенной структурой (N, υ^α, D), представлен на рисунке 7.

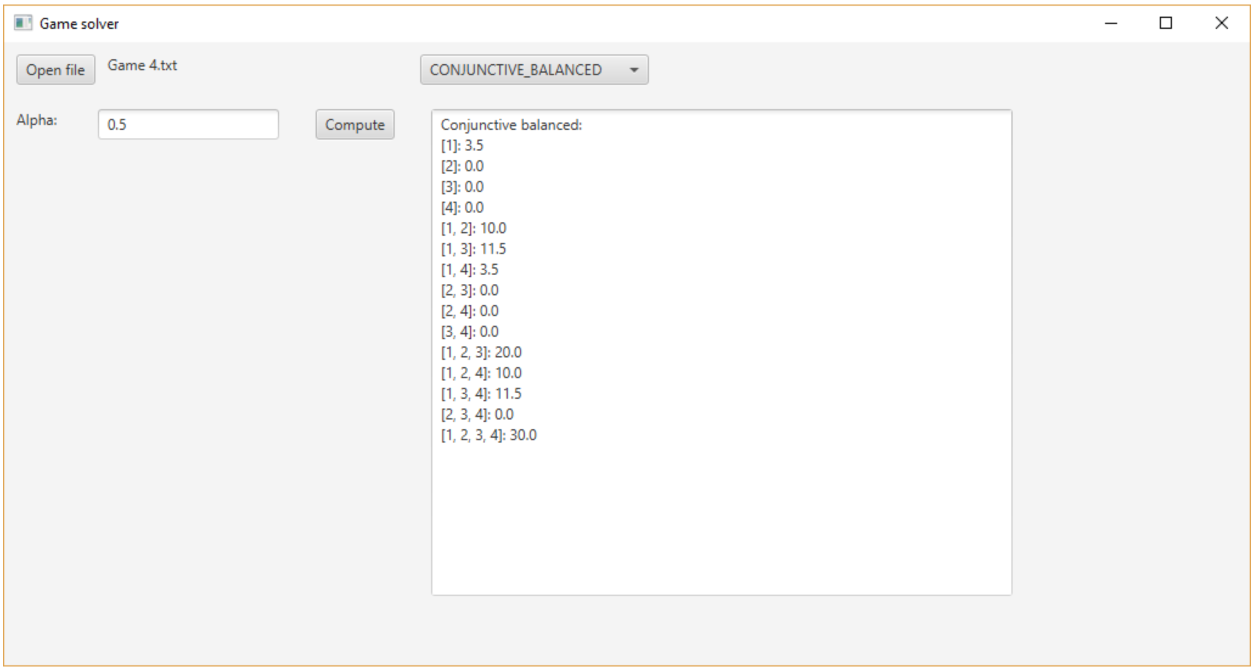


Рис. 7: Игра $(N, r^c_{\upsilon^\alpha})$

Результат построения дизъюнктивно-ограниченной игры $(N, r^d_{\upsilon^\alpha})$, представлен на рисунке 8.

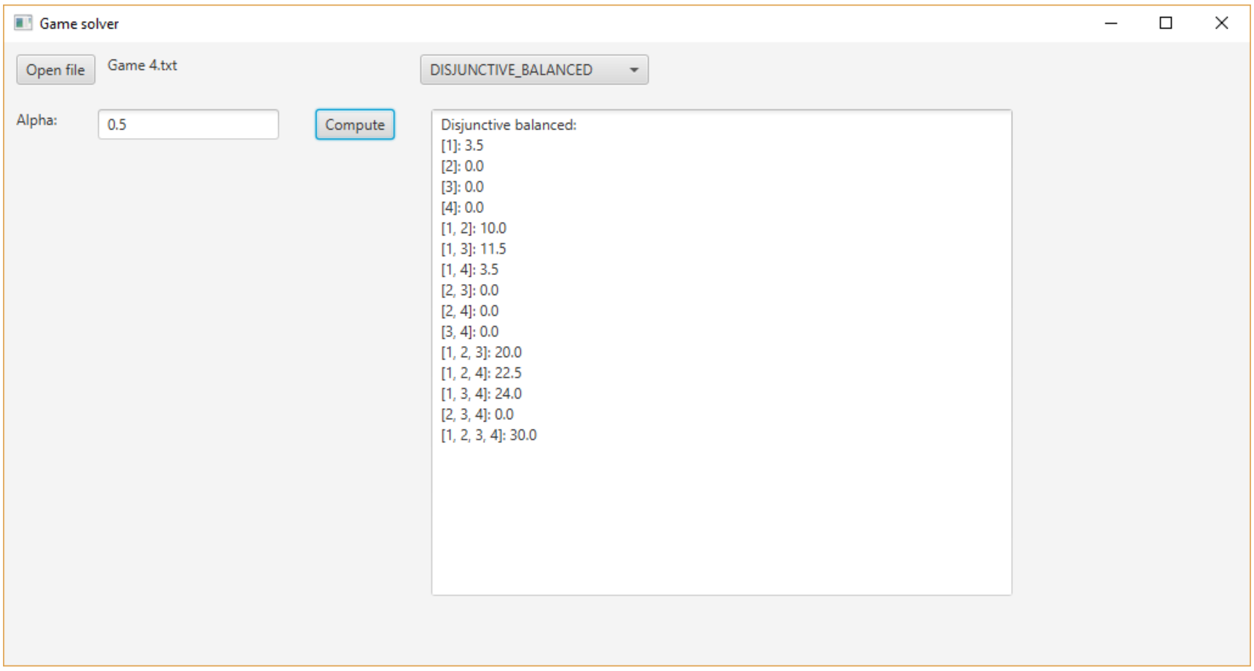


Рис. 8: Игра $(N, r^d_{\upsilon^\alpha})$

Таким образом, реализована программа построения кооперативной игры (N, υ^α), $\alpha \in [0, 1]$, и соответствующих ограниченных игр для N лиц.

4. Алгоритм вычисления пред-N-ядра

4.1. Свойства игры (N, v^α, D)

В работе [11] приведен алгоритм вычисления пред-N-ядра в дизъюнктивно-ограниченной игре (N, r_v^d), соотвествующей игре с разрешенной структурой (N, v, D). Данный алгоритм имеет полиномиальную сложность. Для его применения необходимо, чтобы игра (N, v, D) удовлетворяла свойствам слабой орграфной монотонности и слабой орграфной вогнутости.

Определение 4.1. *Будем говорить, что игра с разрешенной структурой (N, v, D) удовлетворяет свойству слабой орграфной монотонности, если выполняется неравенство*

$$v(S) \leqslant v(N) \text{ для всех } S \in \Phi^d. \tag{8}$$

Данное свойство означает, что в игре (N, v, D) ценность каждой возможной коалиции в дизъюнктивном подходе должна быть не больше ценности максимальной коалиции N. Данное свойство является более слабым, чем свойство монотонности классической кооперативной игры, так как неравенство (1) должно выполняться для $T = N$ и для всех коалиций из дизъюнктивно-возможного набора.

Определение 4.2. *Будем говорить, что игра с разрешенной структурой (N, v, D) удовлетворяет свойству слабой орграфной вогнутости, если для коалиций S, T, $S \cup T = N$, выполняется неравенство*

$$v(S) + v(T) \geqslant v(S \cap T) + v(N) \text{ для всех } S, T \in \Phi^d. \tag{9}$$

Данное определение означает, что свойство вогнутости классических кооперативных игр, т. е. неравенство (2) должно выполняться для дизъюнктивно-возможных коалиций, объединение которых есть максимальная коалиция N.

Проверим выполнение свойства слабой орграфной монотонности для игры с разрешенной структурой (N, v^α, D). Сформулировано и доказано следующее утверждение.

Утверждение 2. *Если игра (N, v, D) удовлетворяет свойству слабой ор-*

графной монотонности, а характеристическая функция $v(S)$ является неотрицательной, то игра (N, v^α, D) также удовлетворяет свойству слабой орграфной монотонности.

Доказательство. Рассмотрим игру с разрешенной структурой (N, v^α, D), где характеристическая функция $v^\alpha(S)$ задается в виде (5). Подставляя характеристическую функцию двойственной игры $v^*(S)$, вычисляемую по формуле (4), в представление функции v^α, получим

$$v^\alpha(S) = \alpha v(S) + v(N) - v(N \setminus S) - \alpha v(N) + \alpha v(N \setminus S).$$

Заметим, что

$$v^\alpha(N) = \alpha v(N) + v(N) - \alpha v(N) = v(N).$$

Для игры (N, v^α, D) проверим свойство слабой орграфной монотонности. Для этого необходимо показать, что для всех дизъюнктивно-возможных коалиций выполняется неравенство (8). Рассмотрим разность

$$v^\alpha(N) - v^\alpha(S) = v(N) - \alpha v(S) - v(N) + v(N \setminus S) + \alpha v(N) - \alpha v(N \setminus S) =$$

$$= -\alpha v(S) + v(N \setminus S) + \alpha v(N) - \alpha v(N \setminus S) = \alpha(v(N) - v(S)) + (1-\alpha)v(N \setminus S). \tag{10}$$

Оценим слагаемые в данном равенстве. Так как игра с разрешенной структурой (N, v, D) по условию теоремы удовлетворяет свойству слабо орграфно-монотонности, то $v(N) - v(S) \geqslant 0$ для всех $S \in \Phi^d$. Также по условию $\alpha \in [0, 1]$, следовательно, слагаемое $\alpha(v(N) - v(S)))$ не меньше нуля. Так как $\alpha \in [0, 1]$, то $(1 - \alpha) \geqslant 0$, а $v(N \setminus S)$ неотрицательно по условию, следовательно $(1-\alpha)v(N \setminus S)$ также не меньше нуля. Таким образом сумма двух слагаемых в (10) не меньше нуля, т. е. $v^\alpha(S) \leqslant v^\alpha(N)$, а значит игра с разрешенной структурой (N, v^α, D) удовлетворяет свойству слабой орграфной монотонности.

□

Согласно данному результату в алгоритме вычисления пред-N-ядра в игре (N, v^α, D) достаточно проверять функцию v на свойство слабой орграфной монотонности и на неотрицательность. Свойство вогнутости в программной реализации алгоритма будем проверять для функции v^{α}.

4.2. Существенные и возможные коалиции

В данном разделе приведем основные утверждения, которые легли в основу алгоритма вычисления пред-N-ядра в игре (N, r_v^d). Результаты сформулированы как в работе [11].

Лемма 3. *Пусть (N, v, D) игра с разрешенной структурой. Если $S \subseteq N$, где $\mid S \mid \geqslant 2$, существенная коалиция в ограниченной игре (N, r_v^d), то S является возможной коалицией.*

Согласно первой лемме любая существенная коалиция, содержащая по крайней мере двух игроков, является возможной.

Введем для коалиции $S \subset N$ и ограниченной игры (N, r_v^d) следующую величину

$$\tau(S, r_v^d) = \frac{r_v^d(N) - r_v^d(S)}{\mid N \setminus S \mid +1}. \tag{11}$$

В дальнейшем будем рассматривать набор дизъюнктивно-возможных коалиций без максимальной коалиции, т. е. набор $\Omega^d = \Phi^d \setminus \{N\}$.

Лемма 4. *Пусть (N, v, D) игра с разрешенной структурой. Если (N, v, D) удовлетворяет свойству слабой орграфной монотонности, тогда*

$$e^*(N, r_v^d) = \min_{S \in \Omega^d} \tau(S, r_v^d).$$

Лемма 5. *Пусть игра с разрешенной структурой (N, v, D) удовлетворяет свойству слабой орграфной монотонности. Рассмотрим коалицию $U \in \Omega^d$, для которой выполняется $\tau(U, r_v^d) = e^*(N, r_v^d)$, и вектор $y \in R^n$, для которого выполняются условия: $y(U) = r(U) + \tau(U, r)$ и $y_j = \tau(U, r_v^d)$ для всех $j \notin U$. Тогда $x = \nu(N, r_v^d)$ удовлетворяет $x(U) = y(U)$ и $x_j = y_j$ для всех $j \notin U$.*

Данные результаты дают основную идею для алгоритма вычисления пред-N-ядра. Согласно лемме, коалиция U из набора Ω^d, для которой величина $\tau(U, r_v^d)$ является минимальной, есть ключевая коалиция в том смысле, что можно найти дележ, соответствующий пред-N-ядру, игроков вне данной коалиции, т. е. игроков $j \notin U$, и он будет равен $\tau(U, r_v^d)$. В дальнейшем будем использовать обозначение $\tau^*(r_v^d) = e^*(N, r_v^d)$. На первом шаге алгоритма будем искать коалицию $U_1 \in \Omega^d$, удовлетворяющую условию

$$\tau(U_1, r_v^d) = \tau^*(r_v^d) \text{ и } \mid U_1 \mid = \max_{\{U \in \Omega^d \mid \tau(U, r_v^d) = \tau^*(r_v^d)\}} \mid U \mid . \tag{12}$$

Таким образом будет определена коалиция $U_1 \neq N$, принадлежащая набору дизъюнктивно-возможных коалиций Ω^d, максимальной размерности, для которой величина (11) будет минимальной. На этом же шаге всем игрокам $j \notin U_1$ пред-N-ядро предписывает выигрыш равный $\tau^*(r_v^d) = \tau(U_1, r_v^d)$, и осуществляется переход к следующему шагу алгоритма. На втором шаге рассматривается игра с разрешенной структурой (U_1, v_1, D_1), где (U_1, v_1) новая кооперативная игра, (U_1, D_1) — новый орграф. Подробные шаги алгоритма описаны в следующем разделе.

4.3. Алгоритм

Приведем алгоритм, позволяющий вычислять пред-N-ядро в дизъюнктивно-ограниченной игре (N, r_v^d).

Шаг 1

Пусть $k = 0$. Рассматриваем игру с разрешенной структурой (N, v, D). На данном шаге $U_0 = N, v_0 = v, D_0 = D$ и $r_0 = r$. Переходим к шагу 2.

Шаг 2

Находим множество $U_{k+1} \subset U_k$, удовлетворяющее условиям (12) и соответствующее игре с разрешенной структурой (U_k, v_k, D_k), т. е.

$$\tau(U_{k+1}, r_{v_k}^d) = \tau^*(r_{v_k}^d) \text{ и } \mid U_{k+1} \mid = \max_{\{U \in \Omega^d \mid \tau(U, r_{v_k}^d) = \tau^*(r_{v_k}^d)\}} \mid U \mid,$$

где $\tau^*(r_{v_k}^d) = \min\limits_{U \in \Omega^d} \tau(U, r_{v_k}^d),\ \tau(U, r_{v_k}^d) = \frac{r_{v_k}^d(U_k) - r_{v_k}^d(U)}{|U_k \setminus U| + 1}$.

Каждому игроку из множества $U_k \setminus U_{k+1}$ определяется выигрыш $x_j = \tau^*(r_{v_k}^d)$. Переходим к следующей итерации.

Шаг 3

Если $U_{k+1} = 1$ — первый игрок, тогда переходим к шагу 4.
Иначе, пусть i_{k+1} корневая вершина подграфа $(U_k \setminus U_{k+1}, D_k(U_k \setminus U_{k+1}))$ соответствующего орграфа (U_k, D_k). Определим новую игру с разрешенной структурой $(U_{k+1}, v_{k+1}, D_{k+1})$, где кооперативная игра $(U_{k+1}, v_{k+1}), U \subseteq$

U_{k+1}, вычисляется следующим образом

$$\upsilon_{k+1}(U) = \begin{cases} \upsilon_k(U), \text{ если } P_{D_k}(i_{k+1}) \cap U = \emptyset, \\ \upsilon_k(U \cup (U_k \setminus U_{k+1})) - \tau(U_{k+1}, r^d_{\upsilon_k}) \cdot \mid U_k \setminus U_{k+1} \mid, \quad \text{иначе.} \end{cases}$$

Орграф (U_{k+1}, D_{k+1}) новой игры определяется как

$$(i,j) \in D_{k+1}, \text{ если } \left[\begin{array}{cr} (i,j) \in D_k & \text{или,} \\ i \in P_{D_k}(i_{k+1}) \text{ и } j \in S_{D_k}(U_k \setminus U_{k+1}) \cap U_{k+1}. & \end{array} \right.$$

Далее вычисляется дизъюнктивно-ограниченная игра $r^d_{\upsilon_{k+1}}$ соответсвующая игре с разрешенной структурой $(U_{k+1}, \upsilon_{k+1}, D_{k+1})$, построенной на этом шаге. Пусть $k = k+1$, переходим к шагу 2.

Шаг 4 Вычислим выигрыш первого игрока

$$x_1 = \upsilon(N) - \sum_{j \in N \setminus \{1\}} x_j.$$

Алгоритм останавливается.

Следующий результат [11] показывает, что орграф новой игры с разрешенной структурой будет удовлетворять пункту 1 предположения.

Лемма 6. *Пусть $U_{k+1} = \{1\}$, тогда для любого $k = 0, 1, \ldots, K-1$ орграф (U_{k+1}, D_{k+1}) удовлетворяет пункту 1 предположения.*

Согласно следующему утверждению, вектор, полученный в результате применения алгоритма, будет являться пред-N-ядром.

Теорема 4. *Пусть игра (N, υ, D) удовлетворяет свойству слабой орграфной монотонности и слабой орграфной вогнутости, тогда алгоритм дает пред-N-ядро игры (N, r^d_υ).*

4.4. Пример программной реализации алгоритма вычисления α-N-ядра

Результатом данной работы является программная реализация алгоритма вычисления пред-N-ядра в игре $(N, r^d_{\upsilon^\alpha})$, приведенного в разделе 4.3. Данный подход позволяет вычислять α-N-ядро в дизъюнктивно-ограниченной игре (N, r_v^d). Код программы представлен в Приложении 1. Приведем пример, иллюстрирующий работу программы.

Рассмотрим игру с разрешенной структурой пяти лиц, $N = \{1, 2, 3, 4, 5\}$. Орграф игры, представлен на рисунке 9.

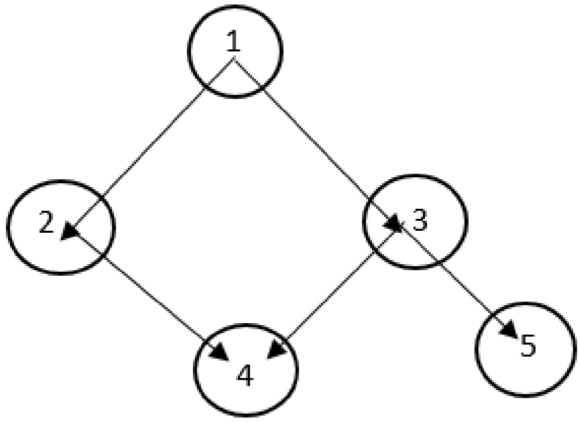


Рис. 9: Орграф игры (N, v, D)

Кооперативная игра (N, v) задается в виде:

$$v(\{1\}) = 1,\ v(\{2\}) = 2,\ v(\{3\}) = 0,\ v(\{4\}) = 3,\ v(\{5\}) = 4,$$

$$v(\{12\}) = 3,\ v(\{13\}) = 1,\ v(\{14\}) = 5,\ v(\{15\}) = 5,$$

$$v(\{23\}) = 3, v(\{24\}) = 5, v(\{25\}) = 8, v(\{34\}) = 4, v(\{35\}) = 6, v(\{45\}) = 10,$$

$$v(\{123\}) = 3,\ v(\{124\}) = 10,\ v(\{125\}) = 9,\ v(\{134\}) = 9,\ v(\{135\}) = 8,$$

$$v(\{145\}) = 11,\ v(\{234\}) = 8,\ v(\{235\}) = 10,\ v(\{245\}) = 12,\ v(\{345\}) = 11,$$

$$v(\{1234\}) = 13,\ v(\{1235\}) = 13,\ v(\{1245\}) = 13,\ v(\{1345\}) = 13$$

$$v(\{12345\}) = 13.$$

Ввод в программу осуществляется с помощью файла формата txt (рисунок 10).

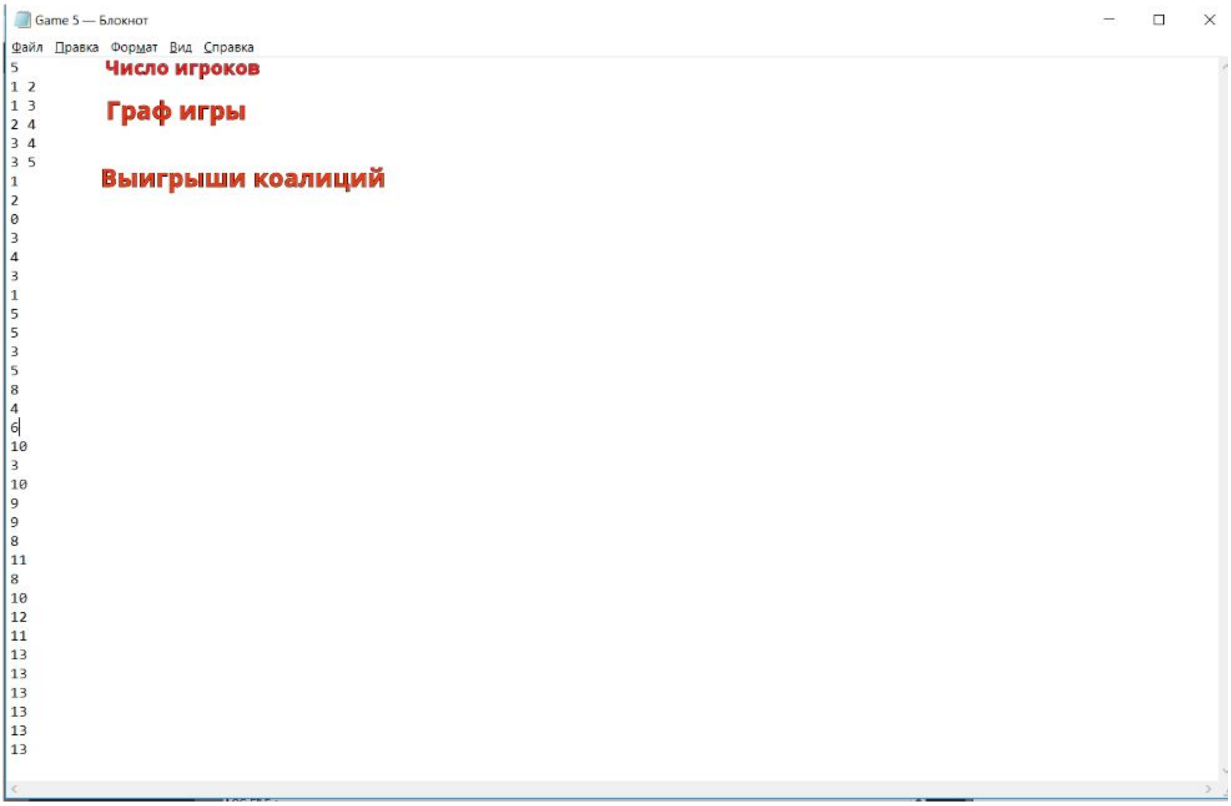


Рис. 10: Файл с входными данными

В начале программы проверяется свойство слабой орграфной монотонности и неотрицательность для функции $v(S)$. Далее вычисляется игра с разрешенной структурой (N, v^{α}, D) (кооперативная игра (N, v^{α})), для которой проверяется выполнение свойства слабой орграфной вогнутости. Если данные свойства не выполняются, то в интерфейсе программы выводится сообщение «Function doesn't satisfy required properties». Пример на рисунке 11.

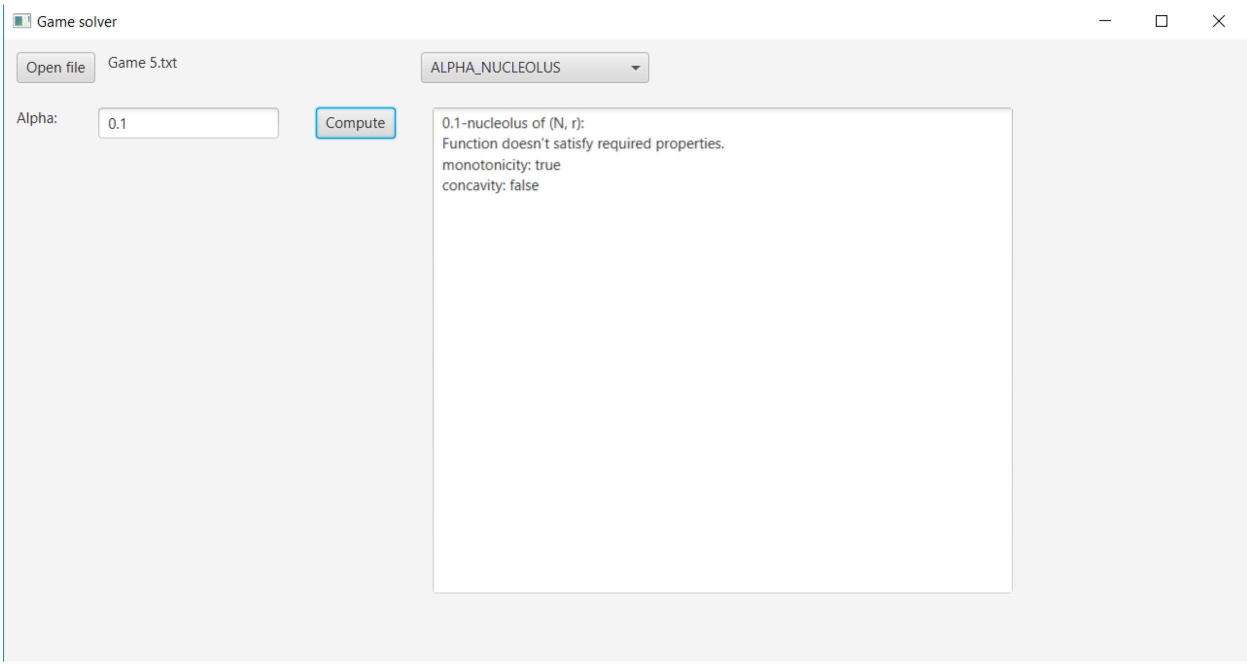


Рис. 11: Сообщение об ошибке программы

Если игра удовлетворяет данным свойствам, то осуществляется переход к шагу 1 алгоритма вычисления пред-N-ядра. На рисунке 12 представлено вычисление SM-ядра в программе.

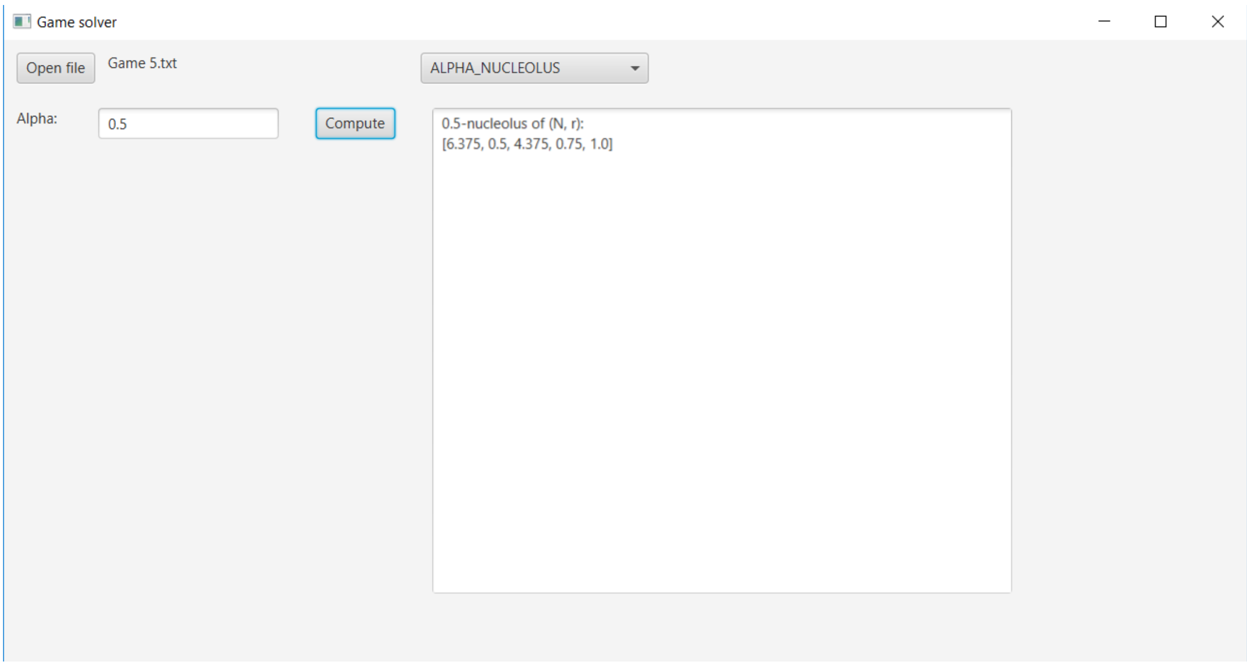


Рис. 12: Вычисление SM-ядра

Далее приведена таблица 1 вычисленных α-N-ядер, $\alpha \in [0, 1]$, в игре (N, r_v^d) с помощью программы.

Таблица 1: α-N-ядра в игре (N, r_v^d)

Значение α	α-N-**ядро**
1	$(8, 5, 0, 0, 0)$
$0, 9$	$(7.7, 0.1, 4.9, 0.1, 0.2)$
$0, 8$	$(7.4, 0.2, 4.7, 0.3, 0.4)$
$0, 7$	$(7, 0.3, 4.6, 0.5, 0.6)$
$0, 6$	$(6.7, 0.4, 4.5, 0.6, 0.8)$
$0, 5$	$(6.4, 0.5, 4.4, 0.7, 1)$

Эмпирически получен следующий результат. При $\alpha \in [\frac{1}{2}, 1]$ игра с разрешенной структурой (N, v^α, D) удовлетворяет свойству слабой орграфной выпуклости, при $\alpha \in [0, \frac{1}{2}]$ данное свойство не выполняется (рисунок 13). Данное утверждение было получено после нахождения α-N-ядер с помощью программы для двадцати примеров.

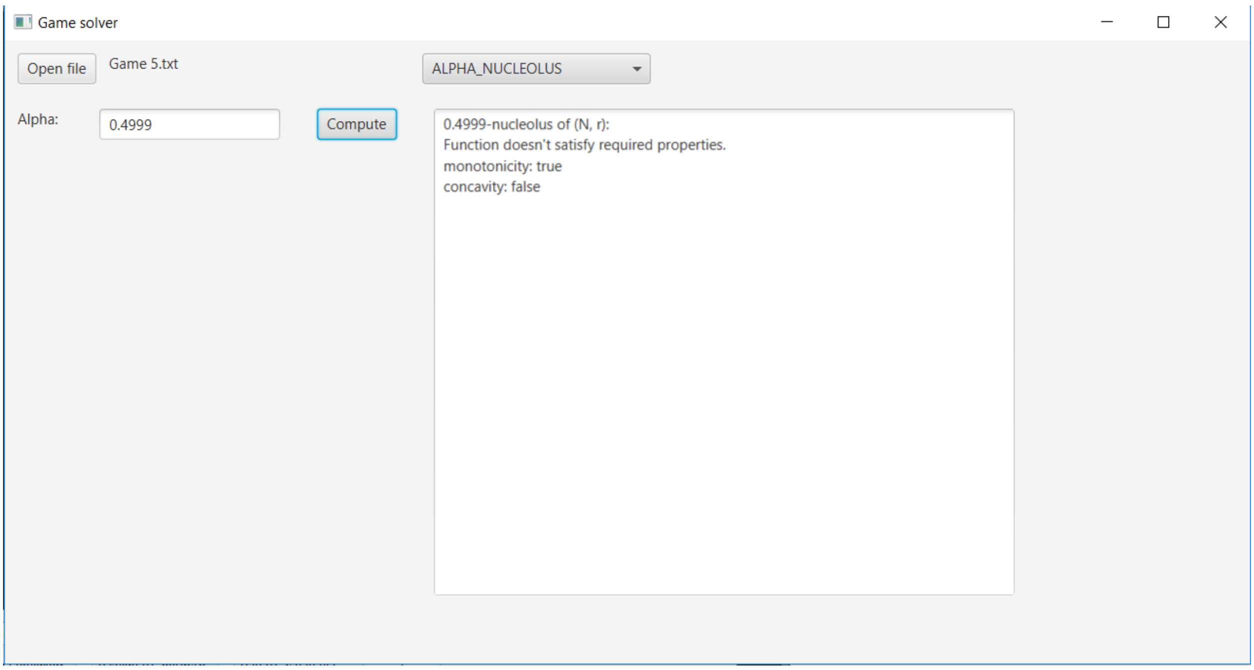
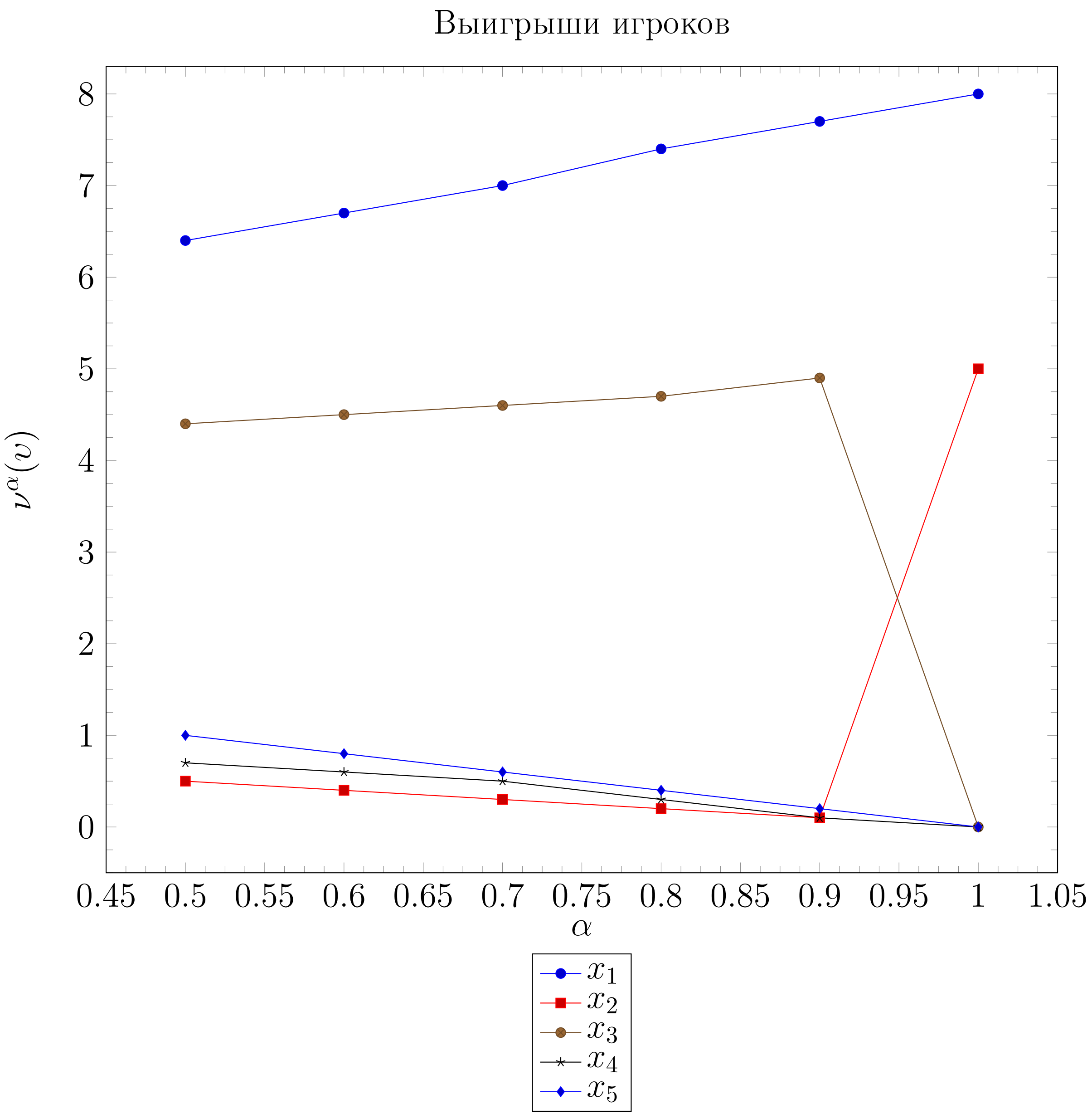


Рис. 13: Ошибка выполнения свойства слабой ографной вогнутости для игры с разрешенной структурой (N, v^α, D)

Таким образом, возникает вопрос о слабой орграфной вогнутости игры с разрешенной структурой (N, v^α, D), который является ключевым для дальнейших исследований.

Проанализируем полученные решения. При уменьшении значения α от 1 к $\frac{1}{2}$ выигрыш первого игрока уменьшается незначительно, в отличие от второго игрока, выигрыш которого резко уменьшается при переходе от $\alpha = 1$ к $\alpha = 0,9$, а далее выигрыш второго игрока начинает расти на $0,1$ при увеличении α на $0,1$. Выигрыш третьего игрока имеет соверешенно противоположную тенденцию: пред-N-ядро приписывает выигрыш равный 0, $0,9$-N-ядро дает выигрыш $4,9$, а далее выгрыш уменьшается при уменьшении значения α. Выигрыши четвертого и пятого игроков увеличиваются при уменьшении значения α. Изменение выигрышей игроков проиллюстрировано на графике.



5. Заключение

В работе представлена игра с разрешенной структурой (N, υ, D), которая позволяет учитывать иерархию игроков и ограничения на коммуникативные способности участников кооперации. Также рассмотрено два подхода к построению таких игр: конъюнктивный и дизъюнктивный. В качестве решения игры с разрешенной струтурой (N, υ, D) выбрано α-N-ядро, $\alpha \in [0, 1]$, из-за его интересных свойств: оно учитывает как конструктивную, так и блокирующую силу коалиций в игре.

В работе предложен метод построения двойственной игры с разрешенной структурой (N, υ^*, D). Также получена и доказана теорема о том, что два способа построения дизъюнктивно-ограниченной игры $(N, r^d_{\upsilon^\alpha})$ эквивалентны.

В ходе исследования было доказано, что игра с разрешенной структурой (N, υ^α, D) удовлетворяет свойству слабой орграфной монотонности.

Для нахождения α-N-ядра, $\alpha \in [0, 1]$, в дизъюнктивно-ограниченной игре (N, r^d) реализован алгоритм нахождения пред-N-ядра игры $(N, r^d_{\upsilon^\alpha})$.

Также экспериментально найдены интервалы для α, на которых для игры с разрешенной структурой (N, υ^α, D) выполняется свойство слабой орграфной вогнутости. Также вопрос вогнутости игры с разрешенной структурой (N, υ^α, D) планируется рассмотреть при дальнейшем исследовании.

Результаты данной работы можно использовать для игры загрязнения рек.

Список литературы

[1] Shapley L. S. A value for n-person games // Contributions to the theory of games, II. — Princeton: Princeton Univ. Press, 1953. P. 307–317.

[2] Смирнова Н. В., Тарашнина С. И. Об одном обобщении N-ядра в кооперативных играх // Дискретный анализ и исследование операций. 2011. Т. 18. №4. С. 77–93.

[3] Tarachnina S. The simplified modified nucleolus of a cooperative TU-game // TOP. 2011. V. 19 (1). P. 150–166.

[4] Смирнова Н. В., Тарашнина С. И. Геометрические свойства $[0,1]$-N-ядра в кооперативных ТП-играх // Математическая теория игр и её приложения. 2012. №1. С. 55–73.

[5] Печерский С Л., Яновская Е. Б. Кооперативные игры: решения и аксиомы. СПб.: Европейский университет в Санкт-Петербурге, 2004. С. 460

[6] Maschler M., Peleg. B, and Shapley L. S. Geometric properties of the kernel, nucleolus and related solution concepts // Mathematics of operations research. 1979. V. 4. P. 303–338.

[7] Schmeidler D. The nucleolus of characteristic function game // SIAM J. Appl. Math. 1969. V. 17. P. 1163–1170.

[8] Kohlberg E. On the nucleolus of a characteristic function game // SIAM Journal on Applied Mathematics. 1971. V. 20. P. 62–66.

[9] Huberman G. The nucleolus and essential coalitions // In: Bensoussan A, Lions J (eds)Analysis and optimization of system. Lecture notes in control and information sciences. V. 28. Springer, Berlin, P. 416–422.

[10] Arin J., Innara E. A characterization of the nucleolus for convex games. // Games Econ Behav 23. P. 12–24.

[11] Van Den Brink R., Katsev I., Van Der Laan G. A polynomial tine algorithm for computing the nucleolus for a class of disjunctive games with permission structure. // Int J Game Theory 40. P. 591–616.

[12] Van Den Brink R. Games with a permission structure - A survey on generalization and applications. // TOP. 2017. V. 25. P. 1–33.

[13] Gilles P., Owen. G. // Cooperative games and disjunctive permission structures. // Department of economics.

6. Приложение 1

```java
package entity;

import java.util.*;

public class Coalition implements Comparable<Coalition> {
  public final List<Player> players = new ArrayList<>();

  public Coalition copy(Map<Integer, Player>
     newPlayerMap) {
    Coalition coalition = new Coalition();
    for (Player player : players) {
      coalition.addPlayer(newPlayerMap.get(player.key));
    }
    return coalition;
  }

  public void addPlayer(Player player) {
    players.add(player);
  }

  public void removePlayer(Player player) {
    players.remove(player);
  }

  public List<Player> getPlayers() {
    return new ArrayList<>(players);
  }

  public boolean contains(Player player) {
    return players.contains(player);
  }

  public boolean contains(List<Player> players) {
```

```java
    for (Player player : players) {
      if (!contains(player)) {
        return false;
      }
    }
    return true;
  }

  public boolean containsIn(List<Player> players) {
    for (Player player : this.players) {
      if (!players.contains(player)) {
        return false;
      }
    }
    return true;
  }

  public boolean strongContains(List<Player> players) {
    if (this.players.size() != players.size()) return
       false;

    return contains(players);
  }

  public boolean isEmpty() {
    return players.isEmpty();
  }

  public List<Player> computeDiff(Coalition coalition) {
    List<Player> result = new ArrayList<>();
    for (Player player : this.players) {
      if (!coalition.players.contains(player)) {
        result.add(player);
      }
    }
```

```java
        return result;
    }

    public String getTitle() {
        StringBuilder builder = new StringBuilder();
        builder.append("{");
//      Collections.sort(players);
        for (int i = 0; i < players.size(); i++) {
            Player player = players.get(i);
            builder.append(player.key);
            if (i != players.size() - 1) {
                builder.append(", ");
            }
        }
        builder.append("}");
        return builder.toString();
    }

    @Override
    public String toString() {
        return players.toString();
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return
            false;
        Coalition coalition = (Coalition) o;
        return Objects.equals(players, coalition.players);
    }

    @Override
    public int hashCode() {
        return Objects.hash(players);
```

```java
  }

  @Override
  public int compareTo(Coalition coalition) {
    int compare = Integer.compare(players.size(), coalition.players.size());
    if (compare == 0) {
      for (int i = 0; i < players.size(); i++) {
        int firstKey = players.get(i).key;
        int secondKey = coalition.players.get(i).key;
        if (firstKey != secondKey) {
          return Integer.compare(firstKey, secondKey);
        }
      }
    }
    return compare;
  }
}
package entity;

import java.util.*;

public class CoalitionUtils {
  public static List<Coalition> createCoalitions(Player[] players) {
    List<Coalition> coalitions = new ArrayList<>();
    for (int i = 1; i < Math.pow(2, players.length); i++) {
      createCoalition(i, players, coalitions);
    }
    return coalitions;
  }

  private static void createCoalition(int n, Player[] players, List<Coalition> result) {
```

```java
      Coalition coalition = new Coalition();
      for (int i = 0; i < players.length; i++) {
        if (((n >> i) & 1) == 1) {
          coalition.addPlayer(players[i]);
        }
      }
      result.add(coalition);
    }
  }
package entity;

import java.util.*;

public class Game {
  public Player[] players;
  public final Map<Coalition, Float> coalitionsValue =
     new HashMap<>();
  public Coalition coalitionWithAllPlayers;
  private float balancedParam;

  public Game(Player[] players) {
    this.players = players;
  }

  public Game copy() {
    Player[] players = new Player[this.players.length];
    for (int i = 0; i < players.length; i++) {
      players[i] = this.players[i].copy();
    }
    Game result = new Game(players);
    Map<Integer, Player> copyPlayersMap = new
       HashMap<>();
    for (Player player : players) {
      copyPlayersMap.put(player.key, player);
    }
```

```java
    for (Coalition coalition : coalitionsValue.keySet()) {
      result.putCoalition(coalition.copy(copyPlayersMap), coalitionsValue.get(coalition));
    }
    result.coalitionWithAllPlayers = coalitionWithAllPlayers.copy(copyPlayersMap);
    result.balancedParam = balancedParam;
    return result;
  }

  public void putCoalition(Coalition coalition, float value) {
    coalitionsValue.put(coalition, value);
    if (coalition.players.size() == players.length) {
      coalitionWithAllPlayers = coalition;
    }
  }

  public void setBalancedParam(float balancedParam) {
    this.balancedParam = balancedParam;
  }

  public Game computeBalancedGame() {
    Game dualGame = computeDualGame();
    Game balancedGame = new Game(Arrays.copyOf(players, players.length));
    for (Coalition coalition : coalitionsValue.keySet()) {
      float coalitionValue = balancedParam * coalitionsValue.get(coalition)
          + (1 - balancedParam) * dualGame.coalitionsValue.get(coalition);
      balancedGame.putCoalition(coalition, coalitionValue);
```

```java
    }
    return balancedGame;
}

public Game computeDualGame() {
  Game dualGame = new Game(Arrays.copyOf(players, players.length));
  for (Coalition coalition : coalitionsValue.keySet()) {
    dualGame.putCoalition(coalition, computeDualValue(coalition));
  }
  return dualGame;
}

private float computeDualValue(Coalition coalition) {
  Coalition dualCoalition = computeDualCoalition(coalition);
  if (dualCoalition == null) {
    return coalitionsValue.get(coalition);
  }
  return coalitionsValue.get(coalitionWithAllPlayers) - coalitionsValue.get(dualCoalition);
}

private Coalition computeDualCoalition(Coalition coalition) {
  for (Coalition coalition1 : coalitionsValue.keySet()) {
    if (isDual(coalition, coalition1)) {
      return coalition1;
    }
  }
  return null;
}
```

```java

    private boolean isDual(Coalition coalition, Coalition coalition1) {
        List<Player> players = coalition.getPlayers();
        List<Player> players1 = coalition1.getPlayers();
        if (players.size() + players1.size() != this.players.length) return false;

        for (Player player : players) {
            for (Player player1 : players1) {
                if (player == player1) {
                    return false;
                }
            }
        }
        return true;
    }

    public boolean checkMonotonic() {
//        System.out.println("values: " + coalitionsValue);
        for (Coalition coalition : coalitionsValue.keySet()) {
            if (coalition == coalitionWithAllPlayers) continue;
            if (coalitionsValue.get(coalitionWithAllPlayers) < coalitionsValue.get(coalition)) {
                System.out.println("all players: " + coalitionWithAllPlayers);
                System.out.println("value: " + coalitionsValue.get(coalitionWithAllPlayers));
                System.out.println("coalition: " + coalition);
                System.out.println("value: " + coalitionsValue.get(coalition));
                return false;
            }
        }
```

```java
    return true;
  }

  public boolean checkConcave() {
    for (Coalition coalition : coalitionsValue.keySet()) {
      for (Coalition coalition1 : coalitionsValue.keySet()) {
        if (coalition == coalition1) continue;

        if (!isFullPlayersSet(coalition1.getPlayers(), coalition.getPlayers())) continue;

        Coalition intersection = computeIntersection(coalition1, coalition);
        float intersectionValue = intersection == null ? 0 : coalitionsValue.get(intersection);


        boolean b = coalitionsValue.get(coalition) + coalitionsValue.get(coalition1)
                >= coalitionsValue.get(coalitionWithAllPlayers) + intersectionValue;
        if (!b) {
          return false;
        }
      }
    }
    return true;
  }

  private Coalition computeIntersection(Coalition coalition1, Coalition coalition2) {
    List<Player> players1 = coalition1.getPlayers();
```

```
    List<Player> players2 = coalition2.getPlayers();
    Coalition intersection = new Coalition();
    for (Player player1 : players1) {
      for (Player player2 : players2) {
        if (player1 == player2) {
          intersection.addPlayer(player1);
        }
      }
    }
    for (Coalition coalition : coalitionsValue.keySet())
       {
      if (coalition.equals(intersection)) {
        return coalition;
      }
    }
    return null;
  }

  private boolean isFullPlayersSet(List<Player>
     players1, List<Player> players2) {
    forI:
    for (Player player : players) {
      for (Player player1 : players1) {
        if (player == player1) {
          continue forI;
        }
      }
      for (Player player2 : players2) {
        if (player == player2) {
          continue forI;
        }
      }
      return false;
    }
    return true;
```

```
    }

    public Game computeConjunctiveGame() {
        List<Coalition> conjunctiveCoalitions = computeConjunctiveCoalitions();
        Map<Coalition, Coalition> coalitionMap = new HashMap<>();
        for (Coalition coalition : coalitionsValue.keySet()) {
            Coalition bestCompareCoalition = findBestCompareCoalition(coalition, conjunctiveCoalitions);
            coalitionMap.put(coalition, bestCompareCoalition);
        }
        Game disjunctiveGame = new Game(Arrays.copyOf(players, players.length));
        for (Coalition coalition : coalitionMap.keySet()) {
            Coalition key = coalitionMap.get(coalition);
            Float value = key.isEmpty() ? 0f : coalitionsValue.get(key);
            disjunctiveGame.putCoalition(coalition, value);
        }
        return disjunctiveGame;
    }

    private List<Coalition> computeConjunctiveCoalitions() {
        List<Coalition> result = new ArrayList<>();
        for (Coalition coalition : coalitionsValue.keySet()) {
            boolean isCon = true;
            for (Player player : coalition.players) {
                List<Player> predecessors = player.getPredecessors();
                if (!containsAll(coalition, predecessors)) {
```

```
                isCon = false;
            }
        }
        if (isCon) {
            result.add(coalition);
        }
    }
    return result;
}

public List<Coalition> computeDisjunctiveCoalitions() {
    List<Coalition> result = new ArrayList<>();
    for (Coalition coalition : coalitionsValue.keySet()) {
        boolean isDis = true;
        for (Player player : coalition.players) {
            List<Player> predecessors = player.getPredecessors();
            if (!containsOne(coalition, predecessors)) {
                isDis = false;
            }
        }
        if (isDis) {
            result.add(coalition);
        }
    }
    return result;
}

public static Map<Player, Float> computeDivision(Game game, Coalition bestCoalition, List<Coalition> set, Map<Player, Float> result) {
    Pair<Coalition, Float> newBestCoalition = findBestCoalition(game, set, bestCoalition);
    List<Player> outPlayers =
```

```
    bestCoalition.computeDiff(newBestCoalition.first);
for (Player player : outPlayers) {
  result.put(player, newBestCoalition.second);
}
if (newBestCoalition.first.players.size() == 1 && newBestCoalition.first.players.get(0).key == 1) {
  return result;
}
Player[] newPlayers = new Player[game.players.length - outPlayers.size()];
int i = 0;
loop:
for (Player player : game.players) {
  for (Player outPlayer : outPlayers) {
    if (player == outPlayer) {
      continue loop;
    }
  }
  newPlayers[i] = player;
  i++;
}
Player outPlayersRoot = outPlayers.get(0);
for (Player outPlayer : outPlayers) {
  boolean find = true;
  for (Player outPlayer1 : outPlayers) {
    if (outPlayer.containsInPredecessors(outPlayer1)) {
      find = false;
    }
  }
  if (find) {
    outPlayersRoot = outPlayer;
    break;
  }
```

```
    }
    List<Player> outRootPredecessors = outPlayersRoot.getPredecessors();
    Set<Player> allChildren = new HashSet<>();
    for (Player outPlayer : outPlayers) {
      List<Player> children = outPlayer.getChildren();
      for (Player child : children) {
        if (!outPlayers.contains(child)) {
          allChildren.addAll(children);
        }
      }
    }
    for (Player player : newPlayers) {
      for (Player outPlayer : outPlayers) {
        player.removePredecessor(outPlayer);
        player.removeChild(outPlayer);
      }
      if (outRootPredecessors.contains(player)) {
        for (Player child : allChildren) {
          child.addPredecessor(player);
        }
      }
    }

    Game newGame = new Game(newPlayers);
    for (Coalition coalition : game.coalitionsValue.keySet()) {
      if (coalition.contains(outPlayers)) continue;

      if (coalition.contains(outRootPredecessors) || coalition.containsIn(outRootPredecessors)) {
        Coalition coalition1 = findCoalition(game, coalition, bestCoalition, newBestCoalition.first);
        Float ul = game.coalitionsValue.get(coalition1);
```

```java
            Float ur = newBestCoalition.second;
            int down = bestCoalition.players.size() -
               newBestCoalition.first.players.size();
            float value = ul - ur * down;
            newGame.coalitionsValue.put(coalition, value);
         } else {
            Float value =
               game.coalitionsValue.get(coalition);
            newGame.coalitionsValue.put(coalition, value);
         }
      }

      return
         computeDivision(newGame.computeDisjunctiveGame(),
         newBestCoalition.first,
         newGame.computeDisjunctiveCoalitions(), result);
   }

   private static Coalition findCoalition(Game game,
      Coalition initCoalition, Coalition bestCoalition,
      Coalition newBestCoalition) {
      List<Player> players =
         bestCoalition.computeDiff(newBestCoalition);
      players.addAll(initCoalition.players);
      for (Coalition coalition :
         game.coalitionsValue.keySet()) {
         if (coalition.strongContains(players)) {
            return coalition;
         }
      }
      return null;
   }

   private static Pair<Coalition, Float>
      findBestCoalition(Game game, List<Coalition> set,
```

```java
    Coalition bestCoalition) {
    Coalition newBestCoalition = null;
    float bestValue = Float.MAX_VALUE;
    for (Coalition coalition : set) {
      if (coalition.players.size() ==
          game.players.length) continue;



      float tau = tau(game, bestCoalition, coalition);
      if (tau < bestValue) {
        newBestCoalition = coalition;
        bestValue = tau;
      } else if (tau == bestValue) {
        if (newBestCoalition == null) {
          newBestCoalition = coalition;
          bestValue = tau;
        } else if (coalition.players.size() >
            bestCoalition.players.size()) {
          newBestCoalition = coalition;
          bestValue = tau;
        }
      }
    }
    return new Pair<>(newBestCoalition, bestValue);
  }

  private static float tau(Game game, Coalition
      bestCoalition, Coalition testCoalition) {
    Float best = game.coalitionsValue.get(bestCoalition);
    Float test = game.coalitionsValue.get(testCoalition);

    return (best - test)
```

```java
            / (bestCoalition.players.size() - testCoalition.players.size() + 1);
    }

    public Game computeDisjunctiveGame() {
        List<Coalition> disjunctiveCoalitions = computeDisjunctiveCoalitions();
        Map<Coalition, Coalition> coalitionMap = new HashMap<>();
        for (Coalition coalition : coalitionsValue.keySet()) {
            coalitionMap.put(coalition, findBestCompareCoalition(coalition, disjunctiveCoalitions));
        }
        Game disjunctiveGame = new Game(Arrays.copyOf(players, players.length));
        for (Coalition coalition : coalitionMap.keySet()) {
            Coalition key = coalitionMap.get(coalition);
            Float value = key.isEmpty() ? 0f : coalitionsValue.get(key);
            disjunctiveGame.putCoalition(coalition, value);
        }
        return disjunctiveGame;
    }

    private Coalition findBestCompareCoalition(Coalition coalition, List<Coalition> set) {
        Coalition best = new Coalition();
        int bestScore = -1;
        for (Coalition element : set) {
            int score = score(coalition, element);
            if (score > bestScore) {
                best = element;
                bestScore = score;
```

```
      }
    }
    return best;
  }

  private static int score(Coalition rootCoalition,
      Coalition mappedCoalition) {
    List<Player> mappedPlayers =
        mappedCoalition.getPlayers();
    List<Player> rootPlayers =
        rootCoalition.getPlayers();
    if (mappedPlayers.size() > rootPlayers.size()) {
      return -1;
    } else {
      if (containsAll(rootCoalition, mappedPlayers)) {
        return mappedPlayers.size();
      }
      return -1;
    }
  }

  private static boolean containsOne(Coalition
      coalition, List<Player> players) {
    if (players.size() == 0) return true;
    for (Player player : players) {
      if (coalition.contains(player)) return true;
    }
    return false;
  }

  private static boolean containsAll(Coalition
      coalition, List<Player> players) {
    if (players.size() == 0) return true;
    for (Player player : players) {
      if (!coalition.contains(player)) return false;
```

```java
        }
        return true;
    }

    @Override
    public String toString() {
        StringBuilder stringBuilder = new StringBuilder();
        List<Coalition> copy = new
            ArrayList<>(coalitionsValue.keySet());
        Collections.sort(copy);

        for (Coalition coalition : copy) {
            stringBuilder.append(coalition).append(": 
                ").append(coalitionsValue.get(coalition)).append("\n")
        }
        return stringBuilder.toString();
    }
}
package entity;

public class Pair<T, E> {
    public T first;
    public E second;

    public Pair(T first, E second) {
        this.first = first;
        this.second = second;
    }

    @Override
    public String toString() {
        return "Pair{" +
                "first=" + first +
                ", second=" + second +
                '}';
```

```
    }
}
package entity;

import java.util.*;

public class Player {
  public final int key;
  private final Set<Player> predecessors = new
     HashSet<>();
  private final Set<Player> children = new HashSet<>();

  public Player(int key) {
    this.key = key;
  }

  public Player copy() {
    Player player = new Player(key);
    for (Player pred : predecessors) {
      player.addPredecessor(pred.copy());
    }
    return player;
  }

  public void addPredecessor(Player predecessor) {
    predecessors.add(predecessor);
    predecessor.children.add(this);
  }

  public void removePredecessor(Player predecessor) {
    predecessors.remove(predecessor);
    predecessor.children.remove(this);
  }

  public void removeChild(Player child) {
```

```java
        children.remove(child);
        child.predecessors.remove(this);
    }

    public boolean isTopPlayer() {
        return predecessors.isEmpty();
    }

    public List<Player> getPredecessors() {
        return new ArrayList<>(predecessors);
    }

    public List<Player> getChildren() {
        return new ArrayList<>(children);
    }

    public boolean containsInPredecessors(Player player) {
        return predecessors.contains(player);
    }

    public static Player[] createPlayers(int n) {
        Player[] players = new Player[n];
        for (int i = 0; i < n; i++) {
            players[i] = new Player(i + 1);
        }
        return players;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Player player = (Player) o;
        return key == player.key;
```

```java
    }

    @Override
    public int hashCode() {
        return Objects.hash(key);
    }

    @Override
    public String toString() {
        return String.valueOf(key);
    }
}
package gameTheory;

import entity.Coalition;
import entity.CoalitionUtils;
import entity.Game;
import entity.Player;
import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

import static gameTheory.Type.CONJUNCTIVE;
```

```java
import static gameTheory.Type.values;

public class Main1 extends Application {
  private FileChooser fileChooser;
  private Scanner scanner;
  private Game initGame;
  private TextField alphaField;
  private TextArea answerArea;
  private ComboBox<Type> gameTypeDropdown;

  public static void main(String[] args) {
    launch(args);
  }

  @Override
  public void start(Stage stage) {
    stage.setTitle("Game solver");

    fileChooser = new FileChooser();
    fileChooser.setTitle("Select data file");

    VBox root = new VBox();
    root.setPadding(new Insets(10));
    root.setSpacing(20);

    HBox contentBox = new HBox();
    contentBox.setSpacing(30);

    contentBox.getChildren().add(new Label("Alpha: "));
    alphaField = new TextField();
    alphaField.textProperty().setValue("0.1");
    contentBox.getChildren().add(alphaField);

    Button startButton = new Button();
    startButton.setText("Compute");
```

```java
    startButton.setOnAction(event -> run());
    contentBox.getChildren().add(startButton);

    answerArea = new TextArea();
    answerArea.setMinHeight(400);
    contentBox.getChildren().add(answerArea);



    Node header = createHeader(stage);
    root.getChildren().add(header);
    root.getChildren().add(contentBox);

    stage.setScene(new Scene(root, 1024, 512));
    stage.show();
  }

  private void createGame(Scanner scanner) {
    Player[] players =
      Player.createPlayers(scanner.nextInt());
    initGame = new Game(players);

    for (int i = 0; i < players.length; i++) {
      int from = scanner.nextInt();
      int to = scanner.nextInt();
      players[to - 1].addPredecessor(players[from - 1]);
    }

    List<Coalition> coalitions =
      CoalitionUtils.createCoalitions(players);
    coalitions.sort(Coalition::compareTo);
    for (Coalition coalition : coalitions) {
      initGame.putCoalition(coalition,
        scanner.nextFloat());
    }
```

```java
  }

  private Node createHeader(Stage stage) {
    HBox header = new HBox();
    header.setSpacing(200);

    Label fileNameLabel = new Label("");

    Button fileChooserButton = new Button("Open file");
    fileChooserButton.setOnAction(actionEvent -> {
      File file = fileChooser.showOpenDialog(stage);
      System.out.println(file);
      if (file != null) {
        try {
          scanner = new Scanner(file);
          createGame(scanner);
          fileNameLabel.textProperty().setValue(file.getName()
        } catch (FileNotFoundException ignored) { }
      }
    });


    ObservableList<Type> types =
       FXCollections.observableArrayList(new
       ArrayList<>(Arrays.asList(values())));
    gameTypeDropdown = new ComboBox<>(types);
    gameTypeDropdown.valueProperty().set(CONJUNCTIVE);


    HBox box = new HBox();
    box.getChildren().add(fileChooserButton);
    box.getChildren().add(fileNameLabel);
    box.setSpacing(10);

    header.getChildren().add(box);
```

```java
    header.getChildren().add(gameTypeDropdown);
    return header;
  }

  private void run() {
    if (initGame == null) return;
    try {
      initGame.setBalancedParam(Float.parseFloat(alphaField.tex
    } catch(NumberFormatException e) {
      e.printStackTrace();
      initGame.setBalancedParam(0.5f);
    }
    Type currentType =
       gameTypeDropdown.valueProperty().get();
    StringBuilder stringBuilder = new StringBuilder();
    switch (currentType) {
      case CONJUNCTIVE:
        stringBuilder.append("Conjunctive: \n");
        stringBuilder.append(initGame.computeConjunctiveGame()
        break;
      case DISJUNCTIVE:
        stringBuilder.append("Disjunctive: \n");
        stringBuilder.append(initGame.computeDisjunctiveGame()
        break;
      case DUAL_GAME:
        stringBuilder.append("Dual: \n");
        stringBuilder.append(initGame.computeDualGame().toStrin
        break;
      case BALANCED_GAME:
        stringBuilder.append("Balanced: \n");
        stringBuilder.append(initGame.computeBalancedGame().to
        break;
      case CONJUNCTIVE_BALANCED:
        stringBuilder.append("Conjunctive balanced: \n");
        stringBuilder.append(initGame.computeBalancedGame().co
```

```
        break;
      case DISJUNCTIVE_BALANCED:
        stringBuilder.append("Disjunctive balanced: \n");
        stringBuilder.append(initGame.computeBalancedGame().co
        break;
      case ALPHA_NUCLEOLUS:
        stringBuilder.append(alphaField.textProperty().get()).
          of (N, r): \n");
        Game balancedGame =
          initGame.computeBalancedGame().copy();
        List<Coalition> coalitions =
          balancedGame.computeDisjunctiveCoalitions();

        boolean isMonotonic = initGame.checkMonotonic();
        boolean isConcave = balancedGame.checkConcave();
        if (!(isConcave && isMonotonic)) {
          stringBuilder.append("Function doesn't satisfy 
            required properties. \n");
          stringBuilder.append("monotonicity: 
            ").append(isMonotonic).append("\n");
          stringBuilder.append("concavity: 
            ").append(isConcave).append("\n");
          break;
        }

        Game balancedDis =
          balancedGame.computeDisjunctiveGame();

        Map<Player, Float> playerFloatMap = new
          HashMap<>();
        Game.computeDivision(balancedDis,
          balancedDis.coalitionWithAllPlayers,
          coalitions, playerFloatMap);
        float sum = 0;
        for (Player player : playerFloatMap.keySet()) {
```

```
                sum += playerFloatMap.get(player);
            }
            playerFloatMap.put(
                    balancedDis.players[0],
                    balancedDis.coalitionsValue.get(balancedDis.co
                      - sum
            );
            stringBuilder.append(playerFloatMap.values().toString(
        }
        answerArea.setText(stringBuilder.toString());
    }
}
package gameTheory;

public enum Type {
    CONJUNCTIVE,
    DISJUNCTIVE,
    DUAL_GAME,
    BALANCED_GAME,
    CONJUNCTIVE_BALANCED,
    DISJUNCTIVE_BALANCED,
    ALPHA_NUCLEOLUS
}
```