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

Пругло Лев Сергеевич

Магистерская диссертация

Теоретико-игровая модель ромбовидной иерархической структуры

Направление 01.04.02
Прикладная математика и информатика
Магистерская программа
Исследование операций и системный анализ

Научный руководитель,
доктор физ.-мат. наук,
профессор кафедры МТИСР
Петросян Л.А.

Санкт-Петербург
2021

Содержание

Введение

Задача относится к проблеме распределения ресурсов в иерархической структуре. В работе [10] представлена модель классической иерархической игры, в которой задан один управляющий центр и некоторое число подчиненных подразделений с разными связями между игроками. В данной работе мы рассмотрим расширение этой математической модели.

В иерархической многошаговой игре с полной информацией геометрическая структура самой игры является ромбом, то есть из одного центра ресурсы поступают в два подчиненных подразделения, а они, в свою очередь, посылают произведенную промежуточную продукцию последнему подразделению. Последнее подразделение производит продукцию, от которой зависит величина полезности каждого игрока. Теоретико-игровые модели, основанные на такой структуре отношений игроков, называются ромбовидными. Рассматривается модель ромбовидной структуры иерархической игры с расширением, в которой присутствует прямая связь между распределяющим центром и нижним производящим подразделением.

Интерпретацией данной задачи может служить экономическая, экологическая или межрегиональная характеристика, когда какой-то ресурс, в рамках межрегиональной проблематики, например воспринимаемый как человеческий капитал, распределяется федеральным центром, а те, в свою очередь, отправляют его в муниципальные районы. Либо экономическая иерархия с предприятиями, а именно в случае, когда в конгломерате от управляющего центра распределяются финансы. Другой экономической интерпретацией данных процессов может служить несовершенная конкуренция, когда в ходе конфликтогенеза среди разных экономических агентов выстраивается иерархическая структура монопольного подчинения. В рамках этой работы будем считать такую структуру уже сложившейся. Для игры с характеристической функцией будет определена проблема посредничества и производственной кооперации в коалиционном разбиении подмножества игроков всех уровней.

Целью данной работы является нахождение ситуации равновесия по Нэшу в общем и численном виде игры Г, и получение значений определенных характеристических функций кооперативной игры с распределением трансферабельной полезности между игроками согласно принципу оптимальности – вектору Шепли, для чего необходима разработка алгоритма и программной реализации по математической модели, заданной на ромбовидной структуре.

Рассмотрен процесс нахождения ситуации равновесия по Нэшу в определенной игре Г. Для этого доказана лемма о существовании ситуации равновесия по Нэшу и игре Г. Конкретизирована теоретико-игровая модель через определение множеств стратегий игроков, в частности, составление систем неравенств. Показано, что добавление связи поставок ресурсов между центром и игроком нижнего уровня приводит к появлению второй системы неравенств игрока нижнего уровня, которая учитывает формализацию производства с использованием ресурсов управляющего центра отдельно от производства с помощью ресурсов игроков среднего уровня. Были построены оптимизационные задачи линейного и нелинейного программирования с параметрами и показана возможность нахождения ситуации равновесия по Нэшу. Чтобы разрешить проблему нахождения равновесия в игре Г была введена кооперативная подыгра между игроками среднего уровня. Разработано и учтено три варианта одноэлементных характеристических функций игроков среднего уровня для вычисления тремя различными способами минимальной гарантированной полезности, необходимой для вычисления вектора Шепли - принятого принципа оптимальности. Представлены численные примеры, показывающие специфику значений вектора Шепли при использовании трех различных подходов к определению минималь-

ной гарантированной полезности. Сформулирована в общем виде кооперативная игра на ромбовидной структуре, составлены равенства, которые определяют действия игроков в рамках антагонистической игры двух коалиций. Для каждой из коалиции в кооперативной игре выведены формулы для вычисления вектора Шепли. Для программной реализации составлен алгоритм с модифицированным методом Монте-Карло, который позволил конкретнее описать методику случайного поиска для нахождения ситуации равновесия по Нэшу, значений характеристических функций в кооперативной игре и вектора Шепли через полное покрытие области допустимых решений систем линейных неравенств игроков среднего уровня. Определена структура алгоритма, проведен алгоритмический анализ и выявлены особенности применения модифицированного метода Монте-Карло к решению задачи. По алгоритму построена программная реализация, которая позволила численно решить данную задачу. Приведен пример выполнения программы по заданному алгоритму, который показывает специфику в использовании трех подходов к определению одноэлементных характеристических функций.

Обзор литературы

Первая группа литературы относится к классическим работам по теории игр в области неантагонистических многошаговых игр, проблематика которых затрагивается в [10], [8], [4]. Для связи оптимизационных проблем с теоретико-игровыми решениями был использован источник [3] и [11]. Также к первой группе относятся работы в области популярных решений для теоретико-игровых задач, например [9] и [1]. И, также, работы в области алгоритмической теории игр, учитывающие особенности различных моделей [5]. Для понимания теории систем линейных неравенств был использован источник [7].

Вторая группа научной литературы посвящена работам в области иерархических игр, которые освещали аспекты классических решений теории игр. К таким работам относятся научные статьи, например [2], [6].

Глава I Ромбовидная иерархическая структура

1.1. Описание теоретико–игровой модели

В данной главе в первом параграфе будут рассмотрен вопрос постановки неантагонистической многошаговой иерархической ромбовидной игры Г в нормальной форме. Во втором параграфе будут определены оптимизационные задачи каждого элемента множества игроков, а в третьем параграфе определены характеристические функции для кооперативного коалиционного разбиения. Введем дефиницию многошаговой игры с полной информацией.

Подклассом неантагонистических многошаговых игр с полной информацией являются иерархические игры. Рассмотрим трёхуровневую ромбовидную структуру иерархической игры с непосредственным ребром между игроком A_0 и игроком C. Отношения подчинения между управляющим центром и производственными подразделениями можно выразить на следующем рисунке:

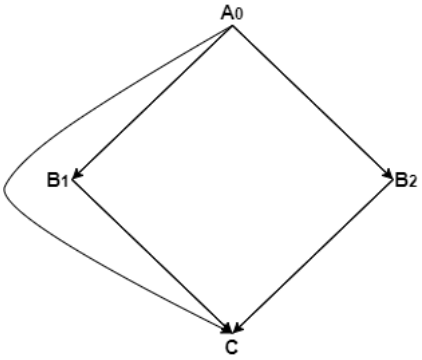

Рис.1

Рассмотрим множество стратегий игрока A_0, который является управляющим центром в данной структуре иерархии и совершает ход первым.

$$u \in U = \{u = (u_1, u_2, u_3) : u_i \geq 0, u_i \in R^l, \sum_{i=1}^{3} u_i \leq b, b \geq 0, i = 1, 2, 3\} \tag{1}$$

Множество стратегий игрока A_0 (1) состоит из наборов 3 неотрицательных векторов $u_i = 1, 2, 3$, которые имеют размерность l. Каждый вектор из набора 3 неотрицательных векторов поступает подчинённым производственным подразделениям как вектор ресурса, который необходим для производства продукции. Каждый вектор этого набора $u_i = 1, 2, 3$ принадлежит полю вещественных чисел с указанной размерностью определяющих число типов ресурсов. Сумма компонент по номерам $i = \{1, 2, 3\}$ векторов набора не должна быть больше соответствующего номера компоненты вектора b. Таким образом, b - вектор l типов ресурсов игрока A_0, который распределяется по трём векторам l типов ресурсов u_1, u_2, u_3, отправляемых игрокам B_1, B_2, C соответственно.

Рассмотрим далее множества стратегий игроков B_1 и B_2, которые одновременно совершают ходы на втором уровне.

$$\begin{gathered} \omega_1 \in \Omega_1(u_1) = \{\omega_1(u_1) = (\omega_1^1(.), ..., \omega_1^m(.)) : \omega_1^k(.) \geq 0, \\ \omega_1(.) \in R^m, \omega_1(u_1)A_1 \leq u_1 + \xi_1, k = 1, ..., m\} \end{gathered} \tag{2}$$

Множество стратегий игрока B_1 (2) состоит из m неотрицательных векторов промежуточной продукции $\omega_1 = (\omega_1^1(.), ..., \omega_1^m(.))$. Каждый неотрицательный вектор ω_1 множества стратегий Ω_1 принадлежит полю вещественных чисел с размерностью m, определя-

ющей число типов промежуточной продукции, которая поступает как ресурс игроку C. Компонента неотрицательного вектора ω_1 является скаляром, определяющим величину k, $k = 1, ..., m$ типа промежуточной продукции. Матрица A_1 является технологической матрицей игрока B_1 состоящей из элементов $A_1 = \left\{\alpha_{kj}^1\right\}$ данной технологической программы, определяющей величину ресурса типа j для производства 1 единицы промежуточной продукции типа $k = 1, \ldots, m$. Также определяется вектор наличных ресурсов ξ_1 игрока B_1, находящихся в его распоряжении вне зависимости от решения A_0. Сумма векторов u_1 и ξ_1 определяет весь объем ресурсов, который распределяет игрок B_1 по k типам промежуточной продукции.

$$\begin{cases} \omega_1^1\alpha_{11}^1 + \cdots + \omega_k^1\alpha_{k1}^1 + \cdots + \omega_m^1\alpha_{m1}^1 \le u_1^1 + \xi_1^1, \\ ..., \\ \omega_1^1\alpha_{1j}^1 + \cdots + \omega_k^1\alpha_{kj}^1 + \cdots + \omega_m^1\alpha_{mj}^1 \le u_j^1 + \xi_j^1, \\ ..., \\ \omega_1^1\alpha_{1l}^1 + \cdots + \omega_k^1\alpha_{kl}^1 + \cdots + \omega_m^1\alpha_{ml}^1 \le u_l^1 + \xi_l^1. \end{cases} \tag{3}$$

Расписанное в координатах выражение (3) неравенства $\omega_1(u_1)A_1 \le u_1 + \xi_1$ из определения множества стратегий игрока B_1 в системе линейных алгебраических неравенств показывает, что $\omega_k^1\alpha_{kj}^1$ – величина, показывающая сколько ресурсов типа $j = 1, \ldots, l$ необходимо для производства продукции типа $k = 1, \ldots, m$, исходя из имеющегося объема всех ресурсов $u_j^1 + \xi_j^1$. Полная возможная величина расхода всего объема ресурса $u_j^1 + \xi_j^1$ типа $j = 1, \ldots, l$ показана как сумма произведений в левой части линейного неравенства.

Множество стратегий игрока B_2 определяется аналогично.

$$\begin{gathered} \omega_2 \in \Omega_2(u_2) = \{\omega_2(u_2) = (\omega_2^1(.), ..., \omega_2^m(.)) : \omega_2^k(.) \ge 0, \\ \omega_2(.) \in R^m, \omega_2(u_2)A_2 \le u_2 + \xi_2, k = 1, ..., m\} \end{gathered} \tag{4}$$

Множество стратегий игрока B_2 (4) состоит из m неотрицательных векторов промежуточной продукции $\omega_2 = (\omega_2^1(.), ..., \omega_2^m(.))$. Каждый неотрицательный вектор ω_2 множества стратегий Ω_2 принадлежит полю вещественных чисел с размерностью m, определяющей число типов промежуточной продукции, которая поступает как ресурс игроку C. Компонента неотрицательного вектора ω_2 является скаляром, определяющим величину k, $k = 1, ..., m$ типа промежуточной продукции. Матрица A_2 является технологической матрицей игрока B_2 состоящей из элементов $A_2 = \left\{\alpha_{kj}^2\right\}$ данной технологической программы, определяющей величину ресурса типа $j = 1, \ldots, l$ для производства 1 единицы промежуточной продукции типа $k = 1, \ldots, m$. Также определяется вектор наличных ресурсов ξ_2 игрока B_2, находящихся в его распоряжении вне зависимости от решения A_0. Сумма векторов u_2 и ξ_2 определяет весь объем ресурсов, который распределяет игрок B_2 по k типам промежуточной продукции.

$$\begin{cases} \omega_1^2\alpha_{11}^2 + \cdots + \omega_k^2\alpha_{k1}^2 + \cdots + \omega_m^2\alpha_{m1}^2 \le u_1^2 + \xi_1^2, \\ ..., \\ \omega_1^2\alpha_{1j}^2 + \cdots + \omega_k^2\alpha_{kj}^2 + \cdots + \omega_m^2\alpha_{mj}^2 \le u_j^2 + \xi_j^2, \\ ..., \\ \omega_1^2\alpha_{1l}^2 + \cdots + \omega_k^2\alpha_{kl}^2 + \cdots + \omega_m^2\alpha_{ml}^2 \le u_l^2 + \xi_l^2. \end{cases} \tag{5}$$

Расписанное в координатах выражение (5) неравенства $\omega_2(u_2)A_2 \le u_2 + \xi_2$ из определения множества стратегий игрока B_2 в системе линейных алгебраических неравенств показывает, что $\omega_k^2\alpha_{kj}^2$ – величина, показывающая сколько ресурсов типа $j = 1, \ldots, l$ необходимо для производства продукции типа $k = 1, \ldots, m$, исходя из имеющегося объема всех

ресурсов $u_j^2 + \xi_j^2$. Полная возможная величина расхода всего объема ресурса $u_j^2 + \xi_j^2$ типа $j = 1, \ldots, l$ показана как сумма произведений в левой части линейного неравенства.

В рамках экономический интерпретации данной задачи будем считать, что векторы промежуточных ресурсов ω_1 и ω_2, доставляющие для производства продукции игроком C, являются однотипными.

Игрок C совершает ход последним. Множество стратегий игрока C учитывает поступление, согласно рис. 1, кроме векторов ω^1 и ω^2 от производства игроков B_1 и B_2 еще и вектор u_3 от игрока A_0.

$$\begin{gathered} v \in V(u_3, \omega_1(u_1), \omega_2(u_2)) = \\ \{v(u_3, \omega_1(u_1), \omega_2(u_2)) = (v_1(u_3, \omega_1(u_1), \omega_2(u_2)), ..., v_q(u_3, \omega_1(u_1), \omega_2(u_2)) : \\ v(u_3, \omega_1(u_1), \omega_2(u_2)) \geq 0, \\ v(u_3, \omega_1(u_1), \omega_2(u_2)) \in R^q, \\ v(u_3, \omega_1(u_1), \omega_2(u_2))D \leq \omega_1(u_1) + \omega_2(u_2) + \gamma_1, \\ v(u_3, \omega_1(u_1), \omega_2(u_2))M \leq u_3 + \gamma_2, \\ p = 1, ..., q\} \end{gathered} \tag{6}$$

Множество стратегий игрока C (6) зависит от векторов ω_1, ω_2 и u_3 и состоит из множества неотрицательных векторов продукции $v_{ip}(u_3, \omega_1(u_1), \omega_2(u_2))$ $p = 1, ..., q$ типа продукции. Следовательно, несмотря на то, что результирующий вектор продукции v имеет один и тот же тип для трех векторов ресурсов ω_1, ω_2 и u_3, поступающих игроку C, производство типов продукции по ресурсам игрока B_1 и B_2 будет отличаться от производства по вектору ресурса u_3, что можно формализовать через две разные системы линейных неравенств, к которым даны технологическая матрицы D и вектор наличных ресурсов γ_1 для обработки поставки ресурсов ω_1 и ω_2 в первой системе линейных неравенств, и технологическая матрица M и вектор наличных ресурсов γ_2 для обработки поставки ресурсов u_3 во второй системе линейных неравенств.

Следовательно, каждый вектор $v_p(u_3, \omega_1(u_1), \omega_2(u_2))$ $p = 1, ..., q$ типа продукции принадлежит полю вещественных чисел размерности q. Компонента неотрицательного вектора $v_{ip}(u_3, \omega_1(u_1), \omega_2(u_2))$ $p = 1, ..., q$ типа продукции является скаляром, показывающим величину данного типа продукции.

Рассмотрим первую систему линейных неравенств. Матрица D является технологической матрицей игрока C, состоящей из элементов $D = \{d_{pk}\}$ данной технологической программы, определяющей величину ресурса типа $j = 1, \ldots, l$ для производства 1 единицы промежуточной продукции типа $p = 1, ..., q$. В распоряжении игрока C находится также вектор наличных ресурсов $\gamma_1 = \{\gamma_{11}, ..., \gamma_{1l}\}$ вне зависимости от решения игроков B_1, B_2 и C.

Поскольку векторы промежуточной продукции ω_1 и ω_2 имеют размерности q, а вектор ресурсов u_3 имеет размерность l, а также потому, что обработка векторов ресурсов ω_1 и ω_2 формализована в другой системе линейных неравенств, не в той же, в какой и u_3, то возможны варианты, что типов промежуточной продукции больше, чем типов ресурсов $q > l$, обратный вариант $l > q$ и вариант их равенства $q = l$. В первом случае вектор u_3 размерности l по компонентам дополняется нулями до размерности q векторов ω_1 и ω_2, а во втором векторы ω_1 и ω_2 размерности q дополняются нулями до вектора u_3, оба случая обеспечивают равенство размерностей $q = l$.

$$\begin{cases} v_1 d_{11} + \cdots + v_p d_{p1} + \cdots + v_q d_{q1} \le \omega_1^2(u_2) + \omega_1^1(u_1) + \gamma_{11}, \\ ..., \\ v_1 d_{1j} + \cdots + v_p d_{pj} + \cdots + v_q d_{qj} \le \omega_j^2(u_2) + \omega_j^1(u_1) + \gamma_{1j}, \\ ..., \\ v_1 d_{1l} + \cdots + v_p d_{pl} + \cdots + v_q d_{ql} \le \omega_l^2(u_2) + \omega_l^1(u_1) + \gamma_{1l}. \end{cases} \quad (7)$$

Расписанное в координатах выражение (7) неравенства $v(u_3, \omega_1(u_1), \omega_2(u_2))D \le \omega_1(u_1) + \omega_2(u_2) + \gamma_1$ из определения множества стратегий игрока C в системе линейных алгебраических неравенств показывает, что $v_p d_{pj}$ – величина, показывающая сколько ресурсов типа $j = 1, ..., l$ необходимо для производства величины продукции типа $p = 1, ..., q$. Полная возможная величина расхода всего объема ресурса $\omega_j^2(u_2) + \omega_j^1(u_1) + \gamma_{1j}$ показана как сумма произведений в левой части неравенства.

Рассмотрим вторую систему линейных неравенств. Матрица M является технологической матрицей игрока C, состоящей из элементов $M = \{m_{pk}\}$ данной технологической программы, определяющей величину ресурса типа $j = 1, \ldots, l$ для производства 1 единицы промежуточной продукции типа $p = 1, ..., q$. В распоряжении игрока C находится также вектор наличных ресурсов $\gamma_2 = \{\gamma_{21}, ..., \gamma_{2l}\}$ вне зависимости от решения игроков B_1, B_2 и C.

$$\begin{cases} v_1 d_{11} + \cdots + v_p d_{p1} + \cdots + v_q d_{q1} \le u_1^3 + \gamma_{21}, \\ ..., \\ v_1 d_{1j} + \cdots + v_p d_{pj} + \cdots + v_q d_{qj} \le u_j^3 + \gamma_{2j}, \\ ..., \\ v_1 d_{1l} + \cdots + v_p d_{pl} + \cdots + v_q d_{ql} \le u_l^3 + \gamma_{2l}. \end{cases} \quad (8)$$

Расписанное в координатах выражение (8) неравенства $v(u_3, \omega_1(u_1), \omega_2(u_2))D \le u_3 + \gamma_2$ из определения множества стратегий игрока C в системе линейных алгебраических неравенств показывает, что $v_p d_{pj}$ – величина, показывающая сколько ресурсов типа $j = 1, ..., l$ необходимо для производства величины продукции типа $p = 1, ..., q$. Полная возможная величина расхода всего объема ресурса $u_j^3 + \gamma_{2j}$ показана как сумма произведений в левой части неравенства.

Таким образом, вектор $v(u_3, \omega_1(u_1), \omega_2(u_2))$выступает в качестве вектора продукции, являющегося функцией, $v(.)$, который зависит от аргументов – векторов промежуточной продукции ω_1, ω_2, поступающего в правую часть линейного неравенства (8) по компонентам в качестве свободных членов систем линейных алгебраических неравенств непосредственно игроку C и вектора ресурсов u_3 игрока A_0, поступающего в правую часть линейного неравенства (8) по компонентам в качестве свободных членов систем линейных алгебраических неравенств непосредственно игроку C; векторы ω_1, ω_2 в свою очередь выступают как функции, зависящие от векторов u_1, u_2 игрока A_0, поступающих в правые части неравенств в качестве свободных членов систем алгебраических неравенств игрокам B_1, B_2. Вектор $v(u_3, \omega_1(u_1), \omega_2(u_2))$ удовлетворяет решению системы линейных неравенств (7) и (8).

Пусть полезности всех игроков A_0, B_1, B_2, C зависят только от производственной программы $v(u_3, \omega_1(u_1), \omega_2(u_2))$ выбираемой игроком C и вектора дохода каждого игрока A_0, B_1, B_2, C, и полагается равным вектору c_f, $f = 1, 2, 3, 4$, который скалярно умножается на вектор продукции $v(.)$, вследствие чего полезности равны соответственно $l_1(v, c_1)$, $l_2(v, c_2)$, $l_3(v, c_3)$, $l_4(v, c_4)$, где $l_f(v, c_f) \ge 0$ для $f = 1, 2, 3, 4$.

В соответствии с данными допущениями определим функции полезности игроков A_0,

B_1, B_2, C

$$K_f(u, \omega_1(u_1), \omega_2(u_2), v(u_3, \omega_1(u_1), \omega_2(u_2))) =$$
$$l_f(v(u_3, \omega_1(u_1), \omega_2(u_2)), c_f) = c_f v(u_3, \omega_1(u_1), \omega_2(u_2)), \quad (9)$$
$$f = 1, 2, 3, 4.$$

Вид функции (9) полезности указывает на терминальное свойство, то есть на то, что вектор продукции произведенный на последнем ходу игроком C вследствие скалярного произведения на каждый вектор дохода c_f, $f = 1, 2, 3, 4$, каждого игрока A_0, B_1, B_2, C производится вычисление полезности этого игрока. Произведение компонент вектора продукции $v(u_3, \omega_1(u_1), \omega_2(u_2))$ и компонент вектора дохода c_f, $f = 1, 2, 3, 4$ показывает доход игрока f от $p = 1, ..., q$ типа в общей продукции.

Сформулированную иерархическую игру можно представить как бескоалиционную игру четырёх лиц в нормальной форме. Полагая, что (9) выполняется, тогда получим нормальную форму игры Γ

$$\Gamma = (U, \Omega_1, \Omega_2, V, K_1, K_2, K_3, K_4). \quad (10)$$

Начнём нахождение ситуации равновесия по Нэшу в игре Γ с вспомогательных построений в виде параметрических задач линейного программирования. Для каждой фиксированной тройки $(\omega_1, \omega_2, u_3) \in \bigcup_{u_1,u_2,u_3 \in U} \bigcup_{\omega_1 \in \Omega_1} \bigcup_{\omega_2 \in \Omega_2} \Omega_1(u_1) \times \Omega_2(u_2) \times U$ обозначим решение параметрической экстремальной задачи

$$\max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} l_4(v, c_4) = l_4(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_4) \quad (11)$$

Считаем, что максимум параметрической задачи (11) достигается. Тогда видно, что решение $v^*(u_3, \omega_1(u_1), \omega_2(u_2))$ является функцией параметров u_3, ω_1, ω_2 и $v^*(u_3, \omega_1(u_1), \omega_2(u_2)) \in V$.

Теперь рассмотрим вторую вспомогательную неантагонистическую игру $\Gamma'(u_1, u_2)$

$$\Gamma'(u_1, u_2) = \{\Omega_1(u_1), \Omega_2(u_2), l_2, l_3\}, \quad (12)$$

где $l_2 = l_2(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_2)$, $l_3 = l_3(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_3)$. Стратегиями игрока B_1 в игре $\Gamma'(u_1, u_2)$ являются элементы $\omega_1 \in \Omega_1(u_1)$, а стратегиями игрока B_2 в игре $\Gamma'(u_1, u_2)$ являются элементы $\omega_2 \in \Omega_2(u_2)$. Предположим, что в игре $\Gamma'(u_1, u_2)$ существует ситуация равновесия по Нэшу, которую обозначим $(\omega_1^*(u_1), \omega_2^*(u_2))$. Отметим, что $\omega_1^*(u_1)$ является функцией параметра u_1 при $\omega_1^*(u_1) \in \Omega_1(u_1)$ для игрока B_1 и, аналогично, $\omega_2^*(u_2)$ есть функция параметра u_2 при $\omega_2^*(u_2) \in \Omega_2(u_2)$ для игрока B_2.

Поскольку мы предполагаем, что ситуация равновесия по Нэшу в игре $\Gamma'(u_1, u_2)$ (12) существует, то, далее, пусть $u^* = (u_1^*, u_2^*, u_3^*)$ – решение следующей оптимизационной задачи:

$$\max_{u \in U} l_1(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_4) = l_1(v^*(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)), c_4) \quad (13)$$

Докажем существование ситуации равновесия в игре Γ.

Лемма. *Совокупность* $(u^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*), v^*(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)))$ *является ситуацией равновесия по Нэшу в игре* Γ.

Доказательство. Согласно определению u^* из (13) следует соотношение

$$K_1(u^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*), v(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*))) = \max_{u \in U} l_1(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_4) =$$
$$= l_1(v^*(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)), c_4) \geq l_1(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_4) = \quad (14)$$
$$= K_1(u, \omega_1^*(u_1), \omega_2^*(u_2), v(u_3, \omega_1^*(u_1), \omega_2^*(u_2))).$$

Для всех $u \in U$. Поскольку $(\omega_1^*(u_1^*), \omega_2^*(u_2^*))$ образует ситуацию равновесия по Нэшу во вспомогательной игре $\Gamma'(u_1^*, u_2^*)$, для любой функции $\omega_1(u_1) \in \Omega_1$, $\omega_1(u_1^*) = \widetilde{\omega_1} \in \Omega_1(u_1^*)$ выполняются соотношения

$$\begin{gathered} K_2(u, \omega_1^*(u_1), \omega_2^*(u_2), v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2))) = \\ = l_2(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_4) \geq l_2(v^*(u_3^*, \widetilde{\omega}_1(u_1^*), \omega_2^*(u_2^*)), c_4) = \\ = K_2(u^*, \omega_1(u_1^*), \omega_2^*(u_2^*), v^*(u_3^*, \omega_1(u_1^*), \omega_2^*(u_2^*))). \end{gathered} \tag{15}$$

Аналогичное неравенство справедливо и для игрока B_2. По определению функции v^* из (11) имеем:

$$\begin{gathered} K_4(u^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*), v(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*))) = \\ = \max_{v \in V(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*))} l_4(v, c_4) = l_4(v^*(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)), c_4) \geq l_4(\widetilde{v}) = \\ = K_4(u^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*), v(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*))). \end{gathered} \tag{16}$$

Для любой функции $v(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*))) \in V$,

$$v(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)) = \widetilde{v} \in V(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*). \tag{17}$$

Лемма доказана.

1.2. Постановка задач оптимизации

Для нахождения ситуации равновесия по Нэшу численно необходимо построить ряд вспомогательных задач линейного и нелинейного программирования с параметрами, ограничения которых содержатся в выражениях множеств стратегий (1), (2), (4), (6) для игроков A_0, B_1, B_2, C соответственно. Покажем оптимизационные задачи для конкретных видов функций $v(u_3, \omega_1(u_1), \omega_2(u_2), \omega_1(u_1), \omega_2(u_2), u_1, u_2, u_3$. При этом функция $v(u_3, \omega_1(u_1), \omega_2(u_2)$ является существенно разрывной нелинейной функций, максимизация которой для решения задачи оптимизации не реализуется известными методами.

$$\max_{u \in U} c_1 v(u_3, \omega_1(u_1), \omega_2(u_2)) \tag{18}$$

$$\begin{cases} \sum_{i=1}^{3} u_i \leq b, \\ u_i \geq 0, i = 1, 2, 3; \end{cases} \tag{19}$$

Система линейных алгебраических неравенств (19) указывает на то, что вектор ресурсов b, $b \in R^l$ распределяется по неотрицательным векторам u_1, u_2, u_3, сумма номеров компонент которых не превышает соответствующей компоненты b. В целевой функции (18), происходит максимизация по u скалярного произведения вектора дохода c_1 и вектора продукции, выступающего функцией, $v(u_3, \omega_1(u_1), \omega_2(u_2))$, который зависит от аргументов – векторов u_3, ω_1, ω_2. Функция $v(u_3, \omega_1(u_1), \omega_2(u_2))$ выступает в задаче нелинейного программирования существенно разрывной функцией параметров u_1, u_2, u_3.

Нахождение ситуации равновесия по Нэшу в игре Γ в заданном виде осложнено тем, что игроки B_1 и B_2 одновременно выбирают стратегии, которые являются аргументами $v(u_3, \omega_1(u_1), \omega_2(u_2))$ и нахождение равновесной ситуации может не быть получено. Чтобы преодолеть данную трудность в одновременной игре Γ мы введем вспомогательную кооперативную подыгру для игроков B_1 и B_2, считая его одним игроком S, который заинтересован в максимизации суммарной полезности, при этом полезность игроков B_1 и B_2 разделяется по принципу оптимальности – вектору Шепли. Тогда игра Γ превратится в игру 3-х лиц, для которых строится ситуация равновесия по Нэшу, модификацию которой

мы рассмотрим далее. Для рассмотрения следующих оптимизационных задач игроков B_1 и B_2 необходимо внести поправку. Формально введем кооперативную подыгру $\overline{\Gamma}$: есть пара (S, v), где $S = \{B_1, B_2\}$, а v – характеристическая функция: $2^S \to \mathbb{R}$, $v(\emptyset) = 0$. В таком случае изменится задача нелинейного программирования параметров ω_1 и ω_2 для искомых игроков B_1 и B_2.

$$\max_{\substack{\omega_1(u_1)\in\Omega_1(u_1)\\ \omega_2(u_2)\in\Omega_2(u_2)}} c_2 v(u_3, \omega_1(u_1), \omega_2(u_2)) + c_3 v(u_3, \omega_1(u_1), \omega_2(u_2)). \tag{20}$$

При ограничениях:

$$\begin{cases} \omega_1(u_1)A_1 \le u_1 + \xi_1, \\ \omega_2(u_2)A_2 \le u_2 + \xi_2, \\ \omega_1 \ge 0; \xi_1 \ge 0, \\ \omega_2 \ge 0; \xi_2 \ge 0. \end{cases} \tag{21}$$

Система линейных алгебраических неравенств (21) показывает, что в качестве свободных членов выступает сумма $u_1 + \xi_1$ для системы неравенств $\omega_1(u_1)A_1 \le u_1 + \xi_1$ и $u_2 + \xi_2$ для системы неравенств $\omega_2(u_2)A_2 \le u_2 + \xi_2$ неотрицательных векторов ресурсов игрока A_0 и вектора наличных ресурсов ξ_1 и ξ_2.

Целевая функция (20) теперь представляет собой сумму скалярных произведений вектора дохода игрока B_1 т. е. c_2 и существенно разрывной функции $v(u_3, \omega_1(u_1), \omega_2(u_2))$ и вектора дохода игрока B_2 т. е. c_3 и существенно разрывной нелинейной функции $v(u_3, \omega_1(u_1), \omega_2(u_2))$ Вследствие этого задача относится к задаче нелинейного программирования с параметрами ω_1, ω_2.

Таким образом, игроки B_1 и B_2 в коалиции S по сформулированной задаче выбирают чистую коалиционную стратегию $\omega_S(u_1, u_2) \in R^m$ из множества общих для B_1 и B_2 коалиционных стратегий $\Omega_S(u_1, u_2)$. Множество коалиционных стратегий с ведённой кооперативной подыгрой, как и в некооперативной формулировке, зависит от стратегий u_1 и u_2.

$$\omega_S(u_1, u_2) \in \Omega_S(u_1, u_2) = \bigcup_{u\in U} \Omega_1(u_1) \times \Omega_2(u_2). \tag{22}$$

Этим можно показать, что игроки B_1 и B_2 действуют как один игрок в новой неантагонистической игре Γ трех лиц в нормальной форме. В упрощенном смысле, вектор коалиции S можно воспринимать как $\omega_S(u_1, u_2) = \omega_1(u_1) + \omega_2(u_2)$ в качестве свободных членов неравенства $v(u_3, \omega_1(u_1), \omega_2(u_2))D \le \omega_1(u_1) + \omega_2(u_2) + \gamma_1$, то есть как $v(u_3, \omega_1(u_1), \omega_2(u_2))D \le \omega_S(u_1, u_2) + \gamma_1$.

Функции полезности игроков B_1 и B_2 в коалиционной формулировке изменятся.

$$\begin{gathered} K_2(u, \omega_1(u_1), \omega_2(u_2), v(u_3, \omega_1(u_1), \omega_2(u_2))) + \\ + K_3(u, \omega_1(u_1), \omega_2(u_2), v(u_3, \omega_1(u_1), \omega_2(u_2))) = \\ = l_2(v(u_3, \omega_1(u_1), \omega_2(u_2)), c_2) + l_3(v(u_3, \omega_1(u_1), \omega_2(u_2)), c_3) = \\ = c_2 v(u_3, \omega_1(u_1), \omega_2(u_2)) + c_3 v(u_3, \omega_1(u_1), \omega_2(u_2)) = \\ = K_S(u, \omega_1(u_1), \omega_2(u_2), v(u_3, \omega_1(u_1), \omega_2(u_2))) = l_S(v(u_3, \omega_1(u_1), \omega_2(u_2)), c_2, c_3) \end{gathered} \tag{23}$$

Вследствие этого неантагонистическая многошаговая игра 3-х лиц в нормальной форме $\widetilde{\Gamma}$, при условии, что игроки B_1 и B_2 действуют в коалиции в рамках кооперативной подыгры $\overline{\Gamma}$, будет выглядеть:

$$\widetilde{\Gamma} = (U, \Omega_S, V, K_1, K_S, K_4). \tag{24}$$

Теперь представим оптимизационную задачу для игрока C.

$$\max_{v(.)\in V(.)} c_4 v(u_3, \omega_1(u_1), \omega_2(u_2)) \tag{25}$$

$$\begin{cases} vD \le \omega_1(u_1) + \omega_2(u_2) + \gamma_1, \\ vM \le u_3 + \gamma_2 \\ u_3 \ge 0; \omega_1(u_1) \ge 0; \omega_2(u_2) \ge 0; \gamma_1 \ge 0; \gamma_2 \ge 0. \end{cases} \tag{26}$$

В двух системах линейных алгебраических неравенств (26) видно, что в качестве свободных членов выступает сумма неотрицательных векторов $\omega_1(u_1) + \omega_2(u_2) + \gamma_1$ и $u_3 + \gamma_2$, которые составляют весь возможный объем ресурсов. В целевой функции (25), где $v(.) = v(u_3, \omega_1(u_1), \omega_2(u_2))$, $V(.) = V(u_3, \omega_1(u_1), \omega_2(u_2))$, происходит максимизация скалярного произведения вектора дохода c_4 и вектора продукции, выступающего существенно разрывной функцией, $v(u_3, \omega_1(u_1), \omega_2(u_2))$, которая зависит от аргументов – векторов u_3, ω_1, ω_2. Отличием данной задачи от остальных является то, что она является задачей линейного программирования с параметрами u_3, ω_1, ω_2 и обладает двумя различными системами линейных неравенств.

Для численного нахождения ситуации равновесия по Нэшу в игре Г необходимо адаптировать выкладки с экстремальными задачами (11), (12), (13) к определенным в этом параграфе оптимизационным задачам.

Поскольку для принятия решения о производстве продукции и формировании вектора $v(u_3, \omega_1(u_1), \omega_2(u_2))$ игрок C должен исходить из некоторых значений параметров, то рассмотрим сначала решение задачи (25)-(26). То есть, задача будет решена обратным проходом по структуре распределения ресурсов. Подбор будет происходить по всем возможным значениям параметров u_3, $\omega_1(u_1)$, $\omega_2(u_2)$.

Рассмотрим целевую функция игрока C (25) и предположим, что её максимум достигается. В таком случае обозначим найденное значение символом $*$. Как и ранее, предполагаем, что $v(.) = v(u_3, \omega_1(u_1), \omega_2(u_2))$, $V(.) = V(u_3, \omega_1(u_1), \omega_2(u_2))$. Тогда,

$$\begin{gathered} \max_{v(.)\in V(.)} c_4 v(u_3, \omega_1(u_1), \omega_2(u_2)) = c_4 v^*(u_3, \omega_1(u_1), \omega_2(u_2)) = \\ = \max_{v(.)\in V(.)} l_1(v(u_3, \omega_1(u_1), \omega_2(u_2)), c_4) = l_1(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_4). \end{gathered} \tag{27}$$

Предположим, что максимум этой функции (20) достигается. Тогда:

$$\begin{gathered} \max_{\substack{\omega_1(u_1)\in\Omega_1(u_1) \\ \omega_2(u_2)\in\Omega_2(u_2)}} l_2(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_2) + l_3(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_3) = \\ = l_2(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_2) + l_3(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_3) = \\ = \max_{\substack{\omega_1(u_1)\in\Omega_1(u_1) \\ \omega_2(u_2)\in\Omega_2(u_2)}} c_2 v^*(u_3, \omega_1(u_1), \omega_2(u_2)) + c_3 v^*(u_3, \omega_1(u_1), \omega_2(u_2)) = \\ = c_2 v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)) + c_3 v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)) = \\ = \max_{\substack{\omega_S(u_1,u_2)\in \\ \in\Omega_S(u_1,u_2)}} l_S(v^*(u_3, \omega_1(u_1), \omega_2(u_2)), c_2, c_3) = \\ = l_S(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_2, c_3) = \\ = \max_{\substack{\omega_S(u_1,u_2)\in \\ \in\Omega_S(u_1,u_2)}} c_2 v^*(u_3, \omega_1(u_1), \omega_2(u_2)) + c_3 v^*(u_3, \omega_1(u_1), \omega_2(u_2)) = \\ = c_2 v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)) + c_3 v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)) \end{gathered} \tag{28}$$

Как видно из выражения (28) максимизация функции происходит по двум функциям ω_1 и ω_2, что обеспечивает суммарную полезность игроков B_1 и B_2 в игре Г. Ту же равно-

весную стратегию можно получить, как показано, и с использованием в (28) максимизации функции суммарной полезности по значению коалиционной функции $\omega_S(u_1, u_2)$ в игре $\widetilde{\Gamma}$.

$$\max_{u \in U} l_1(v^*(u_3, \omega_1^*(u_1), \omega_2^*(u_2)), c_1) = l_1(v^*(u_3^*, \omega_1^*(u_1^*), \omega_2^*(u_2^*)), c_1). \tag{29}$$

По предположению того, что максимумы функций (27) и (28) были достигнуты, ситуация равновесия по Нэшу через решение оптимизационных задач может быть найдено через данные формулировки с использованием равновесных оптимальных чистых стратегий в игре $\widetilde{\Gamma}$.

Поскольку кооперативная подыгра подразумевает, что игроки B_1 и B_2 действуют совместно как один игрок, их совокупная полезность остается неразделенной. Для цели дележа совокупной полезности был выбран принцип оптимальности – вектор Шепли.

Формула вычисления вектора Шепли в общем виде выглядит как:

$$Sh_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{s!(n-s-1)!}{n!}(v(S \cup \{i\}) - v(S)) \tag{30}$$

Формула вектора Шепли для определения минимальной гарантированной полезности игроков B_1 и B_2 должна учитывать наихудший для этих игроков случаи. При этом считаем, что игроки $\{B_2; C\}$ для расчета одноэлементной характеристической функции $v(\{B_1\})$ и игроки $\{B_1; C\}$ для расчета одноэлементной характеристической функции $v(\{B_2\})$ будут действовать согласно своим оптимизационным задачам так же, как и в игре Γ.

Можно рассмотреть 3 вида одноэлементной характеристической функции $v(\{B_1\})$ и $v(\{B_2\})$ с 3 различными способами получения минимальной гарантированной полезности игроками B_1 и B_2. При первом виде одноэлементной характеристической функции игрока B_1 (B_2) наихудшим для него случаем является неполучение им вектора ресурса u_1 (u_2) от игрока A_0. Возможен также и другой подход к определению одноэлементной характеристической функции игрока B_1 (B_2). Во втором подходе наихудшим для него вариантом является не только то, что он не получает вектор ресурсов u_1 (u_2) от игрока A_0, но и то, что игрок, находящийся на одном с ним уровне, также не получает вектор ресурсов u_2 (u_1) от игрока A_0. Но возможен также и третий вид одноэлементной характеристической функции игроков B_1 и B_2 при котором наихудший случай для игрока $B_1(B_2)$ состоит не только в неполучении им вектора ресурсов u_1 (u_2) и неполучении вектора ресурса u_2 (u_1) игроком $B_2(B_1)$, находящимся на одном с ним уровне, но и неполучение вектора ресурсов u_3 игроком C, что подразумевает $u_1 = 0$, $u_2 = 0$, $u_3 = 0$.

Рассмотрим подробно случаи, когда игроку B_1 поступает нулевой вектор ресурсов $u_1 = 0$ в ходе решения оптимизационной задачи (18)-(19) игрока A_0 при отправке им векторов ресурсов игроку B_2 и C соответственно $u_2 \geq 0$ и $u_3 \geq 0$. Фактически, это означает, что игрок B_1 распределяет только вектор наличных ресурсов ξ_1. В системе линейных неравенств изменение относительно системы (3) в свободных членах может быть выписано таким образом:

$$\begin{cases} \omega_1^1\alpha_{11}^1 + \cdots + \omega_k^1\alpha_{k1}^1 + \cdots + \omega_m^1\alpha_{m1}^1 \leq \xi_1^1, \\ ..., \\ \omega_1^1\alpha_{1j}^1 + \cdots + \omega_k^1\alpha_{kj}^1 + \cdots + \omega_m^1\alpha_{mj}^1 \leq \xi_j^1, \\ ..., \\ \omega_1^1\alpha_{1l}^1 + \cdots + \omega_k^1\alpha_{kl}^1 + \cdots + \omega_m^1\alpha_{ml}^1 \leq \xi_l^1. \end{cases} \tag{31}$$

Наихудший случай с минимальной гарантированной полезностью для игрока игрока B_2 аналогичен. Рассматривается случай поступления нулевого вектора $u_2 = 0$ в ходе решения оптимизационной задачи (18)-(19) игрока A_0. Игрок B_2 распределяет только вектор

наличных ресурсов ξ_2. В системе линейных неравенств изменение относительно системы (5) в свободных членах может быть выписано в следующем выражении:

$$\begin{cases} \omega_1^2\alpha_{11}^2 + \cdots + \omega_k^2\alpha_{k1}^2 + \cdots + \omega_m^2\alpha_{m1}^2 \le \xi_1^2, \\ ..., \\ \omega_1^2\alpha_{1j}^2 + \cdots + \omega_k^2\alpha_{kj}^2 + \cdots + \omega_m^2\alpha_{mj}^2 \le \xi_j^2, \\ ..., \\ \omega_1^2\alpha_{1l}^2 + \cdots + \omega_k^2\alpha_{kl}^2 + \cdots + \omega_m^2\alpha_{ml}^2 \le \xi_l^2. \end{cases} \tag{32}$$

Такой характер линейных неравенств обосновывается свойством ромбовидной структуры, в которой распределение происходит на последнем этапе игры $\widetilde{\Gamma}$, с получением вектора $v(u_3, \omega_1(u_1), \omega_2(u_2))$. В интерпретации минимальная гарантированная полезность означает поочередное игнорирование производственного подразделения B_1 и B_2 при формировании плана снабжения ресурсами.

Представим первый вид одноэлементной характеристической функции. Построим процесс решения задачи распределения ресурсов в игре Γ с целью поиска ситуации равновесия по Нэшу с учетом положений леммы о существования ситуации равновесия по Нэшу. Данная структура, подобна трехуровневой ромбовидной структуре исходной задачи. Приведем выкладки для случая в котором нуль–вектор отправляется игроком A_0 игроку B_1, то есть $u_1 = 0$. Случай с игроком B_2 аналогичен.

$$\begin{gathered} \max_{\substack{v(u_3,\omega_1(0),\omega_2(u_2))\in \\ \in V(u_3,\omega_1(0),\omega_2(u_2))}} c_1 v(u_3, \omega_1(0), \omega_2(u_2)) = c_1 v^*(u_3, \omega_1(0), \omega_2(u_2)) = \\ = \max_{\substack{v(u_3,\omega_1(0),\omega_2(u_2))\in \\ \in V(u_3,\omega_1(0),\omega_2(u_2))}} l_1(v(u_3, \omega_1(0), \omega_2(u_2)), c_1) = l_1(v^*(u_3, \omega_1(0), \omega_2(u_2)), c_1) = \\ = \widetilde{l_1}(v^*(u_3, \omega_1(0), \omega_2(u_2)), c_1). \end{gathered} \tag{33}$$

Рассмотрим решение оптимизационной задачи игрока B_1 при нулевом векторе u_1 и предположим, что максимум достигается. Случай минимальной гарантированной полезности для игрока B_2 при $u_2 = 0$, но $u_1 \ge 0$ и $u_3 \ge 0$ аналогичен с точностью до замены знаков.

$$\begin{gathered} \max_{\omega_1(0)\in\Omega_1(0)} c_2 v^*(u_3, \omega_1(0), \omega_2(u_2)) = c_2 v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)) = \\ = \max_{\omega_1(0)\in\Omega_1(0)} l_2(v^*(u_3, \omega_1(0), \omega_2(u_2)), c_2) = l_2(v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)), c_2) = \\ = \widetilde{l_2}(v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)), c_2). \end{gathered} \tag{34}$$

Нахождение оптимального значения вектора $\omega_2^*(u_2)$ возможно за счет того, что в игре с полной информацией известно, что $u_1 = 0$ и за счет утверждения леммы. Также, для игрока B_2 решение его оптимизационной задачи даёт следующий равновесный вектор:

$$\begin{gathered} \max_{\omega_2(0)\in\Omega_2(0)} c_3 v^*(u_3, \omega_1(0), \omega_2(u_2)) = c_3 v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)) = \\ = \max_{\omega_2(0)\in\Omega_2(0)} l_3(v^*(u_3, \omega_1(0), \omega_2(u_2)), c_3) = l_3(v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)), c_3) = \\ = \widetilde{l_3}(v^*(u_3, \omega_1^*(0), \omega_2^*(u_2)), c_3). \end{gathered} \tag{35}$$

В данном случае игрок A_0, в отличии от первого сценария, является активным, вследствие чего его равновесная стратегия при достижении максимумов предыдущих задач:

$$\max_{\substack{0=u_1\in U\\ 0\le u_i\in U\\ i=2,3}} c_1 v^*(u_3^*,\omega_1^*(0),\omega_2^*(u_2^*)) = \max_{\substack{0=u_1\in U\\ 0\le u_i\in U\\ i=2,3}} = l_4(v^*(u_3^*,\omega_1^*(0),\omega_2^*(u_2^*))),c_4) = \\ = \widetilde{l_4}(v^*(u_3^*,\omega_1^*(0),\omega_2^*(u_2^*))),c_4). \tag{36}$$

Рассмотрим второй вид одноэлементной характеристической функции. Построим процесс решения задачи распределения ресурсов в игре Γ с целью поиска ситуации равновесия по Нэшу с учетом положений леммы о существования ситуации равновесия по Нэшу. Данная структура, подобна трехуровневой ромбовидной структуре исходной задачи. Приведем выкладки для случая в котором нуль–вектор отправляется игроком A_0 игроку B_1, то есть $u_1 = 0$. Случай с игроком B_2 аналогичен.

$$\max_{\substack{v(u_3,\omega_1(0),\omega_2(0))\in\\ \in V(u_3,\omega_1(0),\omega_2(0))}} c_1 v(u_3,\omega_1(0),\omega_2(0)) = c_1 v^*(u_3,\omega_1(0),\omega_2(0)) = \\ = \max_{\substack{v(u_3,\omega_1(0),\omega_2(0))\in\\ \in V(u_3,\omega_1(0),\omega_2(0))}} l_1(v(u_3,\omega_1(0),\omega_2(0)),c_1) = l_1(v^*(u_3,\omega_1(0),\omega_2(0)),c_1) = \\ = \widehat{l_1}(v^*(u_3,\omega_1(0),\omega_2(0)),c_1). \tag{37}$$

Рассмотрим решение оптимизационной задачи игрока B_1 при нулевом векторе u_1 и u_2 и предположим, что максимум достигается. Случай минимальной гарантированной полезности для игрока B_2 при $u_2 = 0$, но $u_1 \ge 0$ и $u_3 \ge 0$ аналогичен с точностью до замены знаков.

$$\max_{\omega_1(0)\in\Omega_1(0)} c_2 v^*(u_3,\omega_1(0),\omega_2(0)) = c_2 v^*(u_3,\omega_1^*(0),\omega_2^*(0)) = \\ = \max_{\omega_1(0)\in\Omega_1(0)} l_2(v^*(u_3,\omega_1(0),\omega_2(0)),c_2) = l_2(v^*(u_3,\omega_1^*(0),\omega_2^*(0)),c_2) = \\ = \widehat{l_2}(v^*(u_3,\omega_1^*(0),\omega_2^*(0)),c_2). \tag{38}$$

Нахождение оптимального значения вектора $\omega_2^*(u_2)$ возможно за счет того, что в игре с полной информацией известно, что $u_1 = 0$ и $u_2 = 0$ и за счет утверждения леммы. Также, для игрока B_2 решение оптимизационной задачи его даёт следующий равновесный вектор:

$$\max_{\omega_2(0)\in\Omega_2(0)} c_3 v^*(u_3,\omega_1(0),\omega_2(0)) = c_3 v^*(u_3,\omega_1^*(0),\omega_2^*(0)) = \\ = \max_{\omega_2(0)\in\Omega_2(0)} l_3(v^*(u_3,\omega_1(0),\omega_2(0)),c_3) = l_3(v^*(u_3,\omega_1^*(0),\omega_2^*(0)),c_3) = \\ = \widehat{l_3}(v^*(u_3,\omega_1^*(0),\omega_2^*(0)),c_3). \tag{39}$$

В данном случае игрок A_0, в отличии от первого сценария, является активным, вследствие чего его равновесная стратегия при достижении максимумов предыдущих задач:

$$\max_{\substack{0=u_1\in U\\ 0\le u_i\in U\\ i=2,3}} c_1 v^*(u_3^*,\omega_1^*(0),\omega_2^*(0)) = \max_{\substack{0=u_1\in U\\ 0\le u_i\in U\\ i=2,3}} = l_4(v^*(u_3^*,\omega_1^*(0),\omega_2^*(0))),c_4) = \\ = \widehat{l_4}(v^*(u_3^*,\omega_1^*(0),\omega_2^*(0))),c_4). \tag{40}$$

Рассмотрим третий вид характеристической функции. Построим процесс решения задачи распределения ресурсов в игре Γ с целью поиска ситуации равновесия по Нэшу с учетом положений леммы о существования ситуации равновесия по Нэшу. Структура минимальной гарантированной полезности в этом случае подобна двухуровневой структуре при пассивном участии игрока A_0, вклад которого в процесс минимален. Рассмотрим характеристическую функцию для игрока B_1 и Обозначим, что $v(.) = v(0,\omega_1(0),\omega_2(0))$, $V(.) = V(0,\omega_1(0),\omega_2(0))$. Случай с игроком B_2 аналогичен.

$$\max_{\substack{v(0,\omega_1(0),\omega_2(0))\in\\ \in V(0,\omega_1(0),\omega_2(0))}} c_1 v(0,\omega_1(0),\omega_2(0)) = c_1 v^*(0,\omega_1(0),\omega_2(0)) =$$
$$= \max_{\substack{v(0,\omega_1(0),\omega_2(0))\in\\ \in V(0,\omega_1(0),\omega_2(0))}} l_1(v(0,\omega_1(0),\omega_2(0)),c_1) = l_1(v^*(0,\omega_1(0),\omega_2(0)),c_1) = \quad (41)$$
$$= \overline{l_1}(v^*(0,\omega_1(0),\omega_2(0)),c_1).$$

В следующий шаг обратным проходом рассмотрим нахождение равновесных стратегий в игре Γ при нуль-векторах u_i, $i = 1,2,3$. Рассмотрим решение оптимизационной задачи игрока B_1 при нулевом векторе u_1 и предположим, что максимум достигается, тогда:

$$\max_{\omega_1(0)\in\Omega_1(0)} c_2 v^*(0,\omega_1(0),\omega_2(0)) = c_2 v^*(0,\omega_1^*(0),\omega_2^*(0)) =$$
$$= \max_{\omega_1(0)\in\Omega_1(0)} l_2(v^*(0,\omega_1(0),\omega_2(0)),c_2) = l_2(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_2) = \quad (42)$$
$$= \overline{l_2}(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_3)),c_2).$$

Для игрока B_2, аналогично, рассмотрим решение оптимизационной задачи при нулевом векторе u_2 и предположим, что максимум достигается, тогда:

$$\max_{\omega_2(0)\in\Omega_2(0)} c_3 v^*(0,\omega_1(0),\omega_2(0)) = c_3 v^*(0,\omega_1^*(0),\omega_2^*(0)) =$$
$$= \max_{\omega_2(0)\in\Omega_2(0)} l_3(v^*(0,\omega_1(0),\omega_2(0)),c_3) = l_3(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_3) = \quad (43)$$
$$= \overline{l_3}(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_3).$$

В данном случае обеспечивается не-суммарная полезность игроков B_1 и B_2. Нахождение равновесия по Нэшу облегчается строгой определенностью значений вектора ресурсов u_i, $i = 1,2,3$ для игроков B_1 и B_2. Тогда функция полезности игрока A_0 при равновесных стратегия игроков C, B_1 и B_2 будет равна:

$$\max_{\substack{0=u_i\in U\\ i=1,2,3}} c_1 v^*(0,\omega_1^*(0),\omega_2^*(0)) = \max_{\substack{0=u_i\in U\\ i=1,2,3}} = l_4(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_4) =$$
$$= \overline{l_4}(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_4). \quad (44)$$

Проведем сравнительный анализ трех характеристических функций для получения минимальной гарантированной полезности игроков B_1 и B_2 для чего выпишем справедливые для них неравенства, исходя из данной функции полезности игры Γ.

$$\widetilde{K_f}(u_3^*,\omega_1^*(0),\omega_2^*(u_2^*)) = \widetilde{l_f}(v^*(u_3^*,\omega_1^*(0),\omega_2^*(u_2^*)),c_f) \geq$$
$$\geq \widehat{K_f}(u_3^*,\omega_1^*(0),\omega_2^*(0)) = \widehat{l_f}(v^*(u_3^*,\omega_1^*(0),\omega_2^*(0)),c_f) \geq \quad (45)$$
$$\geq \overline{K_f}(0,\omega_1^*(0),\omega_2^*(0)) = \overline{l_f}(v^*(0,\omega_1^*(0),\omega_2^*(0)),c_f)$$

Тем самым подтверждается, что значения целевых функций каждого игрока A_0, B_1, B_2, C при $u_i = 0$, $i = 1,2,3$ меньше, чем значения целевых функций каждого игрока при $u_i \geq 0$, $i = 1,3$, либо $i = 2,3$.

Формулы вектора Шепли (30) для первого, второго и третьего подхода к определению одноэлементной характеристической игрока $B_1(B_2)$ для коалиции $S = \{B_1,B_2\}$ будут выглядеть одинаково с учетом проекции игроков B_1 и B_2 как одного игрока:

$$Sh_{B_1}(v) = v(\{B_1\}) + \frac{v(S) - v(\{B_1\}) - v(\{B_2\})}{2}$$
$$Sh_{B_2}(v) = v(\{B_2\}) + \frac{v(S) - v(\{B_1\}) - v(\{B_2\})}{2} \quad (46)$$

1.3. Формулировка кооперативной игры

Важным аспектом теоретико-игровой модели с ромбовидной иерархической структурой является кооперативная составляющая. Кооперативная игра должна учитывать особенности ромбовидной структуры игры Γ. Рассмотрим проблематику коалиционных разбиений на множестве игроков A_0, B_1, B_2, C игры Γ.

Введем кооперативную игру (N, v), где $N = \{A_0, B_1, B_2, C\}$, а v - характеристическая функция: $2^N \to \mathbb{R}$, $v(\emptyset) = 0$. Применим максиминный подход для каждой коалиции $T \subset \{A_0, B_1, B_2, C\}$ и определим $v(T)$ как наибольшую гарантированную полезность коалиции T в антагонистической игре между коалицией T и коалицией $T' = \{A_0, B_1, B_2, C\} \backslash T$. Предположим, что существует такой вектор продукции $v_0 \in V(u_3, \omega_1(u_1), \omega_2(u_2))$ для всех $u_1, u_2, u_3, \omega_1, \omega_2$ и существуют такие вектора наличных ресурсов $\xi_1 \geq 0$, $\xi_2 \geq 0$, $\gamma_1 \geq 0$, $\gamma_2 \geq 0$ в правой части систем линейных неравенств (3), (5) и (8), что $l_i(v_0(.)) = 0$, $i = 1, 2, 3, 4$, $v(.) = v(u_3, \omega_1(u_1), \omega_2(u_2))$.

Рассмотрим два вида коалиций: первый исключает игрока C, $T : C \notin T$; второй включает игрока C, $T : C \in T$.

Подразумевается, что в первом случае $T \subset \{A_0, B_1, B_2\}$, а игрок C является членом коалиции $N \backslash T$ и может выбрать, при векторах наличных ресурсов $\xi_1 \geq 0$, $\xi_2 \geq 0$, $\gamma_1 \geq 0$, $\gamma_2 \geq 0$ в правой части систем линейных неравенств (3), (5) и (8), стратегию $v_0 : l_i(v_0(.)) = 0$, $i = 1, 2, 3, 4$, $v(.) = v(u_3, \omega_1(u_1), \omega_2(u_2))$, вследствие чего $v(T) = 0$.

Во втором случае определим характеристическую функцию $v(T)$ следующими равенствами и предположим, что в них максимумы и минимумы достигаются.

Рассмотрим коалицию $T = \{C\}$. Как и прежде, считаем, что $\sum_{i=1}^{3} u_i = u \leq b$.

1) $T = \{C\}$

$$v(T) = \min_{u \in U} \min_{\omega_1 \in \Omega_1(u_1)} \min_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} l_4(v(.)) \tag{47}$$

Рассмотрим также двухэлементные и трехэлементные коалиции.

2) $T = \{A_0, C\}$

$$v(T) = \min_{u_1 \in U} \min_{u_2 \in U} \max_{u_3 \in U} \min_{\omega_1 \in \Omega_1(u_1)} \min_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_1(v(.)) + l_4(v(.))) \tag{48}$$

3) $T = \{B_1, C\}$.

$$v(T) = \min_{u \in U} \max_{\omega_1 \in \Omega_1(u_1)} \min_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_2(v(.)) + l_4(v(.))) \tag{49}$$

4) $T = \{B_2, C\}$.

$$v(T) = \min_{u \in U} \min_{\omega_1 \in \Omega_1(u_1)} \max_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_3(v(.)) + l_4(v(.))) \tag{50}$$

5) $T = \{B_1, B_2, C\}$.

$$v(T) = \min_{u \in U} \max_{\omega_1 \in \Omega_1(u_1)} \max_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_2(v(.)) + l_3(v(.)) + l_4(v(.))) \tag{51}$$

6) $T = \{A_0, B_1, C\}$.

$$v(T) = \max_{u_1 \in U} \min_{u_2 \in U} \max_{u_3 \in U} \max_{\omega_1 \in \Omega_1(u_1)} \min_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_1(v(.)) + l_2(v(.)) + l_4(v(.))) \tag{52}$$

7) $T = \{A_0, B_2, C\}$.

$$v(T) = \max_{u_1 \in U} \min_{u_2 \in U} \max_{u_3 \in U} \min_{\omega_1 \in \Omega_1(u_1)} \max_{\omega_2 \in \Omega_2(u_2)} \max_{v \in V(u_3, \omega_1(u_1), \omega_2(u_2))} (l_1(v(.)) + l_3(v(.)) + l_4(v(.))) \tag{53}$$

8) $T = N = \{A_0, B_1, B_2, C\}$.

$$v(T) = \max_{u\in U} \max_{\omega_1\in\Omega_1(u_1)} \max_{\omega_2\in\Omega_2(u_2)} \max_{v\in V(u_3,\omega_1(u_1),\omega_2(u_2)} \sum_{i=1}^{4} l_i(v(.)) \tag{54}$$

Таким образом, определенные характеристические функции через оптимизации целевых функций $l_i(v(.))$ обладают свойством супераддитивности, то есть для любых коалиций $T, R \subset \{A_0, B_1, B_2, C\}$, для которых $T \cap R = \emptyset$, имеет место неравенство $v(T \cup R) \geq v(T) + v(R)$.

Формулировка вектора Шепли (30) для заданной кооперативной игры (N,v) имеет ряд отличительных элементов, описанных в (47)-(54).

1)

$$\begin{gathered} Sh_{A_0}(v) = \frac{v(\{A_0\})}{4} + \frac{v(\{A_0, C\}) - v(\{A_0\}) - v(\{C\})}{12} + \\ + \frac{v(\{A_0, B_1, C\}) - v(\{B_1, C\}) + v(\{A_0, B_2, C\}) - v(\{B_2, C\})}{12} + \\ + \frac{v(\{A_0, B_1, B_2, C\}) - v(\{B_1, B_2, C\})}{4}. \end{gathered} \tag{55}$$

2)

$$\begin{gathered} Sh_{B_1}(v) = \frac{v(\{B_1\})}{4} + \frac{v(\{B_1, C\}) - v(\{B_1\}) - v(\{C\})}{12} + \\ + \frac{v(\{B_1, B_2, C\}) - v(\{B_2, C\}) + v(\{A_0, B_1, C\}) - v(\{A_0, C\})}{12} + \\ + \frac{v(\{A_0, B_1, B_2, C\}) - v(\{A_0, B_2, C\})}{4}. \end{gathered} \tag{56}$$

3)

$$\begin{gathered} Sh_{B_2}(v) = \frac{v(\{B_2\})}{4} + \frac{v(\{B_2, C\}) - v(\{B_2\}) - v(\{C\})}{12} + \\ + \frac{v(\{B_1, B_2, C\}) - v(\{B_1, C\}) + v(\{A_0, B_2, C\}) - v(\{A_0, C\})}{12} + \\ + \frac{v(\{A_0, B_1, B_2, C\}) - v(\{A_0, B_1, C\})}{4}. \end{gathered} \tag{57}$$

4)

$$\begin{gathered} Sh_C(v) = \frac{v(\{C\})}{4} + \\ + \frac{v(\{A_0, C\}) - v(\{A_0\}) - v(\{C\}) + v(\{B_1, C\}) - v(\{B_1\}) - v(\{C\})}{12} + \\ + \frac{v(\{B_2, C\}) - v(\{B_2\}) - v(\{C\})}{12} + \\ + \frac{v(\{B_1, B_2, C\}) - v(\{B_1, B_2\}) + v(\{A_0, B_1, C\}) - v(\{A_0, B_1\})}{12} + \\ + \frac{v(\{A_0, B_2, C\}) - v(\{A_0, B_2\})}{12} + \\ + \frac{v(\{A_0, B_1, B_2, C\}) - v(\{A_0, B_1, B_2\})}{4}. \end{gathered} \tag{58}$$

Вектор Шепли для кооперативной игры (N, v) подразумевает, что значение характеристической функции v от одноэлементной коалиции $v(\{i\}) = 0$, $i = A_0, B_1, B_2$ поэтому их

рассмотрение имеет формальный интерес. Тем ни менее, сформулированная кооперативная игра (N, v) и принцип оптимальности распределения дележа подтверждают возможность экспериментального апробирования данной модели.

В первой главе были рассмотрены теоретико-игровые математические аспекты построения неантагонистической иерархической ромбовидной игры в нормальной форме в чистых стратегиях Г, рассмотрен и доказан факт существования в данной игре ситуации равновесия по Нэшу через построение задач линейного и нелинейного программирования с параметрами. Показана возможность оптимального дележа коалиции $S = \{B_1, B_2\}$ с использованием вектора Шепли. Определена кооперативная игра с использованием максиминного подхода (N, v) как антагонистической игры двух коалиций: T и $T' = \{A_0, B_1, B_2, C\} \backslash T$. Для кооперативной игры также были построены формулы расчета вектора Шепли.

Глава II Численный эксперимент

2.1. Модификация метода Монте-Карло и алгоритм программы

В данной главе будут рассмотрены аспекты численного эксперимента путём программной реализации модели ромбовидной иерархической структуры, для чего будет подробно описан примененный метод и определены свойства алгоритма.

Метод Монте-Карло относится к группе методов, изучающих случайные стохастические процессы, в том числе с использованием случайных чисел. Сам процесс изучения заключается в генерации случайных величин для чего применяется генератор случайных чисел. После чего многократно итеративно вычисляется возможная область значений случайной величины, границы которой варьируются с учетом необходимой точности. Метод часто используется при численном экспериментировании случайных величин благодаря многочисленным модификациям, дающих достаточные для решения задач результаты.

Для численного нахождения ситуации равновесия по Нэшу в игре в нормальной форме Г предстоит руководствоваться соображениями, данными в (18)-(32). Рассмотрим алгоритм, построенный на основании выводов первой главы для нахождения ситуации равновесия по Нэшу численно. Подробно будет учтен метод Монте-Карло в соответствующих частях алгоритма. Каждая часть алгоритма будет отображать функциональную часть программы или ее процедуры, и являться исходя из логики, данной в (18)-(32), алгоритмом функции. В конечном рассмотрении будет представлен полный алгоритм программной реализации задачи.

Программная реализация по алгоритму была выполнена на языке программирования *python 3.0* с использованием программных функций библиотеки *cvxopt* в интегрированной среде разработки *Jupyter*. В качестве решателя, реализующего симплекс-метод для решения задачи линейного программирования и модифицированный симплекс-метод для решения задачи линейного программирования с параметрами был использован *glpk*. В качестве основной методологии программирования был использована объектно-ориентированная парадигма с частичным использованием функционального подхода в рамках структуризации программы. Тем самым подразумевается, что задача является классом, а функции – методами класса. Методы класса принимают в качестве аргументов известные данные и неизвестные параметры задачи. Программа структурно подразделяется на три части: в первой части выполняется загрузка данных и вычисление массива V^*, после чего производится основная стохастическая процедура метода Монте-Карло с генерацией случайных векторов u_1, u_2, u_3, с учетом ограничения: $u_1 + u_2 + u_3 = b$ и генерацией векторов ω_1 и ω_2 по модифицированному методу Монте-Карло, также происходит перестановка элементов массивов состоящих из векторов ω_1 и ω_2 по одному набору u_1, u_2, u_3 для отправки их в правые части системы линейных неравенств; во второй части с целью решения оптимизационной задачи и получения массива векторов v и значений целевых функций производится применения симплекс-метода к полученным массивам; в третьей части происходит проверка полученных решений на оптимальность, получение оптимального целевого вектора v^* и значений целевых функций игроков A_0, B_1, B_2, C, в том числе с использованием вектора Шепли для коалиции $S = \{B_1, B_2\}$.

Последовательно рассмотрим аспекты первой части выполнения программы Рассмотрим первую функцию, сохраняющую данные.

В качестве неизвестных параметров выступают векторы ресурсов, промежуточной продукции и конечной продукции: u_1, u_2, u_3, ω_1, ω_2, v. Они являются целевыми переменными соответствующих задач линейного и нелинейного программирования с параметрами.

В качестве известных данных и известных параметров для программной реализации выступают: вектора доходов игроков A_0, B_1, B_2, C, то есть, векторы: c_1, c_2, c_3, c_4; векторы наличных ресурсов игроков B_1, B_2, C, то есть векторы: ξ_1, ξ_2, γ; технологические матрицы

Функция 1 Сохранение данных

1: **function** TASK DATA($[1, ..., q]$, $[1, ..., q]$, $[1, ..., q]$, $[1, ..., q]$, $[1, ..., l]$, $[1, ..., l]$, $[1, ..., l]$,$[1, ..., l]$, $[[1, ..., m], ..., l]$, $[[1, ..., m], ..., l]$, $[[1, ..., q], ..., l]$, $[1, ..., q]$)
2: $c_1 \leftarrow [1, ..., q]$
3: $c_2 \leftarrow [1, ..., q]$
4: $c_3 \leftarrow [1, ..., q]$
5: $c_4 \leftarrow [1, ..., q]$
6: $\xi_1 \leftarrow [1, ..., l]$
7: $\xi_2 \leftarrow [1, ..., l]$
8: $\gamma_1 \leftarrow [1, ..., l]$
9: $\gamma_2 \leftarrow [1, ..., l]$
10: $A_1 \leftarrow [[1, ..., m], ..., l]$
11: $A_2 \leftarrow [[1, ..., m], ..., l]$
12: $D \leftarrow [[1, ..., q], ..., l]$
13: $b \leftarrow [1, ..., l]$
14: Return: c_1, c_2, c_3, c_4, ξ_1, ξ_2, γ_1,γ_2, A_1, A_2, D, b
15: **end function**

игроков B_1, B_2, C, а именно: A_1, A_2, D; и общий вектор ресурсов игрока A_0, b.

Для перспективной возможности внесения или изменения известных данных с файла, известные данные оформлены в отдельную программную функцию *Task data*. Функция предназначена для сохранения данных для дальнейшего использования в программе.

По алгоритму функции 1 видно, что она выполняет присваивание значения объектов, которые принимает в роли аргументов списочный тип данных. Функция возвращает присвоенные переменным значения. В итоге, после вызова функции из области её определения она вернёт переменные c_1, c_2, c_3, c_4, ξ_1, ξ_2, γ_1,γ_2, A_1, A_2, D, b и их значения из аргументов функции, которые были даны ей на вход.

Будем использовать далее выходные данный этой функции и качестве входных данных других функций в случае, если понадобится расчет представленных переменных.

Поскольку для численной программной реализации с целью нахождения ситуации равновесия по Нэшу был использован обратный проход по ромбовидной структуре, то есть, начиная с решения задачи (25)-(26), то, следовательно, алгоритм следующей функции будет реализовывать её, в соответствии с (27). Имеется ввиду, что при решении задачи линейного программирования с параметрами ω_1, ω_2 и u_3 будут подбираться всевозможные значения данных параметров при решении. Для этой цели была разработана функция *Solving optimization problems of V*, вычисляющая симплекс-методом в задаче линейного программирования с параметрами с использованием решателя *glpk* по входным аргументам D и γ оптимальное решение v^*, соответствующее этому решению значения параметров ω_1, ω_2 и u_3 и значение целевой функции.

Структура функции *Solving optimization problems of V* заключается в использовании цикла *for* для решения задачи линейного параметрического программирования путем нахождения оптимального значения параметров с критерием останова в виде значения параметра цикла n, $i \leq n$.

Первым шагом объявляется функция. Вторым и третьим шагом 2-3 в теле функции выполняется подключение (импортирование) библиотеки *cvxopt*, которая является основной при формулировке и решении оптимизационных задач симплекс-методом. Шаги функции 6, 7 и 8 присваивают соответствующей переменной значение в форме списка. На шаге 9 переменной присваивается абстрактное ограничение из множества стратегий игрока C. Шаги $10 - 14$ присваивают ограничение неотрицательности переменным. И, наконец, шаг

Функция 2 Предварительная оптимизация задачи игрока С

function SOLVING OPTIMIZATION PROBLEMS OF $\mathrm{V}(D, \gamma, c_4)$
    import cvxopt
    $X \leftarrow n$
    Return V^*
    for all $i \leftarrow X$: **do**
        $ineq_1 \leftarrow D * v \leq \omega_1 + \omega_2 + \gamma_1$
        $ineq_2 \leftarrow D * v \leq u_3 + \gamma_2$
        $ineq_3 \leftarrow u_3 \geq 0$
        $ineq_5 \leftarrow \omega_1 \geq 0$
        $ineq_6 \leftarrow \omega_2 \geq 0$
        $ineq_6 \leftarrow v \geq 0$
        $ineq_7 \leftarrow \gamma_1 \geq 0$
        $ineq_8 \leftarrow \gamma_2 \geq 0$
        $lp \leftarrow c_4 * v = max\ s.t.\ ineq_1,\ ineq_2,\ ineq_3,\ ineq_4,\ ineq_5,\ ineq_6, ineq_7, ineq_8$
    end for
    Return v^*
end function

15 рассматривает присвоение переменной результата максимизации скалярного произведения при данных выше ограничениях. Функция возвращает двумерный массив оптимальных векторов v.

Однако, несмотря на нахождение оптимального решения и значений параметров, вывод данной функции *Solving optimization problems of V* не соотносится с решениями предыдущих задач нелинейного программирования (18)-(19), (20)-(21) и (25)-(26). Это связано с тем, что случайный поиск решения методом Монте-Карло только на первом этапе не учитывает в b – верхней границе отрезка – решения системы линейных неравенств из задач нелинейного программирования. Второй причиной является то, что классическим симплекс-методом с возможность декомпозиции задач решить данную задачу затруднительно, ввиду того, что целевые функции задач (18)-(19), (20)-(21) и (25)-(26) зависят от существенно разрывной нелинейной функции $v(u_3, \omega_1(u_1), \omega_2(u_2)$, максимизация которой классическими методами затруднительна.

Для решения данной проблемы следует рассмотреть задачу (20)-(21) среднего уровня игроков B_1 и B_2. Поскольку максимизация целевой функции этих игроков, состоящих в коалиции $S = \{B_1, B_2\}$ возможна только при полученном v, то локальное решение этих задач невозможно.

Для разрешения данной проблематики был модифицирован метод Монте-Карло для поиска множества решений систем линейных неравенств (21). Сущность метода заключается в следующем. Сначала происходит фиксация векторов u_1 и u_2 при данных векторах ξ_1 и ξ_2 для (21) соответственно. На следующем этапе каждая величина – сумма компонент u_1 и ξ_1 (u_2 и ξ_2) – делится на каждый элемент данной ей строки матрицы A_1 (A_2). В полученной новой матрице $A_1^{'}$ ($A_2^{'}$) находится max по строке. Таким образом, значение максимума в строке $\alpha_1^{'}$ ($\alpha_2^{'}$) этой матрицы является верхней границей отрезка равномерного распределения случайной величины $\omega_1 \sim U[0, \alpha_1^{'}]$ ($\omega_2 \sim U[0, \alpha_2^{'}]$). На следующем этапе генерируется массив случайных чисел из равномерного распределения с заданным верхними граничными точками отрезка, при этом верхние граничные точки обеспечивают полное покрытие сетки допустимых решений систем линейных неравенств (21).

Представим более подробное описание модификации метода Монте-Карло, при котором обеспечивается полное покрытие области допустимых решений с помощью верхней

границы сегмента равновероятной случайной величины. Определим первый этап метода. Из Главы I известно, что матрица $A_1 \in R^{lm}$ и $A_2 \in R^{lm}$, а $u_1 \in R^m$ и $u_2 \in R^m$. Тогда можно ввести матрицу $H \in R^{lm}$ такую, что:

$$\{(h^i_{kj})^{m,l}_{k=1,j=1} = H_i : h^i_{kj} = \frac{u^i_j + \xi^i_j}{\alpha^i_{kj}}, \\ \forall(u^i_j) \in u_i, \forall\xi^i_j \in \xi_i, \forall(\alpha^i_{kj}) \in A_i, i = 1,2; j = \overline{1,l}; k = \overline{1,m}\} \tag{59}$$

Элементы матрицы H_i являются результатом отношения суммы компонентов векторов u_i и ξ_i к каждому элементу соответствующего номера строки матрицы A_i, $i = 1,2$. Матрица задана той же размерности, что и матрица A_i, $i = 1,2$. Далее определим второй этап метода, который заключается в нахождении максимального элемента строки матрицы.

$$\max_{k=\overline{1,m}} H_i = (h^i_{kj})^{m,l}_{k=1,l=1} = (h^{*i}_k)^m_{k=1} \tag{60}$$

Таким образом, реализуется m задач (59) по нахождению максимума в каждом столбце матрицы H_i. В результате получается вектор $h^*_i = (h^{*i}_1, ..., h^{*i}_m)$ с компонентами-максимумами по каждому столбцу матрицы H_i. Третьим этапом метода является подстановка каждого компонента вектора h^* в качестве верхней границы сегмента для реализации равновероятной случайной величины $\omega_i(u_i)$, $i = 1,2$. Пусть $\widehat{\omega^i} = (0, ..., \widehat{\omega^i_k}, ..., 0)$, тогда должно быть справедливо:

$$\{\widehat{\omega^i_k} \sim U[0, h^{*i}_k] : 0\alpha^i_{1j} + \cdots + \widehat{\omega^i_k}\alpha^i_{kj} + \cdots + 0\alpha^i_{mj} = u^i_j + \xi^i_j, i = 1,2; j = \overline{1,l}; k = \overline{1,m}\} \tag{61}$$

Утверждение. Если существует вектор $h^*_i = (h^{*i}_1, ..., h^{*i}_m)$, тогда компонента вектора h^{*i}_k, $k = 1, ..., m$ является верхней границей минимального сегмента равномерного распределения для реализации случайной величины $\widehat{\omega^i_k}$.

Доказательство. Достаточность. Пусть существует ϵ, $0 < \epsilon \leq h^{*i}_k$ такой, что для любого ϵ и для любого h^{*i}_k, $k = 1, ..., m$, величина $\widehat{h^i_k} = h^{*i}_k - \epsilon$ существует. При реализации случайной величины из равномерного распределения $\widehat{\omega^i_k} \sim U[0, \widehat{h^i_k}]$ и при этом $\widehat{\omega^i} = (0, ..., \widehat{\omega^i_k}, ..., 0)$ будет справедливо неравенство:

$$\{\widehat{\omega^i_k} \sim U[0, \widehat{h^i_k}] : 0\alpha^i_{1j} + \cdots + \widehat{\omega^i_k}\alpha^i_{kj} + \cdots + 0\alpha^i_{mj} \leq u^i_j + \xi^i_j, i = 1,2; j = \overline{1,l}; k = \overline{1,m}\} \tag{62}$$

Следовательно, $\widehat{h^i_k}$ не является верхней границей минимального сегмента равномерного распределения для реализации случайной величины ω^i_k в области допустимых решений системы линейных неравенств.

Пусть существует ϵ, для которого справедливо $\epsilon \geq h^{*i}_k$. Тогда при $\widehat{\omega^i} = (0, ..., \widehat{\omega^i_k}, ..., 0)$ будет справедливо неравенство:

$$\{\widehat{\omega^i_k} \sim U[0, \epsilon] : 0\alpha^i_{1j} + \cdots + \widehat{\omega^i_k}\alpha^i_{kj} + \cdots + 0\alpha^i_{mj} \geq u^i_j + \xi^i_j, i = 1,2; j = \overline{1,l}; k = \overline{1,m}\} \tag{63}$$

Следовательно, ϵ не является верхней границей минимального сегмента равномерного распределения для реализации случайной величины ω^i_k в области допустимых решений системы линейных неравенств, поскольку есть меньшее, удовлетворяющее равенству (61) число h^{*i}_k.

Необходимость. Пусть существует вектор $h^*_i = (h^{*i}_1, ..., h^{*i}_m)$, с компонентами h^{*i}_k, $k = 1, ..., m$. При этом $\widehat{\omega^i} = (0, ..., \widehat{\omega^i_k}, ..., 0)$, тогда справедливо:

$$\{\widehat{\omega_k^i} \sim U[0, h_k^{*i}] : 0\alpha_{1j}^i + \cdots + \widehat{\omega_k^i}\alpha_{kj}^i + \cdots + 0\alpha_{mj}^i = u_j^i + \xi_j^i, i = 1, 2; j = \overline{1, l}; k = \overline{1, m}\} \quad (64)$$

Утверждение доказано.

Проиллюстрируем данный факт на численном примере. Зададим матрицу A.

$$A = \begin{pmatrix} 7 & 11 & 3 \\ 0 & 5 & 2 \\ 9 & 1 & 2 \end{pmatrix} \quad (65)$$

Зададим фиксированный вектор ресурсов игрока A_0, u, вектор наличных ресурсов ξ и вектор суммарных ресурсов.

$$\begin{gathered} u = (20, 25, 15) \\ \xi = (10, 15, 35) \\ u + \xi = (30, 40, 50) \end{gathered} \quad (66)$$

Таким образом, система линейных неравенств примет вид:

$$\begin{cases} 7\omega_1 + 11\omega_2 + 3\omega_3 \leq 30, \\ 5\omega_2 + 2\omega_3 \leq 40, \\ 9\omega_1 + \omega_2 + 2\omega_3 \leq 50, \end{cases} \quad (67)$$

Матрица H в данном случае будет равна:

$$H = \begin{pmatrix} \frac{30}{7} & \frac{30}{11} & \frac{30}{3} \\ 0 & \frac{40}{5} & \frac{40}{2} \\ \frac{50}{9} & \frac{50}{1} & \frac{50}{2} \end{pmatrix} = \begin{pmatrix} 4.28 & 2.72 & 10 \\ 0 & 8 & 20 \\ 5.5 & 50 & 25 \end{pmatrix} \quad (68)$$

Путем нахождения максимального значения в каждом столбце матрицы, получим вектор h^*.

$$h^* = (5.5, 50, 25) \quad (69)$$

Способом подстановки полученных компонентов как верхних границ сегмента равномерного распределения $U[a, b]$ для случайной величины $\widehat{\omega_k} \sim U[0, h_k^*]$ получим возможное решение данной системы линейных неравенств:

$$\begin{cases} 7\widehat{\omega_1} + 11\widehat{\omega_2} + 3\widehat{\omega_3} \leq 30, \\ 5\widehat{\omega_2} + 2\widehat{\omega_3} \leq 40, \\ 9\widehat{\omega_1} + \widehat{\omega_2} + 2\widehat{\omega_3} \leq 50, \end{cases} \quad (70)$$

Перейдем в алгоритмическому описанию этого этапа первой части выполнения программы. Сначала представим генерацию набора векторов u_1, u_2, u_3, после чего опишем процесс генерации массива векторов ω_1 и ω_1 при каждом таком полученном наборе векторов u_1, u_2, u_3 и для этого также опишем под-функцию, вычисляющую верхние граничные точки отрезка равномерного распределения из которых будут генерировать случайные числа для каждого вектора ω_1 и ω_1.

Функция 3 Генерация случайных векторов u_1, u_2, u_3

```
1: function RANDOM U(b)
2:     import random
3:     Array_1 ← [...]
4:     Array_2 ← [...]
5:     size b ← dim
6:     for all i ← dim : do
7:         k ← 0
8:         for all j ← 3 : do
9:             x ← random(0; b[i] − k)
10:            Array_1 ← x
11:            k+ = h
12:        end for
13:        c = b[i] − k
14:        for all g ← (dim − dim/i + 1, dim) : do
15:            Array_2 ← Array_1[g] + c/3
16:        end for
17:        u_1 ← [...]
18:        u_2 ← [...]
19:        u_3 ← [...]
20:        for all i ← (0, size Array_2, 3) : do
21:            u_1 ← i
22:        end for
23:        for all i ← (1, size Array_2, 3) : do
24:            u_2 ← i
25:        end for
26:        for all i ← (2, size Array_2, 3) : do
27:            u_3 ← i
28:        end for
29:    end for
30:    Return u_1, u_2, u_3
31: end function
```

Поскольку в рамках рассмотренной задачи значения векторов u_1, u_2, u_3 не являются константами, то следует рассмотреть процедуру случайного распределения значений этих векторов ресурсов, которые в сумме по компонентам дают вектор b.

Функция *Random u* принимает в качестве аргумента функции вектор b как список, состоящий из элементов заданной длины. На первом этапе в среде объявляется данная функция. Для проведения случайного процесса проводится подключение библиотеки *random* с соответствующими библиотечными функциями. После чего создаются два пустых списка с присвоением их значений двум разным переменным. На следующем этапе выполнения тела функции производится присвоение переменной значения длины массива b. Далее запускается цикл *for* с итерируемой переменной i по переменной с раннее присвоенным значением длины массива b. Внутри данного цикла объявляется переменная-счётчик с начальным значением равным 0. После этого конструируется вложенный цикл *for*, пробегающий итерируемой переменной j три раза по нижеследующему телу цикла. В нем самом в первой строке присваивается значение функции библиотеки *random*, которая берет из определенного интервала равномерного распределения случайное число. Верхняя граница определяется как разность элемента списка, вызванного по значению переменной j как индексу и текущему значению счетчика. Следующей после этой процедуры строкой полученное значение добавляется в первый пустой список, который был объявлен ранее. И последней строкой во вложенном списке счетчик обновляет значение на величину значение функции библиотеки *random*. Несмотря на то, что сгенерированные величины получены в справедливых для них верхних граничных точках отрезка равномерного распределения, поэлементная сумма векторов u_1, u_2, u_3 не будет с большой долей вероятности давать вектор b и, следовательно, являться оптимальным решением задачи игрока A_0, что есть препятствие для достижения ситуации равновесия по Нэшу. Для того, чтобы её нивелировать и обеспечить абсолютно достоверное событие в качестве нахождения ситуации равновесия по Нэшу, после вложенного цикла *for* с результатом обработки в форме списка значений u_1, u_2, u_3 объявляем переменную остатка c, которой присваивается значение разности между значением элемента списка b по индексу i и последним значением переменной-счетчика. Далее объявляется цикл, 2 по порядку вложенности, с пробегом итерируемой переменной по целым числам отрезка, нижняя граничная точка которой задается как разность между длиной массива найденного из предыдущего цикла *for* и длиной этого же массива, деленного на величину $i + 1$. Верхняя граничная точка задается как просто длина этого же массива. Элемент этого массива суммируется с переменной остатка, деленного на 3 и добавляется во второй пустой список. После окончания этого цикла объявляются три пустых списка с присвоением их переменным u_1, u_2, u_3. Далее идут три дублирующих себя по структуре цикла начинающиеся с 0, 1, 2 и выполняющихся до числа – длины последнего массива с шагом 3. В деле каждого из 3 циклов *for* производится добавление значения в соответствующий пустой в начале список u_1, u_2, u_3. Тем самым, функция выводит u_1, u_2, u_3, которые в сумме по компонентам дают вектор b.

Для обеспечения численного моделирования верхних граничных точек разработана программная функция *Upper bounds of solutions*, а для генерирования случайных векторов ω_1 и ω_2 функция *Random omg*. Рассмотрим первую функцию, которая является для второй вызываемой в теле функции, или, под-функцией.

Структура функции *Upper bounds of solutions* учитывает специфику применения функции *for* и условных операторов *if*. Функция принимает в качестве аргументов двумерный массив A, списки: u, который является одним из трех векторов, сгенерированных функцией *Random u*, а также ξ.

На первом шаге производится объявление функции. На втором шаге производится импорт библиотеки *math*, предназначенной для математических операций. На третьем шаге объявляется переменная среды, принимающая значение пустого списка. На следу-

Функция 4 Верхние граничные точки ω_1 и ω_2

```
1: function UPPER BOUNDS OF SOLUTIONS(A, u)
2:     import math
3:     A_1 ← [...]
4:     for all i ← A : do
5:         if i = 0 = min then
6:             i ← ∞
7:             del 0
8:         end if
9:     end for
10:    for all i, j ← A, u : do
11:        for all k ← i : do
12:            x ← j/k
13:            A_1 ← x
14:        end for
15:    end for
16:    a ← 0
17:    for all r ← 3 : do
18:        a ← [...]
19:        for all i ← A_1 : do
20:            a[r] ← i
21:        end for
22:    end for
23:    x ← size i, j in A_1
24:    for all i, j ← (size a, a) : do
25:        if i = 0 then
26:            del a[0][x :]
27:        end if
28:        if i ← sizea − 1 then
29:            del a[size a − 1][0; −x]
30:        end if
31:        if i ≠ 0 and i ≠ size a − 1 then
32:            del a[i][: x ∗ i]
33:            del a[i][x :]
34:        end if
35:    end for
36:    x ← max(column) for column in row a
37:    Return x
38: end function
```

ющем шаге задается цикл *for* для всех списков-элементов двумерного массива A. Далее вводится условный оператор для нахождения нулевого элемента как минимального возможного элемента списка-элемента двумерного массива. Если такой элемент существует, то согласно телу условного оператора ему присваивается значение бесконечности. Следующим шагом функции является последовательный цикл *for*, который проходит как по матрице как двумерному массиву A, придавая одной итерируемой переменной i значение элемента-списка, так и по вектору-списку u с итерируемой переменной j и вектору-списку ξ_1 переменной h. Следующим шагом объявляется вложенный цикл *for*, который проходит по спискам-элементам двумерного массива переменной k. После чего во вложенном цикле *for* происходит присвоение переменной x значения отношения j и $k + h$. Тем самым обеспечивается деление суммы правой части неравенства на каждый элемент матрицы A. Результат этой операции добавляется в список A_1. Следующим шагом функции является последовательное, после предыдущей части выполнения, объявление переменной a с присвоением ей пустого списка в качестве значения. После чего запускается цикл, создающий столько массивов пустых списков, каков размер двумерного массива, или, иными словами, сколько имеется в нем строк. Далее задается очередной вложенный цикл *for*, добавляющий элемент списка A_1 в пустой список переменной a, являющейся до этого цикла пустым списком. Поскольку цикл построен так, что добавление элемента списка A_1 происходит целиком в каждый пустой подсписок двумерного массива a, то стоит удалить появившиеся дубликаты. Для этой цели используется технология списочных срезов в массиве a. Сначала определяется число столбцов исходной матрицы A, далее объявляется очередной цикл *for*, который производит преобразование присваивая двум интерируемым переменным i и j значения индекса массива a под которым подразумевается номер строки и самой строки массива. После чего посредством условных операторов производится срез. Результирующая матрица является итерируемым объектом последнего цикла *for*, который проходит по ней и определяет максимальный элемент в строке. Функция возвращает список этих максимальных элементов строк матрицы.

Теперь можно перейти к рассмотрению основной части особенностей выполнения функции *Random omg*.

Функция 5 Генерация случайных векторов ω_1 и ω_2

```
1:  function RANDOM OMG(A, u, ξ)
2:      import random
3:      (u + ξ) ← [...]
4:      for all i ← (size u) : do
5:          for all j, k ← (u, ξ) : do
6:              (u + ξ) ← j + k
7:          end for
8:      end for
9:      Upper ← Upper bounds of solutions(A, u_x i)
10:     Array ← [...]
11:     for all h ← (size Upper) : do
12:         Array ← random(0, Upper[h])
13:     end for
14:     Return Array
15: end function
```

Структура функции *Random omg* заключается в генерации случайных векторов через реализацию методологии генерации списков с помощью циклов *for* и использовании библиотечных функций из *random*. Функция принимает в качестве аргументов двумерный

массив A, списки u, ξ.

Для этого на первом шаге в теле функции производится импорт библиотеки *random* с методами для случайного поиска. На втором шаге объявляется переменная со значением пустого списка. Далее объявляется цикл *for*, проходящий по индексам вектора u. Второй, вложенный, цикл проходит переменными j и k по самими значениям элементов векторов u и xi_1. В теле этого цикла происходит добавление в объявленный ранее массив суммы итерированных переменных $j + k$. Следующим шагом переменной *Upper* присваивается значение функции *Upper bounds of solutions* во второй аргумент которой попадает сумма итерированных переменных. Следующим этапом создается пустой список, в которой нижеследующий цикл *for* добавляет случайное число с отрезком от 0 до значения по списку, выделенному по индексу h, который является итерируемой переменной этого цикла. Функция возвращает список, который содержит случайный вектор.

Теперь необходимо с помощью отдельной функции проверить каждый генерированный случайный вектор ω на соответствие удовлетворению решения систем линейных неравенств, после чего собрать все эти векторы как решения в один массив.

Функция 6 Решение систем неравенств

```
1: function ACCEPTABLE SOLUTION(A, ω, u, ξ)
2:     for all i, l, h ← (A, u, ξ) : do
3:         set u ← [...]
4:         two set u ← [...]
5:         set u_3 ← [...]
6:         a ← 0
7:         b ← 0
8:         for all j, k ← (i, ω) : do
9:             a+ = j * k
10:            b ← l + h
11:        end for
12:        if a ≤ b then set u ← ω
13:        end if
14:    end for
15:    if size A ≠ size set u then
16:        set u ← [...]
17:        Return set u and set u_3
18:    else
19:        set u ← set u[0]
20:        two set u ← set u
21:        set u_3 ← u_3
22:        Return two set u, set u_3
23:    end if
24: end function
```

Структура функции *Acceptable solution* включает использование цикла *for* и условных операторов *if* для вывода результатов вычислений. Функция принимает на вход 4 аргумента: двумерный массив A, списки: ω, u, ξ.

На первом шаге работы в среде объявляется сама функция *Acceptable solution*. После чего конструируется цикл *for* с итерацией переменными i, l, h по трем аргументам функции A, u, ξ. На следующем шаге объявляются две переменные, которым присваиваются начальные значения в форме пустого списка. Они необходимы для записи результатов вычислений полученных u_3 и ω. Шаг 5 заключаются в создании переменной-счетчика с

начальными значениями 0 для дальнейших операций во вложенных циклах. Шаг 7 заключается сумме $l+h$ и ее присвоении объявленной переменной, которая является правой частью неравенства. На следующем шаге объявляется вложенный цикл *for* с итерацией переменными j и k по переменной i, значение которой является списком и взято ранее из A исходным циклом, и аргументу функции ω. В теле этой функции производится присвоение переменной-счетчику значения суммы $j+k$, и тем самым, обеспечивается сумма произведений строки технологической матрицы A и случайного вектора ω в левой части системы неравенств. Следующий, 10 шаг, заключается в использовании условного оператора *if* для проверки истинности неравенства двух переменных согласно неравенству. После чего исходный цикл завершается, запускается конструкция с условным оператором *if* для вывода функции. Если длина двумерного массива A не равна длине массива полученных решений ω, это означает, что не все уравнения удовлетворяют случайному вектору ω, и, в теле этой конструкции, в таком случае, происходит повторное присвоение пустых списков данным переменным с их выводом функции. В другом случае, глобальной переменной u_3 присваивается значение вектора u_3 как элемента списка и происходит обновление переменной ω, в которой остается только вектор, являющийся решением системы линейных неравенств. После чего происходит вывод обоих переменных со связанными с ними списками.

Поскольку функция *Acceptable solution* выполняется с одним набором u_1, u_2, u_3, предстоит построить основную логику процесса генерации массива векторов ω_1 и ω_2 при каждом таком полученном наборе векторов u_1, u_2, u_3.

Функция 7 Основная генерация

```
1:  function MAIN GENERATION(A_1, A_2, ξ_1, ξ_2, n, m)
2:      set ω_1 ← [...]
3:      set ω_2 ← [...]
4:      set u_1 ← [...]
5:      set u_2 ← [...]
6:      set u_3 ← [...]
7:      for all i ← n : do
8:          a = random u(b)
9:          u_1 ← a[0]
10:         u_2 ← a[1]
11:         u_3 ← a[2]
12:         for all j ← m : do
13:             ω_1 ← random omg(A_1, u_1, ξ_1)
14:             ω_2 ← random omg(A_2, u_2, ξ_2)
15:             set u_1 ← u_1
16:             set u_2 ← u_2
17:             set u_3 ← u_3
18:             set ω_1 ← Acceptable solution(A_1, ω_1, u_1, ξ_1)[0]
19:             set ω_2 ← Acceptable solution(A_2, ω_2, u_2, ξ_2)[0]
20:         end for
21:     end for
22:     Return set ω_1, set ω_2, set u_3
23: end function
```

По структуре функция *Main generation* похожа на предыдущие функции, включая конструкции вложенных циклов *for*. Функция принимает на вход двумерные массивы A_1, A_2, векторы наличных ресурсов как списки ξ_1, ξ_2, и также значения параметров n и m как целые числа, определяющие число итераций, генерирующих наборы u_1, u_2, u_3 и ω_1,

ω_2 соответственно.

Первым шагом происходит объявление функции *Main generation* в среде. Вторым шагом происходит объявление переменных с пустыми списками, куда будут сохраняться значения u_1, u_2, u_3, ω_1, ω_2. Третьим конструирование цикла *for*, пробегающий переменной итерации по целочисленному отрезку с верхней границе по значению параметру n - числа генерации набора u_1, u_2, u_3. В теле функции производится присвоение переменной результата выполнения вызванной функции *Random u*. После этого, шагами 5-7, производится присвоение переменным u_1, u_2, u_3 значение переменной с результатами выполнения *Random u* по индексам 1,2,3 соответственно. Следующим, 8 шагом, объявляется вложенный цикл, который пробегает переменной итерации по целочисленному отрезку с верхней границе по значению параметру m - числа генерации набора ω_1, ω_2. По аналогии с предыдущим этапом переменным ω_1 и ω_2 присваивается значение результата вызова функции *Random omg* с аргументами A_1, u_1, ξ_1. В ранее объявленные переменные с пустыми списками добавляются значения u_1, u_2, u_3. В пустые списки для ω_1 и ω_2 добавляются значения результаты вызовов *Acceptable solution*, и поскольку результат функции заключается в двух векторах ω_1 и ω_2, являющихся решениями соответствующих им системам линейных неравенств, вызов производится по индексам: 0 для ω_1 и 1 для ω_2. Также проводится добавление соответствующего набору ω_1 и ω_2 вектора u_3 в два отдельных дублирующих отдельных, пустых в начальном состоянии, списка для синхронизации этой тройки значений. В результате функция выводит 4 списка, состоящих из троек ω_1 ω_2, u_3.

Поскольку выход предыдущей функции является массивом векторов, удовлетворяющих всем полученным наборам u_1, u_2, u_3, для корректного решения стоит разделить массив еще одной системой списков с целью получения двумерного массива. Для этого объявим функцию конвертации массивов.

Структура данной функции включает в себя многократное использование вложенных циклов *for* и условных операторов *if*. В качестве аргументов функция принимает выход предыдущей функции *Main generation*, то есть, 4 списка, состоящих из троек ω_1 ω_2, u_3, также на вход принимается два числа – значения параметров n и m как целые числа, определяющие число итераций, генерирующих наборы u_1, u_2, u_3 и ω_1, ω_2 соответственно.

В первом шаге происходит объявление функции *Converting arrays*. Последующими шагами объявляются переменный с присвоением 4 списков, содержащих нули и все имеющие длину n. После этого объявляются две переменные, с присвоенными им пустыми списками. После чего производится первая конструкция цикла *for*, который пробегает переменной итерации i по числовому целочисленному отрезку длины n. После этого шага конструируется второй вложенный цикл, который осуществляет итерацию переменной j по целочисленному отрезку от числа $m * i$ до числа $m * (i + 1)$. Далее объявляется условный оператор *if*, проверяющий условие того, что поступающий в качестве аргумента элемент массива ω, вызываемый по индексу j не является пустым списком, который может получится в случае, если случайные вектор ω не удовлетворяет своей системе линейных неравенств. Далее вводится условный оператор, который проверяет, является ли тип объекта элемента нового списка, с начальным состоянием нулевого списка длины n, сам списком. Если является, это означает что в двумерном массиве уже есть пустой список, и следует добавить следующий элемент со значением из списка ω по индексу j в проверяемый список на позицию i. Аналогичная операция проводится для массива u_3. Если условие, прописанное в условном операторе *if* не является истинным, функция обращается к блоку под оператором *else*, где первой инструкцией создает пустой список на указанной позиции i. Иными словами, проводится замена элемента списка со значением 0 на значение пустого списка. После этого добавляет очередной элемент из списка ω по индексу j в проверяемый список по индексу j. Поскольку замена элемента списка по инструкции метода списка не удаляет элемент, он переходит на позицию $i + 1$, поэтому стоит удалить лишний элемент

Функция 8 Конвертация массивов

```
1: function CONVERTING ARRAYS(set ω1, set ω2, 1 set u3, 2 set u3, n, m)
2:     Arrays 1, 2, 3, 4 ← [0] * n
3:     Arrays 5, 6 ← [...]
4:     for all i ← n : do
5:         for all j ← (set ω1 * i, set ω1 * (i + 1) : do
6:             if set ω1[j] ≠ [...] then
7:                 if set ω1[i] = list then
8:                     Array1 : i ← set ω1[j]
9:                     Array3 ← 1 set u3[j]
10:                else
11:                    Array1 : i ← set ω1[j]
12:                    Array1 : del i + 1
13:                    Array3 : i ← 1 set u3[j]
14:                    Array3 : del i + 1
15:                end if
16:            end if
17:        end for
18:    end for
19:    for all i ← n : do
20:        for all j ← (setω1 * i, set ω1 * (i + 1) : do
21:            if set ω1[j] ≠ [...] then
22:                Array 5 ← set ω1[j]
23:            end if
24:        end for
25:    end for
26:    if Size Array 5 = 0 then
27:        Return Arrays 1, 2, 3, 4
28:    end if
29:    Similar loops for arrays 3, 4, 6
30: end function
```

с помощью специальной процедуры, что и проводится следующим шагом. Аналогичные шаги в теле этого условия *else* проводятся и для массива u_3. Поскольку в данном случае был принят подход, согласно которому функция вызывается один раз, она должна учитывать двукратное использование скрипта для обработки двумерных массивов, содержащих ω_1 и ω_2. Тем самым, начиная с объявления первого цикла *for* скрипт повторяется ниже для массива ω_2. Следующие циклы являются проверочными и нужны для того, чтобы функция выполнила свое предназначение и не потеряла часть данных. Как и до этого создается цикл *for*, который пробегает переменной итерации i по числовому целочисленному отрезку длины n. После этого шага конструируется второй вложенный цикл, который осуществляет итерацию переменной j по целочисленному отрезку от числа $m * i$ до числа $m * (i + 1)$. Далее объявляется условный оператор *if*, проверяющий условие того, что поступающий в качестве аргумента элемент массива ω, вызываемый по индексу j не является пустым списком. В случае, если это так, то это значение добавляется в один из двух, объявленных в начале тела функции, список. Аналогичная операция проводится для дублирующего скрипта ω_2. Для следующей инструкции проверки создаются 4 пустых списка, по каждому из вывода функции *Main generation*. Создается новый цикл *for* с переменной итерации, которая пробегает по целочисленному отрезку от 0 до длины отрезка, полученном в основном цикле *for*. Следующим шагом после этого конструируется вложенный условный оператор *if*. Проверяется условие неравенства элементов списков, полученных на предыдущих этапах 0 или пустому списку как для массива, содержащего ω_1, так и для массива с ω_2. В случае, если условия истинны, то есть, пустые списки или 0 не найдены параллельно ни в массиве содержащий ω_1, ни в массиве с ω_2, тогда по индексу со значением переменной итерации из 4 массивов, содержащих ω_1, ω_2 и два с векторами u_3, производится добавление в соответствующий для них созданный, пустой в начальном состоянии, список. Далее объявялется переменная-счётчик с нулевым начальным состоянием. Следующий цикл *for* объявляется для проверки нулей в уже проверенном до этого и новом массиве. Делается это постольку, поскольку предыдущий цикл обрабатывал два списка параллельно. Если элемент списка равен 0, то значение переменной-счетчика увеличивается на 1. Аналогичный цикл *for* строится и для ω_2. Следующей проверяющей структурой является цикл *for*, до которого задается переменная-счетчик с нулевым начальным значением. Цикл пробегает посредством переменной итерации по целочисленному отрезку от 0 до длины массива, содержащего ω_1 и полученного при проверке условным оператором двух списков параллельно. В теле цикла обновляется переменная счетчик, увеличивающаяся по длине элемента того же массива. Аналогичная операция и цикл создается для массива, содержащего ω_2. Последней конструкцией вводится условный оператор, проверяющий равенство 0 двух, заданных в самом начале тела с 0 значениями переменных. А также два условия, проверяющих, что переменные-счетчики, увеличивающиеся на 1 при нахождении нулевого элемента нового списка, остались равны 0. В случае истинны, происходит вывод функции, состоящий из 4 двумерных массивов, каждый элемент-список ω_1 или ω_2 в котором, соответствует только одному u_3 из двух других равных по элементам и длине массивов его содержащих.

Таким образом, завершается условный первый этап выполнения программы. Он заключается в генерации случайных векторов u_1, u_2, u_3, которые дают в сумме по компонентам вектор b с использованием для этого функции *Random u*. После чего векторы u_1, u_2 как два списка определенной длины поступают, вместе с технологической матрицей A и вектора наличных ресурсов ξ на вход следующей функции *Random omg*. В ходе её выполнения вызывается функция *Upper bounds of solutions*, которая на вход принимает те же аргументы, что и над-функция *Random omg*: u, A, ξ. На выход она отдает список верхних граничных точек отрезка равномерного распределения, из которого происходит генерация случайных чисел – компонентов векторов ω_1 и ω_1 в рамках последующего выполнения функции

Random omg. Путем двукратного применения функции в среде *main* в очередном генераторе списков *for* происходит создание массива векторов ω_1 и ω_1, являющихся решениями системы линейных неравенств.

Вторая часть алгоритма программы состоит из одной функции, вычисляющей оптимальное решение v^* задачи (26) - (27). Её выполнение основано на модуле *cvxopt*, функции которого используются для формулировки оптимизационной проблемы. Однако сама библиотека не учитывает возможности применения симплекс-метода для решения сформулированной оптимизационной задачи. Для этого применяется решатель *glpk*, который предназначен для решения задач крупномасштабного линейного программирования, смешанного целочисленного программирования, а также других проблем связанных с математической оптимизацией.

Функция 9 Решение оптимизационных проблем

```
1: function SOLVING OPTIMIZATION PROBLEMS(D, γ1, γ2, c4, set ω1, set ω2, set u3)
2:     Array u3 ← [...]
3:     for all i, j, k ← (set ω1, set ω2, set u3) : do
4:         for all h1 ← i do
5:             for all h2, h3 ← j, k do
6:                 vector ω1 ← h1
7:                 vector ω2 ← h2
8:                 vector u3 ← h3
9:                 Array u3 ← vector u3
10:                ω1, ω2, u3, ξ1, ξ2, γ1, γ2 ← variable(size)
11:                c4 ← matrix(c4)
12:                D ← matrix(D)
13:                ineq1 ← D * v ≤ ω1 + ω2 + γ1
14:                ineq2 ← D * v ≤ u3 + γ2
15:                ineq3 ← u3 ≥ 0
16:                ineq5 ← ω1 ≥ 0
17:                ineq6 ← ω2 ≥ 0
18:                ineq6 ← v ≥ 0
19:                ineq7 ← γ1 ≥ 0
20:                ineq8 ← γ2 ≥ 0
21:                lp ← c4 * v = max s.t. ineq1, ineq2, ineq3, ineq4, ineq5, ineq6, ineq7, ineq8
22:            end for
23:        end for
24:    end for
25:    Return v*, l4
26: end function
```

Структура функции *Solving optimization problems* заключается в использовании, помимо библиотечных функций, также циклов *for* для получения массивов ответов. На вход функция принимает вывод функции *Converting arrays*, то есть 4 двумерных массива, каждый элемент-список ω_1 или ω_2 в котором, соответствует только одному u_3 из двух других равных по элементам и длине массивов его содержащих. Помимо этого на вход поступает 2 вектора наличных ресурсов γ_1, γ_2.

Для начала работы функции, как и ранее, необходимо ее вызвать. После чего вторым шагом подключаются необходимые функции модуля *cvxopt*. Следующим, третьим шагом, конструируется цикл *for*, который двумя переменными итерации пробегает по двум двумерным массивам, содержащих решения системы линейных неравенств ω_1 и ω_2. Далее,

необходимо выбрать при фиксации какого из векторов ω будет происходить перебор значений двумерного списка второго массива ω. Был выбран вариант с фиксацией каждой ω_1 для перебора по данным ему ω_2 из второго массива. Для этого был создан цикл *for*, который пробегает переменной итерации по значениям из этого массива, являющегося двумерным списком, чтобы в следующем вложенном цикле *for* была возможность пробежаться двумя переменными итерации по двум спискам: одном содержащем, ω_2 и втором, включающего в себя u_3. Таким образом обеспечивается распаковка массивов, содержащих тройки (ω_1, ω_2, u_3).В последнем вложенном цикле *for* производится последовательное присвоение переменным значений этой тройки (ω_1, ω_2, u_3). Также, в пустой массив добавляется u_3 для возможности проверки и упрощения вывода ответа. Далее задается блок 4 неизвестных переменных с присвоением результата библиотечной функции, которая создает объект типа *variable* с длиной, которая является длиной вектора наличных ресурсов γ и названием. Аналогично далее задается блок с известными переменными γ_1, γ_2. Следующим шагом объявляется переменная c_4, которой присваивается значение библиотечной функции *matrix*, которая создает объект с таким же названием из объекта списка c_4, являющегося вектором дохода игрока C. Поскольку особенностью решателя является то, что им решается только задача минимизации, вектор дохода в функции становится отрицательным. Следующим шагом объявляются две переменные D и M, которым присваивается значение результата работы библиотечной функции *matrix*, которая обрабатывает два двумерных массива данных из вызванной функции $Task\ data$. Далее идет блок с системами ограничений. Помимо естественных ограничений неотрицательности переменных, для данной задачи отдельно также присваивается заданным переменным ограничения в виде двух систем линейных неравенств: $D * v <= \omega_1 + \omega_2 + \gamma_1$ и $M * v <= u_3 + \gamma_2$. После чего с помощью функции *op* конструируется оптимизационная проблема согласно математической модели, представленной в первой главе и с учетом объявленных ранее переменных. После этого к данному объекту модуля применяется метод модуля *solve*, который с помощью решателя *glpk* в аргументе получает решение задачи. Особенностью является то, что не каждое решение задачи является оптимальным. После этого, результаты вычислений сохраняются в 5 массивов: для ω_1, ω_2, $u3$, v и значения целевой функции игрока C. На этом исходный цикл *for* завершается, и далее объявляется второй цикл, который последовательно по списку значений целевой функции игрока C умножает каждый элемент на константу -1 для избавления от отрицательности. Функция выводит 5 двумерных массивов, содержащих ω_1, ω_2, u_3, v и значения целевой функции игрока C.

Третья часть алгоритма программы заключается в проверке найденных оптимальных решений v^* в массиве $V*$, нахождение оптимального решения u_1^*, u_2^*, u_3^* для задачи игрока A_0, получение также оптимального решения ω_1^*, ω_2^* в рамках совместной решения задачи для игроков B_1 и B_2, и, наконец, определение единственного оптимального вектора v^*. Также определяется нахождение значение целевых функций всех игроков A_0, B_1 и B_2, C. Для дележа полезности игроков B_1 и B_2 используется вектор Шепли, определенный в Главе I.

Для целей проверки оптимальности решения проверим вывод предыдущей функции *Solving optimization problems* на предмет нахождения в массиве оптимальных векторов неоптимального вектора, который может появиться в следствии работы функции и выводу сертификата, сообщающего, что решение не оптимально. Поскольку при этом в двумерном массиве, содержащий как задумывалось оптимальные векторы v^*, появляются отрицательные векторы v, следует проверить циклом предмет нахождения этих векторов в массиве. Для этого конструируется функция *Checking restrictions*, принимающая на входе аргумент – двумерный массив векторов v.

Данная функция сконструирована по принципу двойной проверки. На первом шаге объявляется сама функция *Checking restrictions*. На следующем шаге конструируется цикл

Функция 10 Проверяющие циклы

```
1: function Checking restrictions(v*)
2:     for all i ← h do
3:         k = 0
4:         for all j ← v* do
5:             for all k ← v* do
6:                 if j < 0 then
7:                     k+ = 1
8:                 end if
9:             end for
10:        end for
11:        for all j ← v* do
12:            for all k ← v* do
13:                if j < 0 then
14:                    del v_*
15:                end if
16:            end for
17:        end for
18:    end for
19:    Return k
20: end function
```

for, который пробегает переменной итерации по целочисленному отрезки с длиной, заданной параметром. После этого в следующем шаге задается переменная-счетчик с нулевым начальным значением. После чего объявляется вложенный цикл *for*, который итерирует входной двумерный массив, состоящий из списков v. Следующий вложенный цикл *for* обеспечивает допуск к каждому элементу списка. В самом деле этого цикла происходит проверка условия на истинность с помощью условного оператора *if* на предмет того, что хотя бы один элемент списка, то есть, число – компонента вектора v – меньше 0, и если это так, то после этого выполняется действие увеличение переменной-счетчика на 1. Стоит отметить, что при запуске программы данная проблема возникала редко и счетчик остается равен 0. Однако, если все-таки он не равен нулю, то с помощью вложенных циклов обеспечивается допуск к этому списку-вектору v и производится удаление этого списка в входном двумерном массиве и массивах, содержащих его значение целевой функции игрока C, наборы ω_1, ω_2 и u_1, u_2, u_3, как глобальных переменных, которые приводят к получению такого вектора. После выполнения этого цикла проводится повторная проверка на предмет нахождения не оптимальных векторов. Функция выводит всем массивы, над которыми проводились операции.

После того как была проверена оптимальность полученного массива v^* получить значения оптимальных векторов ω_1^*, ω_2^* и, также, набора оптимальных u_1^*, u_2^*, u_3^*. При которых достигается максимумы трех целевых функций игроков A_0, B_1 и B_2, C. Для этого подставим значение v^*, найденное для каждой тройки ω_1, ω_2, u_3 по соответствию и получим массив значений, который точно содержит ω_1^*, ω_2^*. Также, аналогично, подставим в целевую функцию игрока A_0 значение v^*, найденное для каждой тройки u_1, u_2, u_3 и найдем векторы u_1^*, u_2^*, u_3^*, при которых достигается максимум целевой функции.

Структура данной функции *Target functions* включает в себя работу со списочными объектами и индексирование элементов двумерных массивов. Функция принимает на вход в качестве аргументов двумерный массив, состоящий из элементов v^*, а также 4 вектора доходов игроков A_0, B_1 и B_2, C: c_1, c_2, c_3, c_4.

Функция 11 Целевые функции

```
1: function TARGET FUNCTIONS(v, l_4, set ω_1, set ω_2, set u_1, set u_2, set u_3, c_1, c_2, c_3, c_4)
2:     import numpy
3:     c_2 ← array(c_2)
4:     c_3 ← array(c_3)
5:     for all i ← v do
6:         i ← array(i)
7:         x = dot(c_2, i) + dot(c_3, i)
8:     end for
9:     optimal ω_1 and ω_2 ← max(x)
10:    c_1 ← array(c_1)
11:    for all i ← v do
12:        i ← array(i)
13:        y = dot(c_1, i)
14:    end for
15:    optimal u ← max(y)
16:    ω_1^* ← max(optimal ω_1)[optimal u]
17:    ω_2^* ← max(optimal ω_2)[optimal u]
18:    v^* ← v[ω_1^*, ω_2^*]
19:    l_1 ← v^* * c_1
20:    l_2 + l_3 ← v^* * c_2 + v^* * c_3
21:    l_4 ← v^* * c_4
22:    Return optimal u, ω_1^*, ω_2^*, l_1, l_2 + l_3, l_4
23: end function
```

Как и ранее объявляется функция *Target functions*. Следующим шагом импортируется модуль *numpy* для вызова функции внутреннего произведения. Для подсчета максимального значения целевой функции игрока A_0 переменной присваивается значение функции с аргументом вектора дохода c_1 для создания объекта из библиотеки *numpy* с целью возможности скалярного произведения двух однородных объектов. После чего выполняется переменная с присвоенным ей значением пустого списка. Для каждого элемента в объявленном цикле *for*, итерируемая переменная по двумерному массиву, содержащему v^* сохраняется в переменной, также, как и сохраняется результат скалярного произведения этой переменной, являющейся носителем одного списка v^* на вектор-список c_1. После чего результат вычисления сохраняется в массив, который после завершения цикла содержит для всевозможных значений переменных значения целевой функции. Далее из этого массива берется максимальное значение функции и сохраняется в переменной. Следующее значение целевой функции игрока B_1 и B_2 вычисляется со знанием индекса максимального значения целевой функции игрока A_0, по которому и вызывается вектор v придающий максимальное значение игроку A_0. Также известен набор u_1, u_2, u_3, который выбрал игрок A_0. Поэтому игроки B_1 и B_2 по индексам ω_1 и ω_2 по набору u_1^*, u_2^*, u_3^* вызывают, равное числу комбинаций этих двух векторов, вектор v, каждый раз умножая его в цикле *for* на векторы доходов c_2 и c_3 с последующей их суммой. После этого, по номеру индекса находится значение v^*, которое умножается на каждый вектор из целевых функций и тем самым становится известно конечное значение целевых функций игроков A_0, B_1 и B_2, C и векторы u_1^*, u_2^*, u_3^*, ω_1^*, ω_2^*, v^*. Это и является выводом функции *Target functions*.

Однако остается проблема получения значений функций полезности игроков B_1 и B_2, которые действовали согласно модели как один игрок. Теперь их совокупный выигрыш необходимо разделить, для чего будет применен вектор Шепли из (50).

Функция 12 Вектор Шепли коалиции S

1: **function** SHAPLEY VALUE OF COALITION S($c_1, c_2, c_3, c_4, \xi_1, \xi_2, \gamma_1, A_1, A_1, D, b$)
2: $a \leftarrow Main\ generation(A_1, A_2, xi_1, xi_2, u_1 = 0)$
3: $b \leftarrow Main\ generation(A_1, A_2, xi_1, xi_2, u_2 = 0)$
4: $y \leftarrow Converting\ arrays(a, b)$
5: $x \leftarrow Solving\ optimization\ problems(D, \gamma_1, \gamma_2, c_4, set\ \omega_1, set\ \omega_2, set\ u_3)$
6: $h \leftarrow Checking\ restrictions(v)$
7: $Targetfunction(v, l_4, set\ \omega_1, set\ \omega_2, set\ u_1, set\ u_2, set\ u_3, c_1, c_2, c_3, c_4)$
8: $Sheply_1 = l_2 + (((l_2 + l_3) - l_2 - l_3)/2)$
9: $Sheply_2 = l_3 + (((l_2 + l_3) - l_2 - l_3)/2)$
10: $Return\ Sheply_1\ Sheply_2$
11: $Similarly\ 2\ other\ ways$
12: **end function**

В главе I были рассмотрены три вида одноэлементных характеристических функций, были получены 3 способа вычисления минимальной гарантированной полезности для игроков в наихудшем случае сначала для игрока B_1, а потом и для игрока B_2. Для вычисления 1 способом модифицируется функция *Solving optimization problems*, которая вызывается как под-функция *Shapley value of coalition S*. В ней для расчета минимальной гарантированной полезности игрока B_1 в качестве переменной u_1 для итерации по ω_1 и ω_2 приписывается 0. Аналогичная процедура проводится для игрока B_2 и вектора u_2. Во всем остальном логика функции остается неизменной. После этого вызывается функция *Checking restrictions* и проверяются массивы полученных в выводе предыдущей функции. Также вызывается модифицированная функция *Target functions* для получения решений оптимизационных задач, в которой вместо подсчета суммы скалярных произведений векторов дохода игроков B_1 и B_2 на вектор v вычисляются различные целевые функции – для B_1 и для B_2. Для вычисления вторым способом также модифицируется функция *Solving optimization problems*, которая вызывается как под-функция *Shapley value of coalition S* с фиксированными для B_1 и B_2 векторами $u_1 = 0$, $u_2 = 0$. После этого вызывается функция *Checking restrictions* и проверяются массивы полученных в выводе предыдущей функции. Также вызывается модифицированная функция *Target functions* с разделением целевых функций игроков B_1 и B_2 для получения решений оптимизационных задач. И, аналогично, рассматривается третий способ расчета одноэлементной характеристической функции B_1 и B_2 при $u_1 = 0$, $u_2 = 0$, $u_3 = 0$. В данном случае фиксируется весь вектор $u = 0$ и три переменные в теле функции *Solving optimization problems*, которым они соответствуют. После вычислений вызывается функция *Checking restrictions* и проверяются массивы полученных в выводе предыдущей функции. Также вызывается модифицированная функция *Target functions* с разделением целевых функций игроков B_1 и B_2 для получения решений оптимизационных задач. После чего согласно формулам (50) вычисляются три вектора Шепли, которые являются выводами функции.

В Главе I была определена кооперативная игра (N, v), алгоритм для вычисления в рамках программной функции значений характеристических функций и вектора Шепли по логике не значительно от логики вычисления равновесия по Нэшу и вектора Шепли для коалиции S. Иллюстрация псевдокода для двух программных функций кооперативной игры похожи на структуру функции *Shapley value of coalition S*, основанной на модификации данных, присланных на вход уже заданным до этого функциям.

Структура функции *Cooperative game* заключается в использовании под-функций и циклов *for* для получения значений характеристических функций коалиций T. Функция *Cooperative game* принимает в качестве аргументов переменную со списком-результатом

работы функции *Task data*, элементами которого являются все известные переменные задачи.Рассмотрим подсчет характеристическую функцию вида (47). Первым шагом происходит объявление данной функции *Cooperative game*. Вторым шагом происходит вызов *Main generation* с модификацией в виде добавления в двумерный массив для ω_1 и ω_2 пустых списков, и идет присвоение переменной результатов работы этой функции. После этого вызывается функция *Converting arrays* и ее результат возвращается в приписанном переменной значении. После этого вызывается функция *Solving optimization problems* с модификацией: фиксацей по 0 значений всех u_i, $i = 1, 2, 3$. После чего таким же образом с присвоением переменной вызывается функция *Checking restrictions*. Далее вызывается функция *Target functions*, модификация которой заключается только в том, что оптимальный вектор v^* определяется сразу, и его значение участвует в вычислении целевых функций A_0, B_1, B_2. Характерные модификация проводятся для каждой коалиции (47) - (54). Функция выводит значения 8 характеристических функций.

Но необходимо также определить оптимальный дележ для всей структуры в кооперативной игре (N, v). Для этого определим функцию *Shapley value of cooperation*. Структура этой функции заключается в использовании полученных результатов вычисления функции *Cooperative game* как аргументов функции *Shapley value of cooperation* с целью вычисления вектора Шепли. Первым шагом объявляется сама функция *Shapley value of cooperation*, после чего конструируются 4 переменные, которые согласно формулам (55) - (59) присваивают значения. После чего осуществляется вывод функции со значениями этих 4 переменных.

Таким образом, рассмотрены аспекты алгоритма программы, которая приведена в приложении, реализующая теоретико-игровую модель с целью численного нахождения ситуации равновесия по Нэшу и всех переменных, которые к этой ситуации приводят, и также реализующая численное нахождение значений характеристических функций и вычисление принципа оптимальности для дележа – вектор Шепли.

Зависимость функций в ходе поиска ситуации равновесия по Нэшу можно представить в виде следующей блок-схемы:

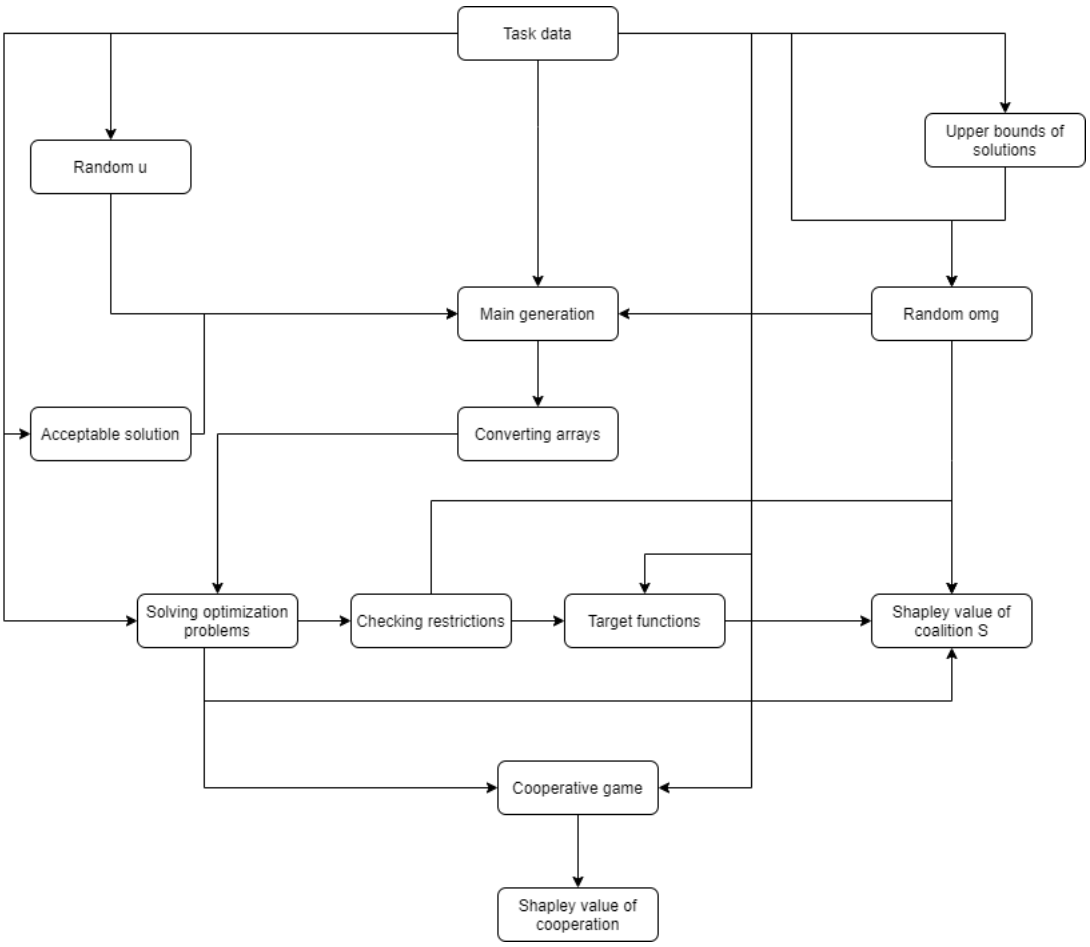

Рис.2

2.2. Результаты эксперимента с нахождением ситуации равновесия по Нэшу

Возможность численного нахождения ситуации равновесия по Нэшу реализуется благодаря описанному в первой главе математическому аппарату, составленному алгоритму по модифицированному методу Монте-Карло и программной реализации этого алгоритма в среде разработки. Основными двумя параметрами, определяющими число выполняемых операций и скорость работы алгоритма являются параметры, определяющие число генерации случайных векторов u_1, u_2, u_3 и, для каждого такого набора, число генерации случайных векторов ω_1 и ω_2. В результате алгоритмическая сложность алгоритма асимптотически оценивается как $\mathcal{O}(mn)$, где m – число генерации случайных векторов u_1, u_2, u_3, а n – число генерации случайных векторов ω_1 и ω_2.

Приведем пример с использованием построенной программной реализации согласно оформленному в параграфе 2.1. алгоритму.

На рынке производства глиняных изделий для домашнего хозяйства есть 4 игрока, один из них, поставщик сырья A_0, обладает некоторым количеством 8 типов глины различных характеристик, измеряемой в килограммах. Игроки B_1 и B_2 обладают оборудованием, необходимым для изготовки заготовок, но печи для обжига заготовок у них нет. Поскольку игроки B_1 и B_2 работают в одной мастерской, они решили объединить свои усилия. Игрок C имеет оборудование как для производства заготовок глиняных изделий, так и печь для обжига. Тем самым, он может произвести 7 типов глиняных изделий, измеряемых также в килограммах. У игроков B_1, B_2, C остался некоторый запас глины 8 типов от прошлых заказов. Полезность является доходом в тысячах рублей. Требуется рассмот-

реть равновесную ситуацию на рынке и определить какие производственные кооперации и почему могут быть созданы.

Приведем известные данные для решения данной задачи. Для игрока A_0 дан вектор b и вектор дохода c_1.

$$\begin{gathered} b = \{120; 460; 370; 380; 830; 910; 120; 660\} \\ c_1 = \{66; 35; 87; 33; 91; 97; 45\} \end{gathered} \tag{71}$$

Игроку B_1 дана технологическая матрица A_1, вектор наличных ресурсов ξ_1 и вектор дохода c_2

$$\begin{gathered} A_1 = \begin{pmatrix} 1 & 3 & 9 & 8 & 2 & 8 & 7 \\ 2 & 8 & 4 & 3 & 6 & 9 & 2 \\ 9 & 2 & 3 & 9 & 4 & 7 & 5 \\ 1 & 8 & 4 & 4 & 4 & 5 & 2 \\ 8 & 9 & 6 & 3 & 2 & 2 & 4 \\ 4 & 4 & 5 & 6 & 4 & 2 & 3 \\ 5 & 6 & 5 & 5 & 2 & 1 & 5 \\ 3 & 1 & 4 & 1 & 4 & 3 & 7 \end{pmatrix} \\ \xi_1 = \{15; 25; 1; 18; 11; 24; 12; 11\} \\ c_2 = \{76; 45; 27; 63; 41; 77; 25\} \end{gathered} \tag{72}$$

Игроку B_2 дана технологическая матрица A_2, вектор наличных ресурсов ξ_2 и вектор дохода c_3

$$\begin{gathered} A_2 = \begin{pmatrix} 1 & 3 & 9 & 8 & 2 & 8 & 4 \\ 2 & 2 & 4 & 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 5 & 4 & 7 & 5 \\ 8 & 2 & 4 & 1 & 2 & 2 & 5 \\ 8 & 1 & 6 & 1 & 2 & 2 & 4 \\ 2 & 4 & 5 & 6 & 4 & 1 & 5 \\ 7 & 1 & 5 & 4 & 2 & 1 & 5 \\ 4 & 2 & 4 & 7 & 5 & 2 & 1 \end{pmatrix} \\ \xi_2 = \{25; 24; 15; 5; 21; 4; 2; 1\} \\ c_3 = \{46; 25; 22; 69; 48; 72; 24\} \end{gathered} \tag{73}$$

Игроку C дана технологическая матрица D по заготовкам, вектор наличных заготовок γ_1, технологическая матрица M по типу глины, вектор наличных остатков глины γ_2 и

вектор дохода c_4.

$$D = \begin{pmatrix} 1 & 3 & 9 & 8 & 2 & 8 & 5 \\ 5 & 5 & 4 & 1 & 5 & 2 & 2 \\ 5 & 2 & 3 & 9 & 4 & 7 & 5 \\ 2 & 8 & 4 & 1 & 2 & 5 & 2 \\ 8 & 2 & 2 & 3 & 2 & 2 & 2 \\ 4 & 4 & 5 & 6 & 4 & 5 & 3 \\ 5 & 6 & 4 & 2 & 2 & 2 & 5 \end{pmatrix}$$

$$\gamma_1 = \{80; 85; 155; 20; 90; 49; 50\}$$

$$M = \begin{pmatrix} 9 & 3 & 9 & 8 & 2 & 8 & 9 \\ 5 & 5 & 4 & 5 & 5 & 2 & 8 \\ 5 & 2 & 3 & 9 & 4 & 7 & 7 \\ 9 & 4 & 4 & 9 & 9 & 9 & 5 \\ 7 & 7 & 2 & 5 & 6 & 7 & 2 \\ 8 & 2 & 5 & 6 & 8 & 5 & 3 \\ 5 & 6 & 4 & 2 & 2 & 2 & 5 \\ 3 & 9 & 7 & 8 & 7 & 3 & 9 \end{pmatrix} \tag{74}$$

$$\gamma_2 = \{86; 22; 10; 50; 71; 48; 19; 19\}$$

$$c_4 = \{96; 95; 57; 73; 52; 87; 76\}$$

Решим данную задачу с помощью программной реализации. Прогоны численной модели по данному примеру осуществлялись на m = 500, 1000, 2000, 3000, 4000, 5000 и, соответственно n = 500, 1000, 2000, 3000, 4000, 5000. Были получены разные результаты прогона модели, для решения задачи были использованы параметры m = 1000, n = 10000, то есть каждого из 1000 наборов случайных векторов u_1, u_2, u_3 были сгенерированы 10000 наборов случайных векторов ω_1, ω_2. Число наборов u_1, u_2, u_3, для которых были найдены решения систем линейных неравенств составило 29, то есть $2,9\%$. Число самих решений составило 82, $0,82\%$ от набора. Для увеличения скорости вычислений использовались серверы компании *Amazon*. Представим результаты, получившиеся в результате выполнения программы, по алгоритму, описанному в первом параграфе.

$$u_1 = \{25.316; 127.9; 185.775; 43.946; 585.494; 794.669; 51.494; 307.931\}$$
$$u_2 = \{49.785; 213.938; 96.779; 81.333; 208.777; 49.241; 63.5; 91.61\}$$
$$u_3 = \{44.898; 118.161; 87.445; 254.72; 35.727; 66.088; 5.005; 260.458\}$$
$$\omega_1 = \{3.132; 0.345; 0.064; 0.081; 4.002; 1.104; 0.409\}$$
$$\omega_2 = \{3.515; 2.267; 0.148; 0.967; 2.052; 0.889; 2.001\} \tag{75}$$
$$v = \{1.686; 0; 0; 5.314; 0; 2.472; 0\}$$
$$K_1 = 516.488$$
$$v(S) = v\{B_1; B_2\} = 1275.617$$
$$K_4 = 745$$

Для дележа полезности используем составленные в Главе I 3 вида одноэлементных характеристических функций, после чего применим формулы (50) для каждого из трех способов подсчета минимальной гарантированной полезности. Результат вычисления вектора Шепли для примера 1 способом, при котором $u_1 = 0$, $u_2 \geq 0$, $u_3 \geq 0$ для B_1 и $u_1 \geq 0$, $u_2 = 0$, $u_3 \geq 0$ для B_2, выглядит следующим образом:

$$\begin{gathered}\widetilde{Sh_{B_1}} = 648.615 \\ \widetilde{Sh_{B_2}} = 606.609 \\ \widetilde{K_1} = 515.151 \\ \widetilde{K_4} = 748\end{gathered} \tag{76}$$

При этом видно, что обеспечивается $\widetilde{Sh_{B_1}} + \widetilde{Sh_{B_2}} = v(S)$. Рассмотрим результат вычисления вектора Шепли для примера 2 способом, при котором $u_1 = 0$, $u_2 = 0$, $u_3 \geq 0$

$$\begin{gathered}\widehat{Sh_{B_1}} = 646.233 \\ \widehat{Sh_{B_2}} = 608.991 \\ \widehat{K_1} = 508.151 \\ \widehat{K_4} = 745\end{gathered} \tag{77}$$

Видно, обеспечивается $\widehat{Sh_{B_1}} + \widehat{Sh_{B_2}} = v(S)$. Рассмотрим третий способ, при котором $u_i = 0$, $i = 1, 2, 3$.

$$\begin{gathered}\overline{Sh_{B_1}} = 664.279 \\ \overline{Sh_{B_2}} = 590.945 \\ \overline{K_1} = 146.333 \\ \overline{K_4} = 286\end{gathered} \tag{78}$$

Видно, что обеспечивается $\overline{Sh_{B_1}} + \overline{Sh_{B_2}} = v(S)$. Видно также, что $K_1 \geq \widetilde{K_1} \geq \widehat{K_1} \geq \overline{K_1}$ и $K_4 \geq \widetilde{K_4} \geq \widehat{K_4} \geq \overline{K_4}$, что в общем виде соответствует (45).

Интересные результаты получены относительно компонентов векторов Шепли, оказывается, что $\overline{Sh_{B_1}} \geq \widetilde{Sh_{B_1}} \geq \widehat{Sh_{B_1}}$. Это означает, что игроку выгодно рассматривать минимальную гарантированную полезности с точки зрения наихудшего возможного варианта, при котором $u_i = 0$, $i = 1, 2, 3$, и подобная позиция дает ему большую полезность. То есть, он будет исходить из того, что как игроки в мастерской B_1 и B_2, так и игрок со всем оборудованием C не получат наборы глины 8 типов.

Подобный вывод справедлив и для игрока B_2. Получается, что $\widetilde{Sh_{B_2}} \geq \widehat{Sh_{B_2}} \geq \overline{Sh_{B_2}}$ и игроку B_2 при дележе исходить из того, что $u_1 = 0$, $u_2 = 0$, $u_3 \geq 0$, или, иными словами из того, что игроки B_1 и B_2 в мастерской при худшем случае не получат никакой набор глины разных типов, но игрок, обладающий всем оборудованием для производства может получить некоторый набор из килограммов глины 8 типов.

Таким образом, численно найдена ситуация равновесия по Нэшу, при котором игрок A_0 получит 526.488 тыс. рублей, игрок C 765 тыс. рублей, в доходы игроков B_1 и B_2 рассмотрены при 3 различных способах подсчета минимальной гарантированной полезности, а также найдены векторы ресурсов и вектор продукции.

2.3. Результаты эксперимента с кооперативной игрой

Рассмотрим проблематику нахождения значений характеристических функций кооперативной игры (N, v), заданной на ромбовидной иерархической структуре игры Г. Для вычисления будем пользоваться программными функциями *Cooperative game* и *Shapley value of cooperation* алгоритма, которые были описаны в параграфе 2.1. Рассмотрим одно-

элементную коалицию T, содержащую только игрока C согласно (47). Значение характеристической функции в этом случае будет равно:

$$v(T) = v(\{C\}) = 286 \tag{79}$$

Теперь исследуем класс двухэлементных характеристических функций. Получим значение характеристической функции при нахождении в коалиции T игроков A_0 и C согласно (48).

$$v(T) = v(\{C, A_0\}) = 1251.985 \tag{80}$$

По аналогичной методике получим значение характеристической функции от коалиции T, состоящей из игроков B_1, C согласно (49).

$$v(T) = v(\{C, B_1\}) = 462.332 \tag{81}$$

Далее, рассмотрим значение характеристической функции от коалиции T, состоящей из игроков B_2, C согласно (50).

$$v(T) = v(\{C, B_2\}) = 389 \tag{82}$$

Численно найдем значения характеристических функций от трехэлементных коалиций. Рассмотрим сначала характеристическую функцию от коалиции T игроков B_1, B_2 и C согласно (51).

$$v(T) = v(\{C, B_1, B_2\}) = 565.333 \tag{83}$$

Рассмотрим характеристическую функцию от коалиции T игроков B_1, A_0 и C согласно (52).

$$v(T) = v(\{C, A_0, B_1\}) = 1892.083 \tag{84}$$

Рассмотрим характеристическую функцию от коалиции T игроков B_2, A_0 и C согласно (53).

$$v(T) = v(\{C, A_0, B_2\}) = 1880.985 \tag{85}$$

Теперь проведем вычисления для нахождения численного значения характеристической функции от гранд-коалиции N, содержащей всех игроков A_0, B_1, B_2, C согласно (54).

$$v(N) = v(\{A_0, B_1, B_2, C\}) = 2534.786 \tag{86}$$

Уже можно увидеть, что нахождение подкоалиции игроков A_0 и C в любой коалиции увеличивает его полезность, а значит взаимодействие игроков A_0 и C достаточно ценно. Построим вектор Шепли для данной кооперативной игры. Напомним, что поскольку кооперативная игра определена как антагонистическая игра двух коалиций T и $T' = \{A_0, B_1, B_2, C\} \backslash T$, то справедливо:

$$\begin{aligned} v(T) &= v(\{A_0\}) = 0 \\ v(T) &= v(\{B_1\}) = 0 \\ v(T) &= v(\{B_2\}) = 0 \end{aligned} \tag{87}$$

Теперь приступим непосредственно к нахождению самого вектора Шепли.

$$\begin{aligned} Sh_{A_0} &= 816.339 \\ Sh_{B_1} &= 246.180 \\ Sh_{B_2} &= 230.259 \\ Sh_C &= 1242.006 \end{aligned} \tag{88}$$

Как видно, кооперация игроков A_0 и C является для них выгодной, даже несмотря на то, что большинство элементов - которые есть величины ресурса типа j, необходимого для производства 1 единицы продукции типа p – матрицы M больше элементов матрицы D. Из того, что справедливо неравенство $Sh_{A_0} \geq \mathrm{K}_1$ и $Sh_C \geq \mathrm{K}_4$, следует, что игроки будут кооперироваться для производства глиняных изделий 7 типов, однако это суждение не справедливо для игроков B_1 и B_2, потому что с учетом решения задачи нахождения ситуации равновесия по Нэшу неравенство значений компонентов вектора Шепли для игрока B_1 выглядит как $\overline{Sh_{B_1}} \geq \widetilde{Sh_{B_1}} \geq \widehat{Sh_{B_1}} \geq Sh_{B_1}$, и для игрока B_2 как $\widetilde{Sh_{B_2}} \geq \widehat{Sh_{B_2}} \geq \overline{Sh_{B_2}} \geq Sh_{B_2}$. Следовательно, кооперации от игроков B_1 и B_2 ожидать не стоит.

Таким образом, в Главе II была представлена алгоритмическая структура, показана логика программной реализации и утверждена модификация метода Монте-Карло, позволяющая численно находить в неантагонистической иерархической ромбовидной игре Г ситуацию равновесия по Нэшу, вычислять тремя разными способами минимальную гарантированную полезность игроков B_1 и B_2, вычислять для коалиции этих игроков вектор Шепли а также вычислять значения характеристических функций для кооперативной игры на ромбовидной структуре с нахождением вектора Шепли для реализации дележа по коалиционным разбиениям.

Заключение

В данной работе рассмотрена проблематика задачи распределения ресурсов в ромбовидной иерархической теоретико-игровой модели, которая описывает свойства рыночных, межрегиональных, экологических и иных классов задач.

Конкретизация теоретико-игровой модели через определение множеств стратегий игроков в игре Г показала, что добавление связи поставок ресурсов между центром и игроком нижнего уровня, которая учитывает формализацию производства с использованием ресурсов управляющего центра отдельно от производства с использованием ресурсов игроков среднего уровня, приводит к появлению дополнительной системы линейных неравенств игрока нижнего уровня. Данное расширение обосновывается тем, что игроки среднего уровня посылают игроку нижнего уровне не тот же набор типов ресурсов, что и управляющий центр. Это существенное для реальных задач дополнение является оригинальным и не рассматривалось ранее в литературе.

Рассмотрен процесс нахождения ситуации равновесия по Нэшу в определенной игре Г. Для этого доказана лемма о существовании ситуации равновесия по Нэшу и игре Г. Конкретизирована теоретико-игровая модель через определение множеств стратегий игроков, в частности, составление систем неравенств. Показано, что добавление связи поставок ресурсов между центром и игроком нижнего уровня приводит к появлению второй системы неравенств игрока нижнего уровня, которая учитывает формализацию производства с использованием ресурсов управляющего центра отдельно от производства с помощью ресурсов игроков среднего уровня. Были построены оптимизационные задачи линейного и нелинейного программирования с параметрами и показана возможность нахождения ситуации равновесия по Нэшу. Чтобы разрешить проблему нахождения равновесия в игре Г была введена кооперативная подыгра между игроками среднего уровня. Разработано и учтено три варианта одноэлементных характеристических функций игроков среднего уровня для вычисления тремя различными способами минимальной гарантированной полезности, необходимой для вычисления вектора Шепли - принятого принципа оптимальности. Представлены численные примеры, показывающие специфику значений вектора Шепли при использовании трех различных подходов к определению минимальной гарантированной полезности. Сформулирована в общем виде кооперативная игра на ромбовидной структуре, составлены равенства, которые определяют действия игроков в рамках антагонистической игры двух коалиций. Для каждой из коалиции в кооперативной игре выведены формулы для вычисления вектора Шепли. Для программной реализации составлен алгоритм с модифицированным методом Монте-Карло, который позволил конкретнее описать методику случайного поиска для нахождения ситуации равновесия по Нэшу, значений характеристических функций в кооперативной игре и вектора Шепли через полное покрытие области допустимых решений систем линейных неравенств игроков среднего уровня. Определена структура алгоритма, проведен алгоритмический анализ и выявлены особенности применения модифицированного метода Монте-Карло к решению задачи. По алгоритму построена программная реализация, которая позволила численно решить данную задачу. Приведен пример выполнения программы по заданному алгоритму, который показывает специфику в использовании трех подходов к определению одноэлементных характеристических функций.

В ходе решения численного примера наглядно показано, что от способа расчета минимальной гарантированной полезности зависит то, какие значения компонент вектора Шепли будут найдены для игроков среднего уровня, и то, что игрокам не всегда выгодно исходить из "оптимистичного"варианта, при котором управляющий центр отправляет только один или два нулевых вектора ресурсов непосредственно игрокам среднего уровня. В рамках продолжения данной научной проблематики возможны улучшения по части эв-

ристик и методов, которые можно использовать для решения систем линейных неравенств в данной структуре. Рассмотрение динамики процесса также позволит полно раскрыть потенциал представленной иерархической структуры. Не исключается также дальнейшее расширение структуры для лучшего отражения отношений в иерархии между игроками.

Список литературы

[1] Amer R., Carreras F. Cooperation Indices and Weighted Shapley Values. Mathematics of Operations Research. – Informs, USA, 1997, 14p. URL: https://pubsonline.informs.org/doi/pdf/10.1287/moor.22.4.955

[2] Gorelov, M.A. and Kononenko, A.F., Dynamic Models of Conflicts. III. Hierarchical Games, Autom.Remote.Control, 2015, vol.76 no. 2, pp. 89-106.

[3] Hillier F.S. Introduction to operation research, Stanford University. – Tenth edition, 1411p.

[4] Jackson, Matthew O. Social and Economic Networks. Princeton; Oxford: Princeton University Press, 2008. URL: www.jstor.org/stable/j.ctvcm4gh1

[5] Nisan N., Roughgarden T., Tardos E., Vazirani V.V. Algorithmic Game Theory. — Cambridge University Press, New York, NY, USA, 2007. – 776p.

[6] Petrosyan L., Pankratova Y. Equilibrium and Cooperation in Repeated Hierarchical Games //International Conference on Mathematical Optimization Theory and Operations Research. – Springer, Cham, 2019. – С. 685-696.

[7] Зоркальцев В. И. З – 86 Системы линейных неравенств: учебное пособие / В. И. Зоркальцев, М. А.Киселева – Иркутск: Изд-во Иркутского гос. ун-та, 2007. – 128с.

[8] Мазалов В. В. Математическая теория игр и приложения: Учебное пособие. – СПб.: Издательство 'Лань', 2010. – 448с.

[9] Наумова Н. И. Вектор Шепли и его обобщения. Учебное пособие. – СПб.: Изд-во ВВМб 2017. – 60с.

[10] Теория игр: учебная литература для вузов/ Л. А. Петросян, Н. А. Зенкевич, Е. В. Шевкопляс. – 2-е. изд. перераб. и доп. – СПб.:БХВ-Петербург, 2012–432с.

[11] Уксусов С.Н., Баркалов С.А., Зенищева Г.В. Решение задачи планирования производства с переменными ресурсами методом жордановых исключений // Сборник трудов международной конференции «Управление современными сложными системами». 2013. С. 210-220.

Приложение

В приложении представлен листинг программной реализации согласно алгоритму программы, описанной в Главе II.

```
import numpy as np
from cvxopt import matrix, solvers
from cvxopt.modeling import variable, max
from cvxopt.modeling import op, dot
import pylab
import random

def Task_data(*args):
    D = [[1.,3.,9.],
         [2.,8.,4.],
         [9.,2.,3.]]
    A_1 = [[6,8,1],
           [4,0,4],
           [5,7,8]]
    A_2 = [[4,3,9],
           [2,0,4],
           [5,2,8]]
    b = [100, 200, 300]
    c1 = [1, 4, 7]
    c2 = [7, 9, 4]
    c3 = [6, 8, 2]
    c4 = [3., 2., 5.]
    numbers_u = 5000
    numbers_omg = 5000
    xi_1 = [1,2,3]
    xi_2 = [3,4,5]
    gamma_resource_1 = [2,4,5]
    gamma_resource_2 = [2,4,5]
    return D, A_1, A_2, b, c1, c2, c3, c4, xi_1, xi_2,
       gamma_resource_1, gamma_resource_2, numbers_u, numbers_omg

def Upper_bounds_of_solutions(A:list, u:list, xi:list):
    import math
    A_1_new = []
    for i in A:
        if min(i) == 0:
            i.insert(i.index(0), math.inf)
            i.pop(i.index(0))
    for i, j in zip(A, u):
        for k in i:
            x = j/k + h
            A_1_new.append(x)
    a = []
    for r in range(len(A)):
        a.append([])
        for c in A_1_new:
            a[r].append(c)
    x = [len(i) for i in A]
```

```
        x = int(max(x))
        for i, j in zip(range(len(a)),a):
            if i == 0:
                del a[0][x:]
            if i == len(a) - 1:
                del a[len(a) - 1][0:-x]
            if i != 0 and i != len(a) - 1:
                del a[i][:x*i]
                del a[i][x:]
        result = [max(column) for column in zip(*a)]
        return result

    def random_u(b:list):
        import random
        array_random_u =[]
        array_random_u_1 =[]
        dim = len(b)
        for i in range(dim):
            cum = 0
            for j in range(3):
                h = random.uniform(0, b[i] - cum)
                array_random_u.append(h)
                cum += h
            c = b[i] - cum
            for k in range((len(array_random_u) - int((len(array_random_u)
               /(i+1)))), len(array_random_u)):
                array_random_u_1.append(array_random_u[k]+(c/3))
            u_1 = []
            u_2 = []
            u_3 = []
            for i in range(0, len(array_random_u_1), 3):
                u_1.append(array_random_u_1[i])
            for i in range(1, len(array_random_u_1), 3):
                u_2.append(array_random_u_1[i])
            for i in range(2, len(array_random_u_1), 3):
                u_3.append(array_random_u_1[i])
        return u_1, u_2, u_3

    def random_omg(A:list, u:list, xi:list):
        import random
        u_xi = []
        for i in range(len(u)):
            for j, k in zip(u, xi):
                u_xi.append(j+k)
        Upper_omg = Upper_bounds_of_solutions(A, u_xi)
        omg_array = []
        for h in range(len(Upper_omg)):
            omg_array.append(random.uniform(0, Upper_omg[h]))
        return omg_array

    def available_resources(x:int, y:int):
        import random
        res = []
```

```
        for i in range(x):
            res.append(random.uniform(0,y))
        return res

    def Acceptable_solution(A:list, omg:list, u:list, xi:list):
        for i,l,h in zip(A, u, xi):
            set_omg = []
            set_u3 = []
            a = 0
            b = l+h
            for j,k in zip(i, omg):
                a += j*k
            if a <= b:
                set_omg.append(omg)
        if len(A) != len(set_omg):
            set_omg = []
            set_u3 = []
            return set_omg, set_u3
        else:
            set_omg = set_omg[0]
            set_u3.append(u_3)
            return set_omg, set_u3

    def Main_generation(A_1:list, xi_1:list, A_2:list, xi_2:list,
       numbers_u:int, numbers_omg:int):
        all_set_u1 = []
        all_set_u2 = []
        all_set_u3 = []
        for i in range(numbers_u):
            a = random_u(b)
            u_1 = a[0]
            u_2 = a[1]
            u_3 = a[2]
            for j in range(numbers_omg):
                omg_1 = random_omg(A_1,u_1,xi_1)
                omg_2 = random_omg(A_2,u_2,xi_2)
                all_set_u1.append([u_1])
                all_set_u2.append([u_2])
                all_set_u3.append([u_3])
                set_omg_1.append(acceptable_solution(A_1, omg_1, u_1, xi_1
                    )[0])
                set_u3_1.append(acceptable_solution(A_1, omg_1, u_1, xi_1)
                    [1])
                set_omg_2.append(acceptable_solution(A_2, omg_2, u_2, xi_2
                    )[0])
                set_u3_2.append(acceptable_solution(A_2, omg_2, u_2, xi_2)
                    [1])
        return  set_omg_1, set_u3_1, set_omg_2, set_u3_2

    def Converting_arrays(set_omg_1:list, set_u3_1:list, set_omg_2:list,
       set_u3_2:list, numbers_u:int, numbers_omg:int):
        new_set_omg_1 = [0]*numbers_u
        new_set_omg_2 = [0]*numbers_u
```

```
new_set_u3_1 = [0]*numbers_u
new_set_u3_2 = [0]*numbers_u
new_set_omg_1_1 = []
new_set_omg_2_1 = []
for i in range(numbers_u):
    for j in range((numbers_omg_1*i), numbers_omg_1*(i+1)):
        if set_omg_1[j] != []:
            if type(new_set_omg_1[i]) == list:
                new_set_omg_1[i].append(set_omg_1[j])
                new_set_u3_1.append(set_u3_1[j])
            else:
                new_set_omg_1.insert(i, [])
                new_set_omg_1[i].append(set_omg_1[j])
                new_set_omg_1.pop(i+1)
                new_set_u3_1.insert(i, [])
                new_set_u3_1[i].append(set_u3_1[j])
                new_set_u3_1.pop(i+1)
for i in range(numbers_u):
    for j in range((numbers_omg_2*i),numbers_omg_2*(i+1)):
        if set_omg_2[j] != []:
            if type(new_set_omg_2[i]) == list:
                new_set_omg_2[i].append(set_omg_2[j])
                new_set_u3_2.append(set_u3_2[j])
            else:
                new_set_omg_2.insert(i, [])
                new_set_omg_2[i].append(set_omg_2[j])
                new_set_omg_2.pop(i+1)
                new_set_u3_2.insert(i, [])
                new_set_u3_2[i].append(set_u3_2[j])
                new_set_u3_2.pop(i+1)
for i in range(numbers_u):
    for j in range((numbers_omg*i),(numbers_omg*(i+1))):
        if set_omg_1[j] != []:
            new_set_omg_1_1.append(set_omg_1[j])
for i in range(numbers_u):
    for j in range((numbers_omg*i),(numbers_omg*(i+1))):
        if set_omg_2[j] != []:
            new_set_omg_2_1.append(set_omg_2[j])
final_set_omg_1 = []
final_set_omg_2 = []
final_set_u3_1 = []
final_set_u3_2 = []
for i in range(len(new_set_omg_1)):
    if new_set_omg_1[i] != 0 and new_set_omg_1[i] != [] and
       new_set_omg_2[i] != 0 and new_set_omg_2[i] != [] :
        final_set_omg_1.append(new_set_omg_1[i])
        final_set_omg_2.append(new_set_omg_2[i])
        final_set_u3_1.append(new_set_u3_1[i])
        final_set_u3_2.append(new_set_u3_2[i])
m = 0
for i in final_set_omg_1:
    if i == 0:
        m += 1
```

```
        n = 0
        for i in final_set_omg_2:
            if i == 0:
                n += 1
        s = 0
        for i in range(len(final_set_omg_1)):
            s += len(final_set_omg_1[i])
        k = 0
        for i in range(len(final_set_omg_2)):
            k += len(final_set_omg_2[i])
        if len(new_set_omg_1_1) == s and len(new_set_omg_2_1) == k and m
           == 0 and n == 0:
            return final_set_omg_1, final_set_omg_2, final_set_u3_1,
               final_set_u3_2

    def Solving_optimization_problems(full_set, gamma_resource_1:list,
       gamma_resource_1:list):
        from cvxopt import matrix, solvers
        from cvxopt.modeling import variable, max
        from cvxopt.modeling import op, dot
        for elem_omg_1,elem_omg_2,elem_set_u3_2 in zip(full_set[0],
           full_set[1],full_set[3]):
            for i in elem_omg_1:
                for j, k in zip(elem_omg_2, elem_set_u3_2):
                    vector_omg_1 = i
                    vector_omg_2 = j
                    vector_u_3 = k
                    new_u3.append(vector_u_3)
                    v = variable(len(gamma_resource_1), 'v')
                    omg_1 = variable(len(gamma_resource_1), 'omg_1')
                    omg_2 = variable(len(gamma_resource_1), 'omg_2')
                    u_3 = variable(len(gamma_resource_1), 'u_3')
                    gamma_1 = variable(len(gamma_resource_1), 'gamma')
                    gamma_2 = variable(len(gamma_resource_1), 'gamma')
                    c4 = matrix (c4)
                    D = matrix(D)
                    M = matrix(M)
                    eq1 = (D*v <= omg_1 + omg_2 + gamma_1)
                    eq2 = (M*v <= u_3 + gamma_2)
                    ineq1 = (v>=0)
                    ineq2 = (omg_1>=0)
                    ineq3 = (omg_2>=0)
                    ineq4_1 = (gamma_1>=0)
                    ineq4_2 = (gamma_2>=0)
                    ineq5 = (u_3 == vector_u_3)
                    ineq6 = (omg_1 == vector_omg_1)
                    ineq7 = (omg_2 == vector_omg_2)
                    ineq8 = (gamma == gamma_resource)
                    lp = op(dot(c4, v), [eq1,eq2,ineq1,ineq2,ineq3,ineq4_1
                       ,ineq4_2, ineq5, ineq6, ineq7, ineq8])
                    lp.solve('glpk')
                    two_set_omg_1.append(list(omg_1.value))
                    two_set_omg_2.append(list(omg_2.value))
```

```
                two_set_obj.append(list(lp.objective.value()))
                two_set_v.append(list(v.value))
                two_set_u_3.append(list(u_3.value))
    for i in two_set_obj:
        for j in i:
            two_set_obj2.append(j*-1)
    return two_set_omg_1, two_set_omg_2, two_set_v, two_set_obj2,
       two_set_u_3

def Checking_restrictions(two_set_v):
    for h in range(5):
        k = 0
        for i in two_set_v:
            for j in i:
                if j < 0:
                    k += 1
        for u in range(k+10):
            if k != 0:
                for i in two_set_v:
                    for j in i:
                        if j < 0:
                            x = two_set_v.index(i)
                            two_set_v.pop(x)
                            two_set_omg_1.pop(x)
                            two_set_omg_2.pop(x)
                            two_set_u_3.pop(x)
                            two_set_obj2.pop(x)
                            two_set_obj.pop(x)
                            new_set_omg_1.pop(x)
                            new_set_omg_2.pop(x)
                            set_u1.pop(x)
                            set_u2.pop(x)
                            set_u3_1.pop(x)
                            set_u3_2.pop(x)
                k_new = 0
                for i in two_set_v:
                    for j in i:
                        if j < 0:
                            k_new += 1
    return k, k_new

def target_functions(two_set_v, c1, c2, c3, c4):
    import numpy as np
    for i in two_set_v:
            i = np.array(i)
            x = np.dot(c2, i) + np.dot(c3, i)
    v_opt = two_set_v[set_value_obj_A0.index(max(set_value_obj_A0))]
    v_opt = np.array(v_opt)
    c2 = np.array(c2)
    c3 = np.array(c3)
    opt_coalition = np.dot(c2, v_opt) + np.dot(c3, v_opt)
    c1 = np.array(c1)
    set_value_obj_A0 = []
```

```
        for i in two_set_v:
                i = np.array(i)
                x = np.dot(c1, i)
                set_value_obj_A0.append(x)
        opt_A0 = max(set_value_obj_A0)
        v_opt = two_set_v[set_value_obj_A0.index(max(set_value_obj_A0))]
        v = two_set_v[set_value_obj_A0.index(max(set_value_obj_A0))]
        ind = set_value_obj_A0.index(max(set_value_obj_A0))
        x1 = new_set_omg_1.index(two_set_omg_1[ind])
        x2 = new_set_omg_2.index(two_set_omg_2[ind])
        v_opt = two_set_v[set_value_obj_A0.index(max(set_value_obj_A0))]
        omg_1_fix = new_set_omg_1[set_value_obj_A0.index(max(
           set_value_obj_A0))]
        omg_2_fix = new_set_omg_2[set_value_obj_A0.index(max(
           set_value_obj_A0))]
        j9 = two_set_omg_1[two_set_v.index(v_opt)]
        f0 = two_set_omg_2[two_set_v.index(v_opt)]
        x3 = new_set_omg_1.index(j9)
        x4 = new_set_omg_2.index(f0)
        u3 = two_set_u_3[ind]
        u1 = set_u1[set_u3_1.index(set_u3_1[ind])]
        u2 = set_u2[set_u3_1.index(set_u3_1[ind])]
        return opt_A0, opt_coalition, opt_C, v, u1, u2, u3, omg_1_fix,
           omg_2_fix
```