Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

# Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**СТОХАСТИЧНІ КОАЛІЦІЙНІ ІГРИ З ЛОКАЛЬНОЮ СТРУКТУРОЮ ВЗАЄМОДІЇ/STOCHASTIC COALITION GAMES WITH LOCAL INTERACTION**»

Виконав: студент 4-го року навчання,

Спеціальності

113 Прикладна математика

Коваленко Аркадій Юрійович

Керівник Чорней Р.К.,_____

доцент, к.ф.-м.н.

Рецензент _____

Кваліфікаційна робота захищена

З оцінкою _____

Секретар ЕК _____

«____» _____ 20____р.

Київ 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри математики, доцент, к.ф.-м.н.

Р. К. Чорней

(підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Коваленку Аркадію Юрійовичу факультету інформатики 4-го курсу

ТЕМА Стохастичні коаліційні ігри з локальною структурою взаємодії/Stochastic coalition games with local interaction

Зміст ТЧ до курсової роботи:

1. Індивідуальне завдання
2. Календарний план
3. Вступ
4. Розділ 1. Алгоритм для пошуку оптимальних стратегій
5. Розділ 2. Розробка програми для знаходження оптимальних стратегій
6. Розділ 3. Аналіз стохастичних коаліційних ігор з локальною структурою взаємодії
7. Висновки
8. Список використаних джерел

Дата видачі «___» _____ 2024 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

## Календарний план виконання роботи

| № | Назва етапу кваліфікаційної роботи | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Вибір теми роботи | 23.10.2023 | |
| 2. | Дослідження літератури за темою роботи | 29.12.2023 | |
| 3. | Моделювання процесу гри | 20.02.2024 | |
| 4. | Знаходження алгоритму для пошуку оптимальних стратегій | 20.03.2024 | |
| 5. | Написання програми за допомогою алгоритму | 05.04.2024 | |
| 6. | Написання кваліфікаційної роботи | 27.04.2024 | |
| 7. | Створення презентації. | 23.05.2024 | |
| 8. | Захист роботи | 03.06.2024 | |

Студент Коваленко А.Ю.

Керівник Чорней Р.К.

«____» _____

**ЗМІСТ**

**АНОТАЦІЯ** ..... 5

**ВСТУП** ..... 6

**РОЗДІЛ 1. АЛГОРИТМ ДЛЯ ПОШУКУ ОПТИМАЛЬНИХ СТРАТЕГІЙ** .. 9

**1.1 Базові поняття та позначення** ..... 9

**1.2 Знаходження ваг** ..... 11

**1.3 Покращення стратегії** ..... 13

**1.4 Ітераційний цикл** ..... 15

**1.5 Доведення ефективності алгоритму** ..... 16

**1.6 Ітераційний метод знаходження оптимальних стратегій для коаліцій** 18

**РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ДЛЯ ЗНАХОДЖЕННЯ ОПТИМАЛЬНИХ СТАРАТЕГІЙ** ..... 20

**2.1 Визначення базових структур для гри** ..... 20

**2.2 Задання умов та початкових значень для гри** ..... 23

**2.3 Симуляція гри** ..... 24

**2.4 Інтеграція алгоритму пошуку оптимальних стратегій** ..... 26

**РОЗДІЛ 3. АНАЛІЗ СТОХАСТИЧНИХ КОАЛІЦІЙНИХ ІГОР З ЛОКАЛЬНОЮ СТРУКТУРОЮ ВЗАЄМОДІЇ** ..... 35

**3.1 Аналіз гри з двома гравцями** ..... 35

**3.2 Аналіз гри з чотирма гравцями** ..... 43

**3.3 Аналіз гри з трьома гравцями** ..... 46

**ВИСНОВКИ** ..... 54

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** ..... 55

# АНОТАЦІЯ

В роботі досліджуються стохастичні коаліційні ігри з локальною структурою взаємодії. Метою роботи було знаходження алгоритму для пошуку оптимальних стратегій, написання програми, що його реалізує, та аналіз ігор за допомогою програми.

У першому розділі розглянутий алгоритм для пошуку оптимальних стратегій для стохастичних коаліційних ігор з локальною структурою взаємодії та процес його пошуку.

У другому розділі міститься інформація про написання програми, яка реалізує алгоритм, описаний у першому розділі.

Третій розділ містить аналіз різних ігор за допомогою описаної в другому розділі програми.

# ВСТУП

Для аналізування та прогнозування процесів в інженерії, економіці та соціології вже доволі давно використовують математичні моделі. Серед них особливе місце займають стохастичні ігри. Вони дозволяють не тільки змоделювати та проаналізувати можливі ситуації, але й обрати такі рішення, що приведуть до отримання найкращих результатів. Ключовою проблемою подібних ігор є знаходження алгоритмів, які за допомогою повної чи неповної інформації про гру, зможуть знайти оптимальні стратегії для гравців.

Стохастичні коаліційні ігри - це математичні моделі, що дозволяють вивчати взаємодію між гравцями, об’єднаними в коаліції. Гравці утворюють коаліції, які спільно приймають рішення та ділять прибутки та втрати.

Локальна структура взаємодії в стохастичних іграх визначається взаємодією лише тих гравців, які взаємодіють між собою. Це дуже розповсюджена ситуація, коли гравці не впливають прямо на один одного. Проте вони цілком можуть впливати на спільного сусіда і таким чином змінювати ситуацію друг для друга.

Серед процесів, які можуть бути змодельовані за допомогою стохастичних коаліційних ігор з локальною структурою взаємодії, варто виділити такі як розподіл ресурсів, формування соціальних груп, торговельні відносини, конкуренція та навіть ведення бойових дій.

Метою даної роботи є:

1. Описати правила та базові позначення для стохастичних коаліційних ігор з локальною структурою взаємодії.
2. Проаналізувати формули та алгоритми, що використовуються в стохастичних іграх, та розробити алгоритм пошуку оптимальних стратегій.
3. Довести правильність роботи алгоритму за допомогою математичних методів

4. Написати програму, що для введених даних про гру та гравців, за допомогою раніше розробленого алгоритму знаходитиме оптимальні стратегії для коаліцій.
5. Проаналізувати отримані програмою результати для ігор з різними початковими умовами.
6. Зробити висновки про виконану роботу

Об'єктом дослідження даної роботи є стохастичні коаліційні ігри з локальною структурою взаємодії. В даній роботи будуть досліджені ігри з двома коаліціями, так як вони є іграми з нульовою сумою. Також всі значення, що будуть використовуватися для станів та дій гравців – дискретні.

Предметом дослідження є алгоритм пошуку оптимальних стратегій для стохастичних коаліційних ігор з локальною структурою взаємодії. Алгоритм повинен працювати та правильно знаходити оптимальні рішення для будь яких ігор, що підпадають під об’єкт дослідження.

Актуальність даної роботи полягає у можливості застосування отриманих результатів для аналізу різних процесів, а також у знаходженні оптимальних стратегій, які найкраще використовувати в заданих умовах. Дослідження та розвиток алгоритмів для пошуку оптимальних стратегій може дуже допомогти у вирішенні реальних проблем або стати базою для більш складних та глибоких досліджень в даній галузі.

Наукова новизна дослідження полягає в розробці алгоритму пошуку оптимальних стратегій для стохастичних ігор з локальною структурою взаємодії для двох коаліцій з дискретними значеннями станів та дій гравців, та його програмній реалізації. Так як це доволі специфічний випадок стохастичної гри, даних якими можна було б скористатися при наявності подібної проблеми не дуже багато, тому ця робота зможе додати нові рішення, збільшити базу знань в теорії ігор та розкрити нові аспекти та проблеми.

Для написання даної роботи було використано багато джерел, але основними слугували стаття [1], в якій описані принципи, позначення, властивості та формули, що були використані, і [2], де був описаний ітераційний метод, за допомогою якого можна знаходити оптимальні рішення в іграх, що використовують Марківські ланцюги.

# РОЗДІЛ 1. АЛГОРИТМ ДЛЯ ПОШУКУ ОПТИМАЛЬНИХ СТРАТЕГІЙ

## 1.1 Базові поняття та позначення

Стохастична гра з локальною структурою взаємодії будується на базі скінченного неорієнтованого графа. Вершини графа позначають гравців. А ребра означають те, що ці гравці взаємодіють один з одним. Локальність ігор якраз і зумовлена цими зв'язками. Гравці, що взаємодіють один з одним, можуть визначати свій наступний стан залежно не тільки від свого поточного стану та дії, а й від поточних станів гравців, з якими він взаємодіє. Також, після прийняття рішень, гравці виплачують один одному деяку числову суму грошей, не обов'язково додатню. Виплачувати гравець може лише такому гравцю з яким він пов'язаний та не є в одній коаліції. Сума виплати – це функція, що залежить від поточного стану обох гравців та їх дій. Отримати прибуток інакше як від іншого гравця (або виплативши іншому гравцю від'ємну суму) – неможливо. Таким чином отримуємо гру з нульовою сумою, де ціль кожного гравця – максимізувати свій прибуток.

Коаліційною будемо називати таку стохастичну гру, де гравці намагаються отримати максимальний дохід не для себе, а для своєї коаліції. Таким чином, навіть якщо гравці однієї коаліції зв'язані один з одним, визначати функцію виплат для них немає сенсу, адже вони мають спільний дохід. В даній роботі буде розглянуто ігри з двома коаліціями. Таким чином, в нашій грі з нульовою сумою матимемо значення, що показуватиме скільки сумарно прибутку отримала перша коаліція. Тоді це означає, що метою першої коаліції буде максимізувати це значення, а другої – мінімізувати його.

Позначимо $V$ – множина вершин неорієнтованого скінченного графу, $B -$ множина ребер графа. Для довільної вершини $i$ введемо поняття сусідства $N(i)$, яким будемо позначати множину вершин, з якими вершина $i$ сполучена ребрами $N(i) = \{j: \{i,j\} \in B\}$. Повним сусідством вершини $i$ вважатимемо її сусідство, включаючи саму вершину k: $\widetilde{N}(i) = N(i) \cup i$. Для довільної множини

вершин I, позначимо сусідство $N(I) = \bigcup_{i \in I} N(i) \setminus I$ і повне сусідство $\widetilde{N}(I) = \bigcup_{i \in I} N(i) \cup I$. Для всіх вершин $i \in V$ введемо поняття стану $X_i$, який може визначатися будь-якою зліченною мірою. Тоді позначимо $X = \times_{i \in V} X_i$ – стан повної системи і $N$ – кількість можливих станів системи. Також позначимо стан для множини вершин $I \in V$ як $x_i = \times_{i \in I} X_i$. Позначимо $a_i^j$ – можлива дія гравця $i$ в стані $j$ [1].

Так як в подібних іграх використовуються марківські процеси і гра має локальну структуру взаємодії, стратегії будуть залежати лише від стану сусідніх гравців, стану гравця та його дії. Також вони будуть стаціонарними, так як розглядатися будуть лише ігри з нескінченним горизонтом. Тоді позначимо, що стратегія для гравця – це функція, аргументами якої є стан гравця та стани його сусідів, а результатом – дія гравця, яку він здійснить при таких умовах. Сформуємо список всіх можливих стратегій для кожного гравця. Нехай $s_i^j$ – стратегія $j$ гравця $i$, а $S_i$ – список всіх можливих стратегій гравця $i$.

В даній роботі акцент буде зроблено на дискретних множинах станів і дій, і в подальшому вони будуть позначатися як невід'ємні цілі числа, починаючи з нуля. Так як метою є знайти оптимальну стратегію саме для коаліції, сформуємо список, що складатиметься зі всіх можливих стратегій для коаліцій. Тоді введемо $k_i^j$ – стратегія $j$ для коаліції $i$, а $K_i$ – список всіх можливих стратегій коаліції $i$. Стратегія коаліції – всі можливі комбінації всіх можливих стратегій її гравців. Наприклад, якщо в коаліції є два гравця 0 і 1, і в кожного з гравців є дві можливі стратегії, а саме $s_0^0, s_0^1 - \text{для гравцяя } 0, \text{та } s_1^0, s_1^1 - \text{для } 1$, тоді матимемо чотири можливих стратегії для коаліції, а саме: $(s_0^0, s_1^0)$, $(s_0^1, s_1^0)$, $(s_0^0, s_1^1)$, $(s_0^1, s_1^1)$.

Позначимо ймовірність переходу системи зі стану $i$ в стан $j$ при вибраних стратегіях для обох коаліцій як $p_{ij}$.

Будемо вважати, що гра буде тривати нескінченно або надзвичайно велику кількість кроків. В такій ситуації, для певної коаліції оптимальною вважатимемо таку стратегію, очікуваний прибуток якої за крок буде більшим.

В даній роботі не будуть використовуватися змішані стратегії, так як головна задача – вибрати таку стратегію, що максимізує середній прибуток за хід, а цього якраз можна досягнути за допомогою саме чистих стратегій. Навіть якщо розглядати окремі ситуації, в яких дві чисті стратегії досягають однакового середнього прибутку за крок, обирати змішану стратегію з ними обома сенсу не має, так як середній прибуток ми цим ніяк не змінимо.

Для знаходження оптимальних стратегій будемо використовувати методи динамічного програмування. Такі методи часто використовуються для пошуку оптимальних стратегій в іграх, де гравці повинні приймати рішення, чи послуговуватися певними стратегіями. Базою для побудови алгоритму було обрано ітераційний метод запропонований Рональдом Говардом у його роботі [2]. Модифікація даного алгоритму що підходить для постановки задач, які вирішувати в межах даної роботи і буде продемонстрована далі в розділі.

### 1.2 Знаходження ваг

Припустимо розглядається система з заданим графом гравців, їх стани та дії, допустимі при довільних станах, і вибрані та зафіксовані стратегії для обох коаліцій. Введемо функцію $q_i$ – середній очікуваний прибуток першої коаліції в стані $i$, при вибраних стратегіях для обох коаліцій. Введемо поняття ціни гри $c$. В нашому випадку ціною гри позначатимемо величину, що при вибраних стратегіях для коаліцій, позначатиме очікувану середню кількість грошей, отриманих першою коаліцією за крок (очевидно, що це значення буде від’ємним, якщо друга коаліція в середньому буде отримувати більше). Нехай $\pi_i$ – ймовірність потрапити в стан $i$, після великої кількості кроків n ($n \to \infty$). Тоді

якщо гра триватиме нескінченно або величезну кількість кроків, справедливим буде вираз:

$$c = \sum_{i=0}^{N-1} \pi_i q_i$$

Також додамо функцію $v_i(n)$ – очікуваний середній прибуток, отриманий першою коаліцією за n кроків, якщо система почала функціонувати зі стану $i$. Також введемо відносну функцію $v_i$, що позначатиме наскільки збільшиться прибуток першою коаліції, якщо починати гру зі стану $i$ відносно середнього прибутку при виборі випадкового стану як стартового. Очевидно, що $v_N = 0$. Тоді матимемо:

$$v_i(n) = nc + v_i$$

Також $v_i(n)$ – можна вивести за допомогою рекурсії. Так як прибуток за $n$ кроків починаючи зі стану $i$, буде рівним сумі прибутку за поточний крок і сумі прибутків за n-1 крок з кожного стану помноженого на ймовірність переходу в ці стани, матимемо:

$$v_i(n) = q_i + \sum_{j=0}^{N-1} p_{ij} v_j\,(n-1)$$

Прирівнявши ці дві умови, отримуємо:

$$nc + v_i = q_i + \sum_{j=0}^{N-1} p_{ij} v_j (n-1)$$

$$nc + v_i = q_i + \sum_{j=0}^{N-1} p_{ij}((n-1)c + v_j)$$

$$nc + v_i = q_i + (n-1)c \sum_{j=0}^{N-1} p_{ij} + \sum_{j=0}^{N-1} p_{ij} v_j$$

Так як ймовірність переходу зі стану $i$ в будь-який інший стан дорівнює одиниці, одержуємо:

$$nc + v_i = q_i + (n-1)c + \sum_{j=0}^{N-1} p_{ij} v_j$$

$$c + v_i = q_i + \sum_{j=0}^{N-1} p_{ij} v_j \qquad (1.1)$$

Таким чином з'являється система з N лінійних алгебраїчних рівнянь з N+1 невідомими, а саме $v_0, v_1 \dots v_{N-1}$ і $c$. Так як така система буде мати безліч розв'язків, мусимо позбутися хоча б одного невідомого. Насправді, в даній ситуації можна обмежитися відносними вагами $v_i$. Наприклад, візьмемо $v_{N-1} = 0$. Тоді в нашому випадку $v_i$ позначатимуть, наскільки більше прибутку отримає перша коаліція, якщо система починатиметься зі стану $i$ відносно стану $N-1$. Візьмемо до уваги той факт, що при цьому значення $v_i$ все ж будуть різнитися від тих, які ми використовували в формулах на константне значення, а саме на попереднє значення $-v_{N-1}$. Проте це не має критичного значення при грі з величезною кількістю кроків, так як наша основна мета в результаті - знайти різницю між значеннями вагів, а різниця – це відносне число, то і для її знаходження можна використовувати відносні ваги.

Знайшовши корені системи, отримуємо відносні від $v_{N-1}$ значення $v_0, v_1 \dots v_{N-2}$ та c. В наступному підпункті буде написано про те, як ці значення можна використовувати для пошуку більш оптимальних стратегій.

### 1.3 Покращення стратегії

В попередньому пункті для обраної раніше стратегії було показано метод знаходження відносних вагів та ціни гри. Тепер за їх допомогою з'являється змога знайти стратегію, яка принесе для однією з коаліцій найкращий результат.

Позначимо прибуток першої коаліції в стані $i$, при вибраних стратегіях $k_p^1$ і $k_m^2$ як $q_i^{pm}$. Також при аналогічних стратегіях для позначимо $p_{ij}^{pm}$ – ймовірність переходу зі стану $i$ в стан $j$.

Припустимо, необхідно обрати оптимальну стратегію для першої коаліції. Нехай ми маємо фіксовану стратегію $f_2$ для другої коаліції, а стани та дії гравців дискретні. Для початку потрібно визначити значення, яке потрібно буде максимізувати для отримання оптимальної стратегії для першої коаліції в такому випадку. За допомогою рівняння Беллмана [3] матимемо такий вираз:

$$v_i(n+1) = \max_{k_p^1 \in K_1} (q_i^{k_p^1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{k_p^1 f_2} v_j(n))$$

Відповідно, маємо обрати таку стратегію, яка максимізує значення $q_i^{k_p^1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{k_p^1 f_2} v_j(n)$.

Розкладемо $v_j(n)$, матимемо:

$$q_i^{k_p^1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{k_p^1 f_2} (nc + v_j)$$

Так як $\sum_{j=0}^{N-1} p_{ij}^{k_p^1 f_2} = 1$, а nc та константа, яку ми додавали до всіх значень вектору $v$ не залежать від $k_p^1$, то можемо скористатися тими вагами, які ми знаходили на етапі знаходження вагів. Отримуємо вираз, який потрібно буде максимізувати для пошуку оптимальної стратегії для першої коаліції, а саме:

$$q_i^{k_p^1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{k_p^1 f_2} v_j$$

Все навпаки для другої коаліції. Якщо $f_1$ - фіксована стратегія першої коаліції, то, так як другій коаліції треба мінімізувати отримання прибутку першою, отримуємо:

$$v_i(n+1) = \min_{k_m^2 \in K_2} (q_i^{f_1 k_m^2} + \sum_{j=0}^{N-1} p_{ij}^{f_1 k_m^2} v_j(n))$$

Використовуючи подібні судження та скорочення, отримуємо, що для другої коаліції потрібно обрати таку стратегію, що мінімізує значення $q_i^{f_1 k_m^2} + \sum_{j=0}^{N-1} p_{ij}^{f_1 k_m^2} v_j$.

## 1.4 Ітераційний цикл

Для того, щоб знайти оптимальне рішення, етапи знаходження вагів та покращення стратегії потрібно виконувати один за одним. Немає різниці з якого з етапів починати.

Якщо починати з етапу знаходження вагів, потрібно обрати початкову стратегію для тієї коаліції, стратегію для якої ми шукаємо. Якщо починати з етапу покращення стратегії, потрібно обрати початкові значення вагів, наприклад, виставивши їх всі як нулі.

Після етапу знаходження вагів, виконується етап покращення стратегії, використовуючи ті ваги, які були знайдені на попередньому етапі. Після етапу покращення стратегії, виконується етап знаходження вагів, під час якого ми використовуємо ту стратегію, яку знайшли на попередньому етапі.

Цикл повинен припинити свою роботу тоді, коли дві однакові стратегії вибрані послідовно, що говорить про те, що зміна стратегії в такому випадку, зробить лише гірше і сенсу її міняти немає.

Варто зазначити, що на етапі покращення стратегії може виникнути ситуація, коли дві або більше стратегій видають однаково найкраще значення. В такому випадку, якщо стратегія, що була обрана на попередньому етапі покращення рішень теж присутня в цьому списку, ми обираємо її. Так робиться тому, що

інакше алгоритм може назавжди застрягти в двох стратегіях і ніколи не закінчитися. Якщо ж при такій ситуації попередня стратегія не входить в цей список, можна вибрати будь яку з найкращих стратегій.

Таким чином, після завершення роботи алгоритму, отримуємо стратегію, яка максимізує значення ціни гри, якщо ми обирали для першої коаліції, або мінімізує це значення, якщо ми обирали для другої.

## 1.5 Доведення ефективності алгоритму

Вищеописаний алгоритм знаходить оптимальні стратегії для довільної стохастичної коаліційної гри з локальною структурою взаємодії. Доведемо це твердження.

Позначимо $c^{PM}$ – ціна гри при вибраних стратегіях $P$ і $M$ для першої та другої коаліцій відповідно.

Нехай ціль алгоритму – обрати оптимальну стратегію для першої коаліції. Фіксованою стратегією другої коаліції нехай буде $f_2$. Припустимо на етап покращення стратегії система увійшла зі стратегією A, а після покращення повернулася стратегія B, що відрізняється від стратегії A. Ціль даного етапу, для першої коаліції, максимізувати значення $c$, тому для доведення правильності роботи потрібно переконатися, що $c^{Bf_2} \geq c^{Af_2}$.

З принципу роботи алгоритму зрозуміло, що:

$$q_i^{Bf_2} + \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^A \geq q_i^{Af_2} + \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A$$

Позначимо їх різницю як Δ:

$$\Delta_i = q_i^{Bf_2} + \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^A - q_i^{Af_2} - \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A \qquad (1.2)$$

Використовуючи формулу (1.1) матимемо:

$$\mathrm{c}^A + v_i^A = q_i^{Af_2} + \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A$$

$$\mathrm{c}^B + v_i^B = q_i^{Bf_2} + \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^B$$

Позначимо $c^B - c^A = c^\Delta$ і $v_i^B - v_i^B = v_i^\Delta$, віднімемо ці два рівняння і отримаємо:

$$c^\Delta + v_i^\Delta = q_i^{Bf_2} + \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^B - q_i^{Af_2} - \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A$$

Підставимо $q_i^{Bf_2} - q_i^{Af_2}$ з рівняння (1.2), матимемо:

$$c^\Delta + v_i^\Delta = \Delta_i - \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^A + \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A + \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^B - \sum_{j=0}^{N-1} p_{ij}^{Af_2} v_j^A$$

$$c^\Delta + v_i^\Delta = \Delta_i - \sum_{j=0}^{N-1} p_{ij}^{Bf_2} v_j^\Delta$$

Це рівняння по формі співпадає з рівнянням (1.1), з тією відмінністю, що тут використовуються відносні а не абсолютні величини. Маючи формулу:

$$c = \sum_{i=0}^{N-1} \pi_i q_i$$

Можемо застосувати її для $\mathrm{c}^\Delta$, отримаємо:

$$c^\Delta = \sum_{i=0}^{N-1} \pi_i^B \Delta_i$$

Де $\pi_i^B$ – ймовірність опинитися в стані $i$ на кроці $n$, де $n \to \infty$.

Загалом, так як всі значення $\pi_i^B$ та $\Delta_i$ більші або рівні нулю, то можна стверджувати, що $c^\Delta \geq 0$. Так як $c^\Delta$ - різниця між ціною гри при виборі

стратегій B та A, справедливим є твердження, що обрана після етапу покращення стратегії, стратегія B буде, як мінімум, збільшувати значення $c$, що і потрібно було довести.

Але те, що алгоритм не вибере гіршої стратегії ніж була не значить, що він обере найкращу. Доведемо, що неможливий варіант того, що при виборі стратегій, алгоритм зупиниться на такій, яка не максимізує ціну гри.

Нехай на етапі покращення рішення була вибрана стратегія A, хоча існує стратегія B, для якої $c^B > c^A$. Тоді для всіх станів $\Delta_i \leq 0$. Так як при всіх значеннях $i$, матимемо $\pi_i^B \geq 0$, то використовуючи рівняння:

$$c^{\Delta} = \sum_{i=0}^{N-1} \pi_i^B \Delta_i$$

Бачимо, що $c^B - c^A \leq 0$, що суперечить заданій на початку умові. Отже, це доводить, що алгоритм завжди обере стратегію, що максимізує значення с.

Це були доведення ітераційного методу для першої коаліції. Для другої коаліції матимемо аналогічне доведення, з тією лише відмінністю, що друга коаліція мінімізує значення ціни гри, тому, всі знаки в доведенні необхідно буде змінити на протилежні. В інших аспектах доведення співпадає, тому не бачу сенсу на ньому зупинятися.

### 1.6 Ітераційний метод знаходження оптимальних стратегій для коаліцій

За допомогою алгоритму, який був описаний в пунктах вище, маємо змогу отримати стратегію для однієї з коаліцій, яка буде оптимальною при грі проти конкретної фіксованої стратегії, обраної іншою коаліцією. Але оптимальна стратегія проти одної стратегії може бути програшно. Тому потрібно вирішити проблему того, яку саме стратегію фіксувати для іншої коаліції.

Щоб вирішити дану проблему скористаємося методом поступових наближень. Для цього на початку для другої коаліції зафіксуємо певну випадкову стратегію (або, наприклад, стратегію, при якій всі гравці при будь якій ситуації обиратимуть нульову дію). Після цього виберемо оптимальну стратегію для першої коаліції та фіксуємо вже її. Потім вибираємо стратегію для другої коаліції, і так далі.

Пошук повинен закінчуватися тоді, коли під час двох повторних ітерацій отримуватимемо одні й ті ж самі стратегії для обох коаліцій. Це буде означати що зміна стратегії для будь-якої з коаліцій зробить для неї лише гірше, адже якщо б це було не так, то при знаходженні такої системи, на наступному кроці ми б отримували саме ту стратегію, яка краще підходить проти вибраної.

Таким чином отримуємо алгоритм, що знаходить оптимальні стратегії для обох коаліцій.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ДЛЯ ЗНАХОДЖЕННЯ ОПТИМАЛЬНИХ СТАРАТЕГІЙ

## 2.1 Визначення базових структур для гри

Програму було вирішено писати на мові програмування C#. Програма буде консольною, початкові дані вводитимуться в коді, так як початкових даних багато, кожен раз вводити їх в консолі доволі проблематично.

Для початку потрібно створити базові структури, за допомогою яких, будуть задаватися та зберігатися дані. Так як гра, яка буде моделюватися, може бути проілюстрована за допомогою графа, то вона містить в собі дві основні структури, а саме: вершина та ребро. Вершинами, очевидно, є гравці, а от з ребрами все не настільки однозначно. Ребра в грі позначають локальну структуру взаємодії і для роботи програми це значить те, що гравець А, який зв’язаний ребром з гравцем В, буде мати ймовірності переходу своїх станів в залежності від стану гравця В, та в кінці ходу виплачувати певну суму гравцю В. Гравець А може й нічого не виплачувати гравцю В, тоді вважатимемо, що він виплачує нульову суму грошей, незалежно від станів та дій. В програмі було вирішено зробити модель ребра лише для виплат, а зв’язаних гравців задавати в конструкторі вже самого гравця.

Модель гравця (Player) буде містити в собі:

- Ідентифікатор гравця (id), який буде використовуватися для ідентифікації гравця в колекціях та для спрощення його знаходження.
- Номер коаліції (team), який необхідний для розуміння, до якої коаліції належить гравець.
- Кількість можливих станів гравця (statesCount)
- Стан гравця (state), який, у випадку даної роботи, позначений цілим числом, що означає номер стану з дискретного списку станів.
- Масив цілих чисел, що позначає кількість можливих дій, в залежності від поточного стану гравця (actionsCountPerState)

- Масив індексів гравців, з якими зв’язаний гравець (neighbors). Ці дані будуть використовуватися лише для того щоб, при переході на наступний стан, алгоритм розумів від станів яких гравців залежить перехід.
- Стратегія гравця (strategy). Стратегія позначається словником. Словник – структура даних, яка присутня у багатьох мовах програмування в стандартних бібліотеках. Вона представляє собою колекцію типу ключ-значення, що означає що при створенні елементу колекції, дані задаються парами, а отримання значень виконується за допомогою задання ключа. У випадку стратегії, очевидно, ключом виступає стан гри, а значенням – дія, яку гравець буде виконувати, коли стан гри такий самий як ключ.
- Функція отримання ймовірностей переходу (translatePossibilitiesFunction). В C# є інструменти, які дозволяють задавати функції, як поля класу. Аргументами функції ймовірностей переходу для гравця A виступатимуть стан гравця A, стан гравців, які зв’язані з гравцем A та дія гравця A. Функція буде повертати ймовірності переходу як масив чисел з плаваючою крапкою. Сума чисел масиву, очевидно, повинна бути рівною одиниці.

Модель ребра (Edge) буде містити в собі:

- Посилання на гравців, яких зв’язує ребро (p1 та p2).
- Функцію (payAction), що повертає, яку суму повинен заплатити гравець p1 гравцю p2. Сума виплати залежить від стану гравця p1, стану гравця p2, дії гравця p1, та дії  гравця p2. Результатом виконання функції є ціле число.

За допомогою цих двох моделей можна задати початкові умови гри. Також ці моделі потрібні для симуляції самої гри.

Багато ігрових процесів залежить від стану, в якому перебуває гра, та загальної стратегії для коаліції та для всіх гравців. Стан гри, у випадку з дискретними станами гравців зручно задавати у двох виглядах:

- У вигляді цілого числа. Для того щоб перетворити стан всіх гравців на єдиний стан всієї гри у вигляді цілого числа, можна скористатися степенями десятки, тобто станом гри буде сума станів гравців, помножених на десять в степені кількість гравців мінус ідентифікатор мінус один. Наприклад маємо чотири гравця з ідентифікаторами 0. 1. 2. 3 зі станами 2, 3, 4, 5 відповідно, то станом гри буде число 2345. Звісно, такий спосіб не підходить, якщо в грі є більше дев'яти гравців і в першого гравця більше двох станів, так як найбільше ціле число, яке можна використати в C# - 2147483647. Також цей спосіб не спрацює, якщо в хоч одного з гравців буде більше 10 станів. Цей метод також не є дуже зручним з точки зору використання в коді, так як для взяття стану певного конкретного гравця, потрібно або перетворювати число в стрічку, після чого брати символ по ідентифікатору гравця і перетворювати його в число; або дістати число стану по формулі: число ділиться на десять в степені ідентифікатора, а потім береться остача від 10. Але цей метод має одну важливу перевагу – його дуже легко записувати як початкову умову, на відміну від створення колекцій. Саме через цю причину стани, подані числовими значеннями, і будуть використовуватися в програмі.
- У вигляді колекції. За допомогою словника, де ключ – гравець, а значення – стан, в якому цей гравець знаходиться. За допомогою подібної колекції матимемо дуже легкий доступ до станів будь-якого гравця, але записувати його доволі проблематично.

Для запису початкових умов буде використовуватися подання станів як цілих чисел, а для маніпуляцій зі станами і подальших дій подання як колекції.

Також для реалізації алгоритму знадобляться стратегії не тільки для одного гравця, а й для декількох гравців, записані однією колекцією. Такі колекції можна передавати в функції, що є доволі гнучким та простим способом використовувати стани й стратегії, відмінні від тих що наявні в моделях гравців гри, що беззаперечно зручно, коли є потреба дуже часто їх міняти. А Алгоритм

по пошуку оптимальних стратегій якраз потребує постійної заміни станів та стратегій.

Тому для подання стратегій для декількох гравців використаємо колекцію типу словник, де ключом виступатиме ідентифікатор гравця, а значенням – ту ж саму колекцію, що використовувалася для позначення стратегії гравця, тобто словник, ключом якого є стан гри, а значенням – дія гравця, яку він виконає, коли гра знаходитиметься у відповідному стані.

Для того, щоб в подальшому не писати для цих структур такі довгі описи, будемо позначати стан гри як State, а стратегію для всіх гравців як Strategy.

## 2.2 Задання умов та початкових значень для гри

Для ігор з дискретними станами та діями, задання початкових умов гри – доволі складний процес. Особливо це проявляється у заданні ймовірностей переходів. На певному довільному кроці, ймовірності переходу гравця залежить від стану гравця та його сусідів, і дії, яку він обрав. Тому, якщо в грі присутні два гравця, які мають по два стани та по дві дії в кожному зі станів, матимемо 2 (кількість гравців) * 2 (кількість станів гравця 1) * 2 (кількість станів гравця 2) * 2 (кількість можливих дій гравця) = 16 записів про ймовірності переходів, що відносно небагато. Але якщо ускладнити гру наприклад до 4 гравців, не змінюючи при цьому кількість можливих станів та дій у них, то по цій же формулі матимемо: $4 * 2 * 2 * 2 * 2 * 2 = 2^7 = 128$, що вже викликає значні проблеми, хоча і є однією з мінімальних конфігурацій, за допомогою якої можна нормально проілюструвати коаліційну гру.

Для зручності роботи алгоритму та допоміжних функцій, які будуть написані у майбутньому, було вирішено використовувати статичний клас Game, для зберігання даних про гравців та ребра, присутні в грі.

Тому для задання умов потрібно створити об’єкти типу Player, та додати їх до масиву всіх гравців, що присутній у класі Game. В конструкторі об’єкта Player потрібно задати всі базові параметри, а саме: ідентифікатор, коаліцію, кількість станів, початковий стан, масив ідентифікаторів зв’язаних гравців, масив кортежів зі значеннями стану гри та відповідних їм ймовірностей переходу та масив кількостей можливих дій для кожного стану гравця. Для задання об’єктів ребра, використовується конструктор що приймає двох гравців та функцію оплати, що залежить від дій та станів вищеназваних гравців.

## 2.3 Симуляція гри

Перед інтеграцією алгоритму пошуку оптимальних стратегій, потрібно створити систему, яка зможе симулювати гру. Потрібно це для перевірки роботи функцій, які будуть в подальшому написані та для перевірки того, що стратегії, які були вибрані за допомогою алгоритму реально показують найкращі результати. Це можливо перевірити, наприклад, запустивши симуляцію на велику кількість кроків, припустимо на 100000. Зрівнявши результати ігор двох довільних стратегій однієї коаліції і фіксованої стратегії другої, які ми отримаємо після таких симуляцій можна буде з великою ймовірністю визначити, яка з них краща. Результатом гри тут позначений загальний прибуток першої коаліції за весь проміжок гри.

Для початку має сенс окреслити, які етапи проходить гра на кожному кроці:

1. Спершу гравці, зважаючи на обрану стратегію та стан гри, обирають для себе одну з можливих дій.
2. Далі проходить етап оплати: в залежності від станів гравців та їх дій, зв’язані гравці виплачують один одному певну суму.
3. В кінці кроку гравці переходять в інші стани. Ймовірності переходу залежать від стану гравця, станів його сусідів та дії гравця.

Як видно з плану, для тестування симуляції знадобляться стратегії для гравців. Так як алгоритму для пошуку оптимальних стратегій ще немає, будуть використані тестові стратегії.

На першому етапі потрібно обрати для кожного гравця дію. Це доволі просто зробити, так як стратегія представляє з себе словник, просто візьмемо ту дію, яка знаходиться по ключу, що дорівнює нашому стану. Цей стан ми присвоюємо до відповідного поля вибраного гравця.

На другому етапі потрібно вирахувати суму виплати для кожного ребра та додати це значення до поточного результату гри. Для цього потрібно циклом пройтися по всім ребрам, та отримати у кожного з них результат виконання функції оплати, віддавши аргументами стани та дії гравців. Потім додаємо до значення результату гри суму всіх повернутих функціями значень.

На третьому етапі потрібно, за допомогою наданих в умові ймовірностей переходу, вирахувати в який стан переходить гравець. Для початку додамо гравцю поле nextState. В це поле буде присвоюватися результат переходу гравця. Це робиться для того, щоб не збити систему. Припустимо, маємо гру з двома гравцями зі станами (1, 1). Нехай в результаті переходу, перший гравець перейшов у стан 0 і програма одразу змінила його стан на це значення. Тоді, коли програма намагатиметься взяти стан гри для другого гравця, він буде дорівнювати вже не (1, 1), а (0, 1), що призведе до помилкового виконання алгоритму симуляції. Тому для початку змінюватиметься поле nextState, а після переходу всіх гравців в наступні стани значення із nextState присвоюватиметься стану гравця. Загалом, для виконання переходу знадобиться взяти випадкове число від нуля до одного, викликати функцію, яка за допомогою стану гравця, його сусідів та дії знайде ймовірності переходу в різні стани і, за допомогою раніше знайденого випадкового числа, обрати наступний стан гравця.

Таким чином маємо програму, яка може симулювати стохастичну коаліційну гру з локальною структурою взаємодії, зі заздалегідь вказаними стратегіями для всіх гравців, та повертати результат гри в кінці.

## 2.4 Інтеграція алгоритму пошуку оптимальних стратегій

Перед інтеграцією алгоритму потрібно для початку отримати дані, які знадобляться у майбутньому.

В алгоритмі використовуються значення відносних вагів $v_i$, де значення $i$ позначає відповідний стан системи. Так як ці значення повинні зберігатися впродовж алгоритму, потрібно зробити масив з їх значеннями. Для цього потрібно дізнатися кількість всіх можливих станів системи. Так як стан системи визначається станами гравців, для того щоб отримати всі можливі різні стани системи маємо перемножити кількість станів всіх гравців. Тобто, якщо в грі наявні 4 гравця з кількістю станів 2, 3, 4 і 1, то, відповідно, матимемо кількість станів рівну 2*3*4*1 = 24.

Також для алгоритму знадобиться список всіх станів гри. Для цього ініціюємо початковий стан, де всі гравці перебувають у нульовому стані і додамо до шуканого списку. Після цього, за допомогою циклу, потрібно пройтися по всіх гравцях. Для кожного з гравців ще одним циклом проходимося по всім його станам, відмінним від нуля. Після цього копіюємо всі стани, що вже перебувають у шуканому списку і, замість нульового стану, виставляємо для відповідного гравця поточний стан. Таким чином отримуємо список всіх можливих станів системи.

Додатково, впродовж роботи алгоритму потрібно буде отримувати всі можливі стратегії коаліції, для якої буде проводитися пошук оптимальної. В подальшому буде використовуватися колекція зі стратегіями для всіх гравців, де стратегії для гравців ворожої коаліції – фіксовані. Тоді логічно буде мати метод, який

повертає масив всіх можливих стратегій для вибраної коаліції з фіксованими стратегіями для іншої. Для цього скористаємося методом, подібним до того, за допомогою якого знаходили всі можливі стани гри. Створимо стратегію для всіх гравців, де для гравців вибраної коаліції для будь-яких станів обиратимемо нульову дію, а для гравців іншої коаліції обиратимемо раніше вибрані фіксовані стратегії. Потім потрібно пройтися циклом по всім гравцям обраної коаліції та доповнювати загальний список стратегій копіями цього списку зі зміненими значеннями стратегій для вибраних гравців. Всі можливі стратегії для одного гравця, які, очевидно, потрібні для знаходження всіх стратегій для всіх гравців, знаходяться таким самим методом, тому не бачу сенсу акцентувати на цьому увагу.

На даному етапі всі допоміжні методи готові до використання. Для початку потрібно визначитися зі сигнатурою для методу пошуку оптимальних стратегій. Очевидно, що цей метод повинен в результаті повертати обрану стратегію, але важливим для роботи алгоритму також буде отримати ціну гри, яку гра матиме, при використанні вибраною коаліцією даної стратегії. Тоді метод повинен повертати обидва цих значення і найбільш логічним рішенням буде реалізувати це за допомогою кортежу, який міститиме типи Strategy та decimal. Decimal було вирішено використовувати для чисел з плаваючою точкою тому, що його метод зберігання в пам’яті допомагає йому бути максимально точним, на відміну від типів float або double. Хоч похибка у останніх незначна, але значно краще взагалі її не мати, хоча, звісно, швидкість виконання операцій з типом decimal значно нижча, ніж з double або float.

Так як метод буде шукати алгоритм для однієї коаліції з фіксованою стратегією для другої, аргументами для нього буде логічно встановити номер коаліції, для якої буде вестися пошук, і фіксовану стратегію для ворожої коаліції. Загалом, не важливо, чи будуть в стратегії, яка буде передаватися як фіксована, наявні стратегії для гравців з вибраної коаліції, так як на протязі алгоритму ці стратегії

будуть просто ігноруватися. Ця властивість другого аргументу дуже допоможе в подальшому використанні даного методу.

На початку роботи методу потрібно визначити, що нам потрібно робити зі значеннями, отриманими за допомогою формули:

$$q_i^{f_1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{f_1 f_2} v_j$$

Якщо коаліцією, для якої здійснюється пошук є перша коаліція, потрібно обирати таку стратегію, яка максимізує дане значення, якщо ж для другої – мінімізує його. В такому випадку скористаємося механізмом мови програмування C#, а саме лямбда функціями. За допомогою цієї лямда функції будемо порівнювати два значення, і вона повертатиме результат один, якщо нове значення краще за найкраще минуле, нуль, якщо вони рівні, і мінус один, якщо нове значення гірше. Таким чином, на початку методу потрібно перевірити, для якої коаліції наразі проводиться пошук стратегії, і відповідним чином реалізувати лямбду better.

Наступним кроком потрібно отримати список всіх стратегій, для шуканої коаліції, за допомогою раніше написаного методу. Також, перед початком роботи ітераційного методу, потрібно додати масив значень v, довжиною з кількість станів гри, і створити змінні поточна стратегія та минула стратегія.

Починати ітераційний метод було вирішено з етапу покращення стратегії.

На етапі покращення стратегії створюється пустий лист з найкращими стратегіями і поточне значення, що набрала найкраща з уже випробуваних стратегій. Далі алгоритм починає проходитися циклом по списку стратегій, нехай поточну стратегію, значення для якої наразі шукає алгоритм, позначимо як S. Для стратегії S алгоритм проходиться по всім можливим станам системи, нехай поточний стан системи який алгоритм перевіряє для стратегії S, буде позначений як I. Тоді для стану I алгоритм додає значення $q_I^S$ до значення S.

Далі алгоритм знову проводить ітерацію по всім можливим станам гри, нехай позначаючи ці стани як J, і додає $p_{IJ}^{S} v_J$ до значення S. Таким чином, пройшовши двічі по всім станам, отримуємо фінальне значення стратегії S, яке потім за допомогою лямбди буде порівнюватися зі значеннями інших стратегій.

На даному етапі має сенс звернути увагу на можливе непорозуміння, чому в теоретичній частині для стратегій було вирішено порівнювати значення

$$q_i^{f_1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{f_1 f_2} v_j \tag{2.1}$$

А в алгоритмі наразі порівнюється значення

$$\sum_{i=0}^{N-1} q_i^{f_1 f_2} + \sum_{j=0}^{N-1} p_{ij}^{f_1 f_2} v_j \tag{2.2}$$

Ця заміна трапляється тому, що в теоретичній частині стратегії розглядалися окремо для кожного стану, а в даній програмній реалізації стратегія позначає собою дію для будь якого стану для гравця. Тому, стратегії, що використовуються в програмі, насправді включають у себе набір зі стратегій з теоретичної частини для всіх станів. Доведемо, що дані правила еквівалентні, і лише містять різні типи даних. Для цього розглянемо ситуацію вибору стратегії для першої коаліції, тобто дані значення потрібно буде максимізувати. Якщо для всіх станів системи вибрати такі дії, що максимізуватимуть значення (2.1), то і якщо зібрати ці значення в стратегію для кожного стану, значення (2.2) вийде максимальним. Тому формула (2.2) працює і не суперечить доведеній раніше теорії.

Знайшовши значення для стратегії S, якщо раніше вже знаходили значення для інших стратегій, порівнюємо це значення з найкращим, що було знайдено раніше. Якщо значення краще, то очищаєм лист кращих стратегій і додаємо туди стратегію S, позначивши при цьому її значення як найкраще. Якщо значення гірше – нічого не робимо. Якщо ж значення рівні – додаємо стратегію

S до листа. Стратегію, для якої знаходили значення першою просто додаємо до листа і позначаємо її значення як найкраще.

Далі, після аналізу всіх стратегій, позначаємо поточну стратегію як попередню. Якщо в листі найкращих стратегій присутні більше ніж одна стратегії, обираємо попередню, якщо вона там є, або, в інакшому випадку, першу по списку.

Якщо вибрана стратегія така ж як і попередня, припиняємо роботу функції та повертаємо вибрану стратегію та ціну гри. Очевидно, дана перевірка має сенс починаючи з другої ітерації, адже на першій немає ні попередньої стратегії, ні знайденої ціни гри на даному етапі ітерації.

Наступним етапом ітераційного методу є визначення вагів. Так як на цьому етапі потрібно вирішити систему лінійних рівнянь, для додавання даних будемо використовувати матрицю, де в кожному із стовпчиків буде визначено число, яке стоїть перед відповідною змінною рівняння. А для знаходження коренів системи рівнянь буде використаний метод Гауса.

Опишемо процес заповнення рівняння номер $i$ з системи. Рівняння, яке потрібно перенести в матрицю має вигляд (1.1). Першою змінною в кожному рівнянні виступає ціна гри, тобто $c$, тому перший стовпчик одразу можна заповнити одиницями. Далі додається одиниця до $i + 1$ змінної, при умові, що $i \neq N$, де $N$ стан, для якого $v_N = 0$. Після цього за допомогою циклу виконується прохід по всім станам, крім стану N, позначимо поточний стан в циклі як $j$. В циклі до комірки $[i, j + 1]$ додається ймовірність переходу із стану $i$ в стан $j$. В останню ж комірку кожного рядка вписуємо прибуток, який отримує перша коаліція в стані $i$.

Корені отриманої після цього матриці знаходимо за допомогою методу Гауса. Таким чином отримуємо масив вагів $v_i$ та ціну гри $c$, які будуть отримані гравцями при обраній раніше стратегії.

Нижче наведений програмний код реалізації даної функції:

```
public static (decimal,Strategy) FindStrategyForCoalition(int team, Strategy
anotherCoalitionStrategy)
{
   bool isMax = team == 1;
   Func<decimal, decimal, int> better;
   if (isMax)
      better = (a, b) => a > b ? 1 : (a == b ? 0 : -1);
   else
      better = (a, b) => a < b ? 1 : (a == b ? 0 : -1);

   var ss = GetStrategies(team, anotherCoalitionStrategy);

   decimal[] v = new decimal[vCount];
   for (int i = 0; i < vCount - 1; i++)
      v[i] = 0;
   decimal c = 0;

   int counter = 1;

   Strategy chosenStrategy = null;
   Strategy prevChosen = null;
   while (true)
   {
      //Policy improvement routine
      List<Strategy> bestStrategies = new List<Strategy>();
      decimal bestScore = -team*1000000;
      foreach (var s in ss)
      {
         decimal value = 0;
         for (int i = 0;i<vCount;i++)
```

```
            {
                decimal add = Stats.GetAllPlayersReward(states[i], s);
                value += add;
                for (int j = 0;j<vCount;j++)
                {
                    add = Stats.GetTransitionProbability(states[i], states[j], s);
                    value += add * v[j];
                }
            }
            int betterV = better(value, bestScore);
            if (betterV == 1)
            {
                bestScore = value;
                bestStrategies = new List<Strategy> { s };
            }
            else if (betterV == 0)
            {
                bestStrategies.Add(s);
            }
        }
        prevChosen = chosenStrategy;
        if (bestStrategies.Count == 1 || chosenStrategy == null ||
!bestStrategies.Contains(chosenStrategy))
            chosenStrategy = bestStrategies[0];

        //Value determination operation
        decimal[,] matrix = new decimal[vCount,vCount+1];
        for (int i = 0;i<vCount;i++)
        {
            matrix[i,0] = 1;
```

```
            for (int j = 0;j<vCount-1;j++)
            {
                matrix[i, j+1] = -Stats.GetTransitionProbability(states[i], states[j],
chosenStrategy);
                if (i == j)
                    matrix[i, j+1] += 1m;
            }
            matrix[i, vCount] = Stats.GetAllPlayersReward(states[i], chosenStrategy);
        }
        if (Helpers.SolveMatrix(matrix, out decimal[] results))
        {
            c = results[0];
            for (int i = 0; i< vCount-1 ; i++)
            {
                v[i] = results[i+1];
            }
        }
        else
        {
            throw new Exception("Error while solving matrix...");
        }
        if (prevChosen == chosenStrategy)
            break;
    }

    return (c,chosenStrategy);
}
```

Після реалізації даного алгоритму залишається лише реалізувати пошук стратегій для обох коаліцій. Для цього створимо функцію

FindStrategiesForBothCoalitions. В цій функції за допомогою циклу буде змінюватися стратегії обох коаліцій на оптимальні по черзі, використовуючи при цьому минулу стратегію, знайдену для ворожої коаліції. Нижче приведений код, що реалізує дану функцію:

```
public static Strategy FindStrategiesForBothCoalitions()
{
   Strategy s1 = GetStartStrategy();
   Strategy prev = null;
   Strategy current = s1;
   decimal c1, c2;
   int iter = 1;
   do
   {
      prev = current;
      (c1, current) = FindStrategyForCoalition(1, current);
      (c2, current) = FindStrategyForCoalition(-1, current);
      Console.WriteLine($"Iteration {iter++}");
      Console.WriteLine($"c = {c2}");
      Console.WriteLine($"strategy = {current}");
   } while (!current.Equals(prev) || c1 != c2);
   return s1;
}
```

Таким чином маємо повністю реалізований алгоритм, що був описаний у першому розділі. У наступному розділі, за допомогою написаної програми, буде проведено аналіз ігор з різними початковими умовами та те, як змінить зміна умов гри результати вибору оптимальних стратегій та ціну гри.

# РОЗДІЛ 3. АНАЛІЗ СТОХАСТИЧНИХ КОАЛІЦІЙНИХ ІГОР З ЛОКАЛЬНОЮ СТРУКТУРОЮ ВЗАЄМОДІЇ

## 3.1 Аналіз гри з двома гравцями

Для початку аналізу ігор потрібно змоделювати їх початкові умови. Як вже раніше було зазначено, для стохастичних ігор із дискретними станами доволі складно описати початкові умови, так як маємо величезну кількість можливих станів гри, для кожного з яких і для кожного з гравців потрібно описати ймовірності переходу, в залежності від дії гравця. Тому для початку було прийнято рішення змоделювати гру з двома гравцями, де, очевидно, вони будуть з різних коаліцій. Також в кожного гравця буде по два можливих стани, в кожному з яких гравці зможуть обирати одну з двох можливих дій. Таку гру буде максимально легко аналізувати, та зрозуміти, звідки були отримані ті чи інші результати.

Загалом, природа гри не має значення для моделі, так як алгоритм буде працювати незалежно від того, яку саме проблему за допомогою нього вирішують. Але для кращого розуміння та більш простої інтерпретації результатів, далі буде описано що саме визначають гравці, їх стани та дії.

Для першої гри використаємо стандартний для стохастичних ігор приклад, а саме галузь економіки. Нехай маємо гру з двома виробниками хлібних виробів у невеличкому містечку. Очевидно, що між ними буде постійна конкуренція, і кожен з них хоче отримувати найбільший прибуток. Якщо гравець знаходиться у першому стані - це значить що справи йдуть добре, якщо в другому – погано. Якщо в будь якому з двох станів гравець обрав першу дію, то він нічого не робить для реклами своїх товарів, якщо ж другу – він замовляє рекламу, що, очевидно, збільшує його шанси опинитися в кращому стані на наступний хід, але, разом з тим, коштує певну суму грошей.

Загалом граф гравців буде виглядати так:

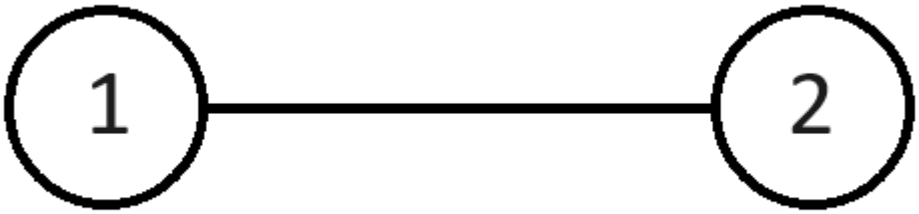


*Рисунок 3.1 Схема гри з двома гравцями*

Для початку визначимо ймовірності переходу гравців. Так як ці ймовірності залежать від станів гравця і сусідів, який у даному випадку один, та від дії гравця, в обох гравців таких пар ймовірностей буде вісім. Нижче за допомогою таблиці проілюстровані всі можливі випадки для першого гравця:

| Стан гравця 1 | Стан гравця 2 | Дія гравця 1 | Ймовірність перейти в стан 1 | Ймовірність перейти в стан 2 |
|---|---|---|---|---|
| 1 | 1 | 1 | 0.3 | 0.7 |
| 1 | 1 | 2 | 0.6 | 0.4 |
| 1 | 2 | 1 | 0.5 | 0.5 |
| 1 | 2 | 2 | 0.9 | 0.1 |
| 2 | 1 | 1 | 0.1 | 0.9 |
| 2 | 1 | 2 | 0.4 | 0.6 |
| 2 | 2 | 1 | 0.2 | 0.8 |
| 2 | 2 | 2 | 0.6 | 0.4 |

Для другого гравця будуть використовуватися такі ж ймовірності переходів як і для першого, з тією лиш відмінністю, що в таблиці потрібно змінити значення станів першого та другого гравців.

Також потрібно визначити яку суму гравці будуть виплачувати один одному в залежності від їх станів та дій. Загалом немає сенсу описувати конфігурацію виплат для обох гравців, так як цей обмін грошима можна описати зі сторони одного гравця, наприклад першого. В ситуаціях, коли другий гравець платить

першому, ця сума буде додатньою, якщо ж інакше – від'ємною. Функцію виплат не будемо записувати у формі таблиці, так як в такому випадку вийде занадто багато рядків, що буде важко сприймати. Легше буде записати відповідні умови за допомогою списку правил, де сумою виплати виступатиме те, скільки перший гравець заплатить другому, і на початку рівна нулю:

- Якщо гравець один знаходиться в першому стані, а гравець два - в другому, сума виплати зменшиться на $p_1$
- Якщо гравець один знаходиться в другому стані, а гравець два - в першому, сума виплати збільшиться на $p_2$
- Якщо гравець один обрав дію два, сума виплати збільшиться на $p_3$
- Якщо гравець два обрав дію два, сума виплати зменшиться на $p_4$

$p_1, p_2, p_3$ і $p_4$ – змінні, змінюючи значення яких і буде проводитися аналіз. Якщо говорити менш формально, матимемо:

- $p_1$ – сума грошей, яку заплатить другий гравець першому, якщо матиме гірший стан
- $p_2$ – сума грошей, яку заплатить перший гравець другому, якщо матиме гірший стан
- $p_3$ – сума грошей, яку перший гравець повинен заплатити за рекламу
- $p_4$ – сума грошей, яку другий гравець повинен заплатити за рекламу

Для початку варто зазначити, що ймовірності переходу, які були визначені для гри, є однаковими для обох гравців відносно один одного. Тому цікаво було б проаналізувати, якою буде ціна гри та вибрані стратегії, якщо встановити такі ж однакові виплати для обох гравців, тобто $p_1 = p_2$ і $p_3 = p_4$.

Для початку встановимо $p_1 = p_2 = 10$ і $p_3 = p_4 = 3$, що означає що обидва гравця платять один одному по 10 умовних одиниць, а за рекламу – по 3. Отримаємо такі результати:

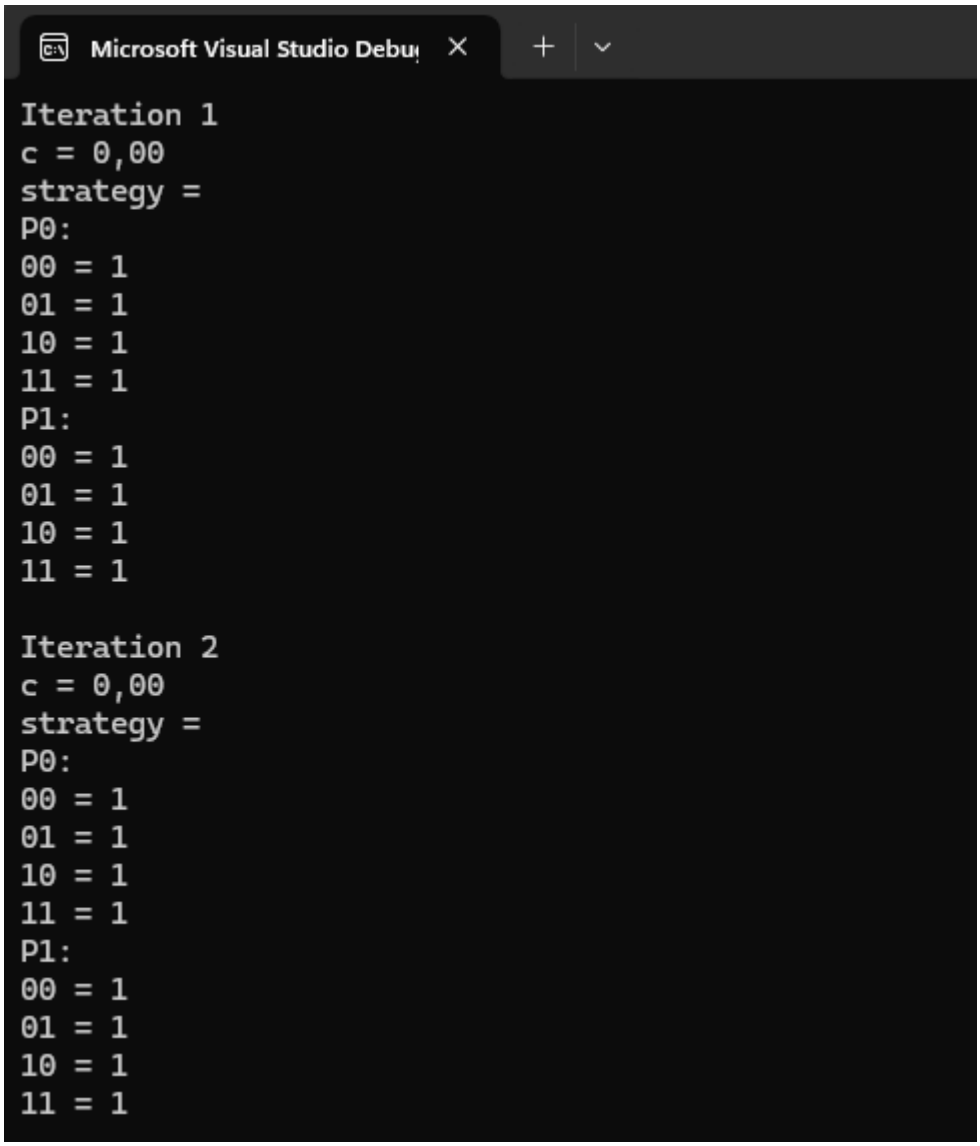


*Рисунок 3.2 Результати виконання програми*

Це означає що алгоритм завершив своє виконання на другій ітерації, повернувши значення ціни гри $c = 0$, та оптимальні стратегії для гравців, які для кожного стану обирають замовлення реклами (всі числа в стратегіях тут потрібно збільшувати на одиницю, так як в програмі перший стан або дія позначені як нуль). Те, що ціна гри рівна нулю, є очікуваним, так як при однакових умовах для гравців, та однакових обраних стратегіях логічно, що в середньому за хід ніхто не отримуватиме перевагу. Вибір однакових стратегій при однакових умовах також є оптимальним рішенням, так як якщо б існувала стратегія краще, її б обрали обидва учасника.

Для другого експерименту змінимо $p_3 = p_4 = 10$, що, очевидно, зробить замовлення реклами набагато менш вигідним рішенням. Отримаємо такі результати:

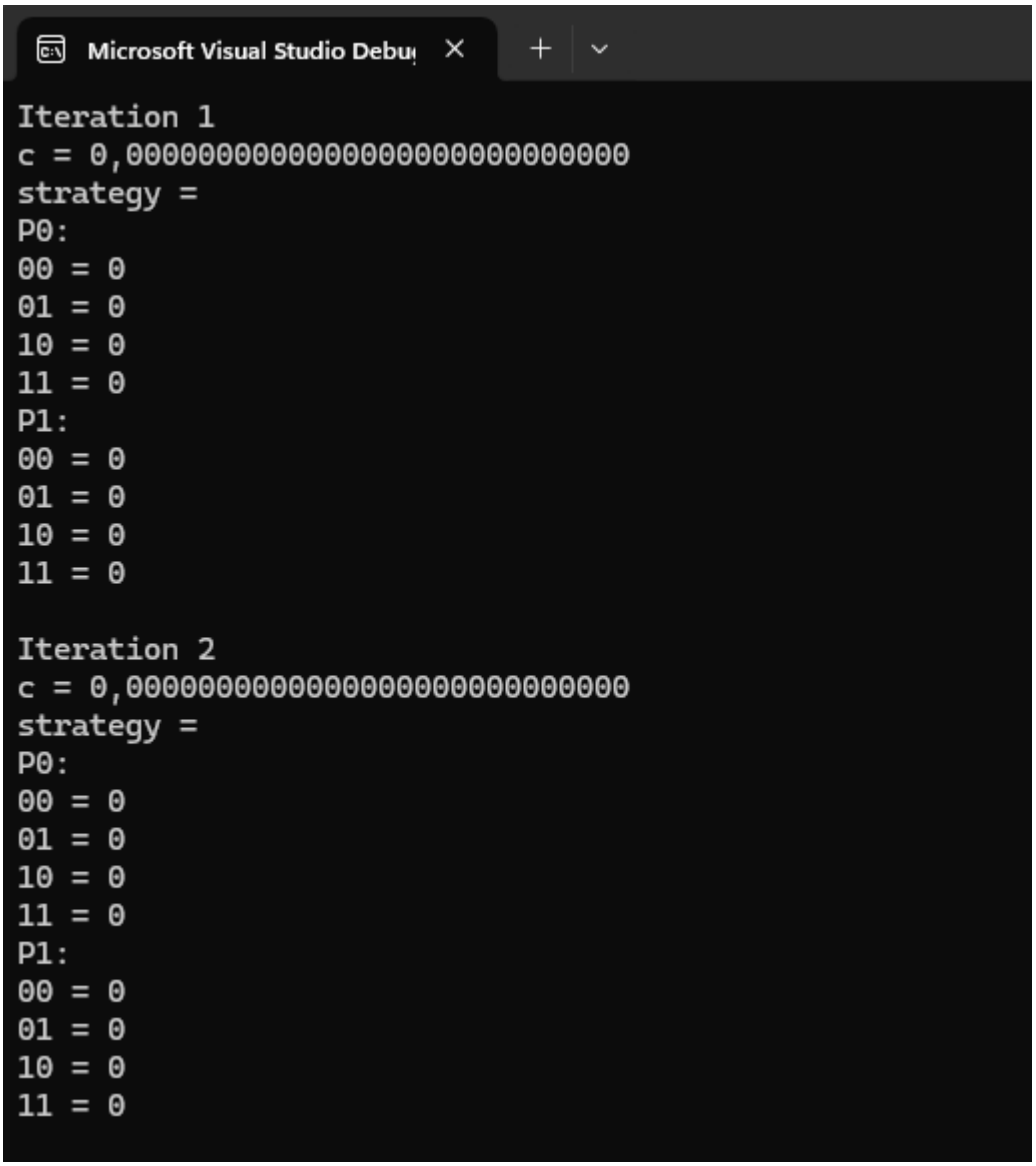


*Рисунок 3.3 Результати виконання програми*

Як видно з результатів, тепер гравці ні в якому разі не купляють рекламу, але при цьому ціна гри так само лишається нулем, так як умови все ще однакові.

Спробуємо при такій самій ціні реклами змінити ймовірності переходу в першого гравця при станах 2 і 1 для першого та другого гравця відповідно, і при замовленні реклами з [0.4, 0.6] на [0.9, 0.1]. Тепер в даному стані для першого гравця стало вигідніше замовляти рекламу, і результат тепер виглядатиме так:

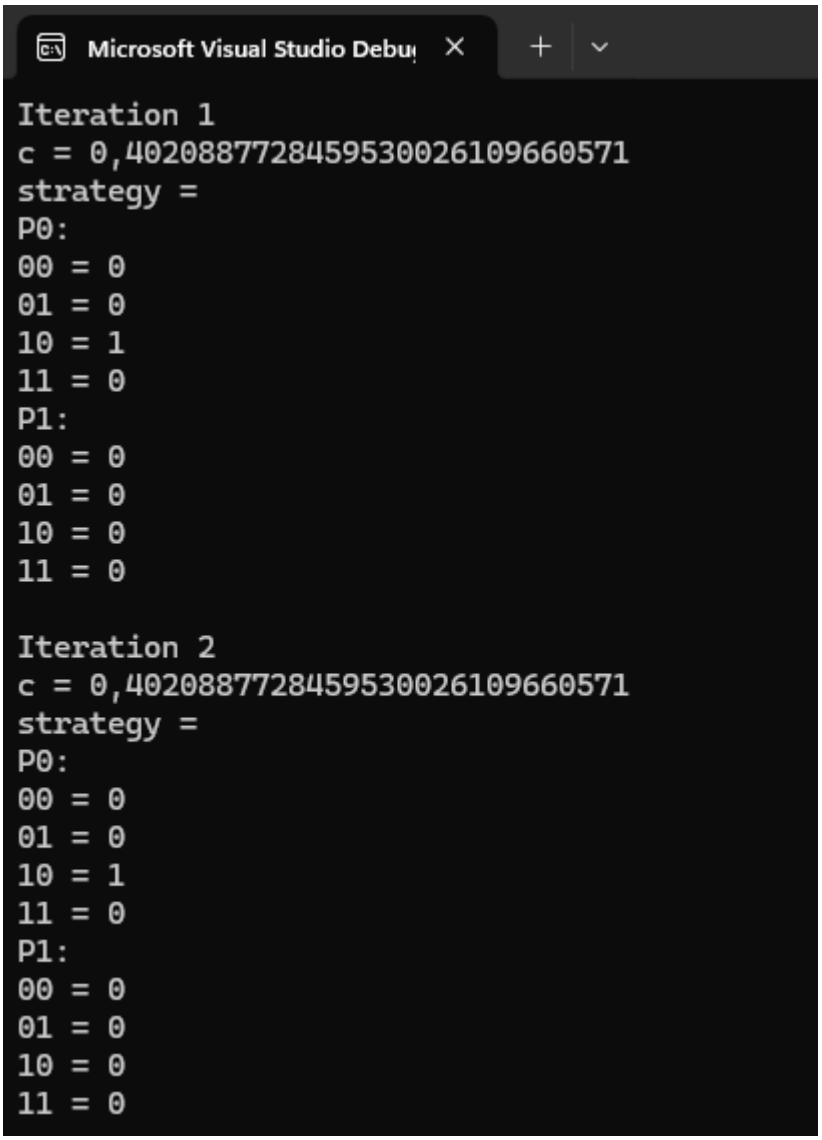


*Рисунок 3.4 Результати виконання програми*

Як видно з результату, такого збільшення ймовірності переходу в кращий стан вистачило для того щоб покрити велику ціну за рекламу. Також видно що так як умови різні маємо позитивну ціну гри, що означає що в середньому за крок перший гравець буде отримувати на $\approx 0.4021$ суму більше ніж другий.

Повернемо ймовірності переходів до заданих на початку розділу, а ціни за рекламу визначимо як $p_3 = 2$ і $p_4 = 1$, тобто другий гравець платить за рекламу на одиницю менше. Матимемо:

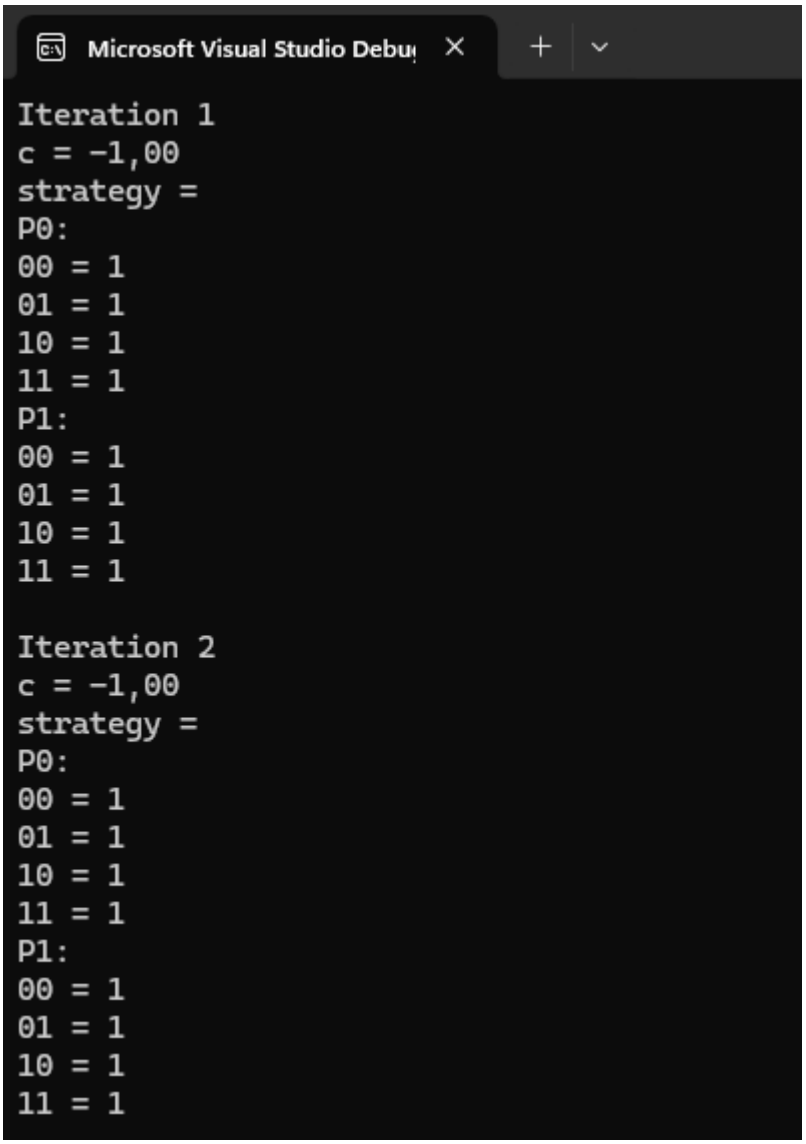


*Рисунок 3.5 Результати виконання програми*

Отримані результати також цілком передбачувані. Так як покупка реклами для обох гравців коштує доволі дешево, то обидва гравця на кожному кроці будуть її замовляти, а так як перший гравець платить за неї на одиницю більше, матимемо що і середній його прибуток за хід буде на одиницю менше, про що й говорить значення ціни гри в мінус одиницю. Така ж тенденція зберігається до ціни реклами для першого гравця в 5 одиниць. При $p_3 = 6$, матимемо такий результат:

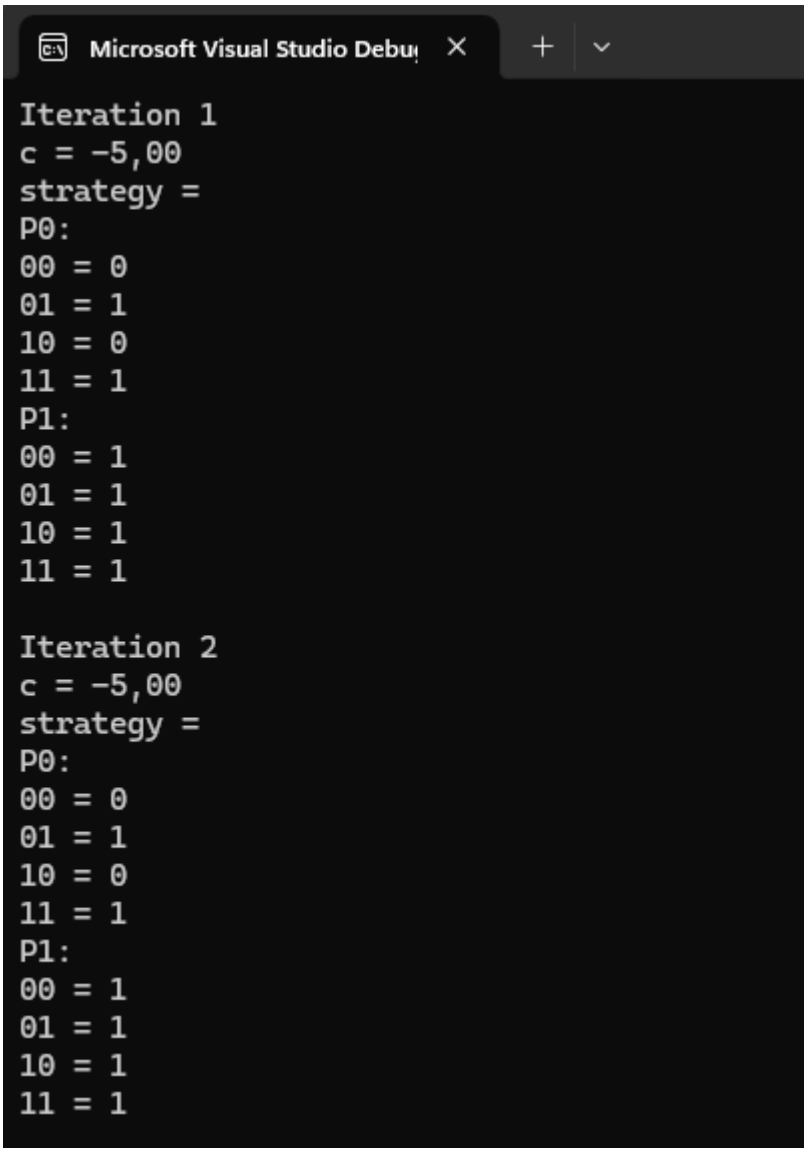


*Рисунок 3.6 Результати виконання програми*

Бачимо, що перший гравець не замовляє реклами при двох з можливих станів гри, але ціна гри лишається такою ж, як та, при якій він би замовляв рекламу для всіх станів, що є доволі цікавим збігом. А от вже при $p_3 = 7$, отримаємо:

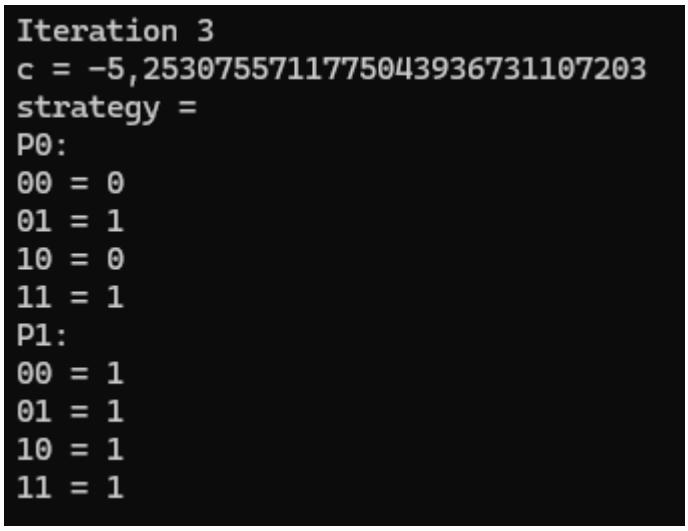


*Рисунок 3.7 Результати виконання програми*

Бачимо, що гравець хоч і обирає ті ж самі стани, але ціна гри на цей раз вже не співпадає з такою, яка була б якщо він обирав би завжди замовляти рекламу.

Таким чином на основі гри з двома гравцями вдалося проаналізувати як початкові умови та їх зміна впливають на вибір гравцями оптимальних стратегій.

## 3.2 Аналіз гри з чотирма гравцями

Як раніше було зазначено, складність роботи алгоритму і, що очевидно, його аналізу росте в геометричній прогресії від кількості гравців, тому аналіз гри з чотирма гравцями буде не настільки детальним.

Визначимо, що гравці 1 і 3 грають за першу коаліцію, а 2 і 4 – за другу. Галуззю інтерпретації цієї гри так само оберемо економічну. Нехай гравці 1 і 2 – виробники мікропроцесорів, які конкурують між собою, а 3 і 4 – виробники смартфонів, які також конкурують. Гравець 1 постачає мікропроцесори гравцю 3, а гравець 2 – гравцю 4. Стани та дії для гравців 3 і 4 такі ж, як і в минулому прикладі, а саме:

- Стан 1 – продажі йдуть добре

- Стан 2 – продажі йдуть погано
- Дія 1 – Нічого не робити
- Дія 2 – Замовити рекламу

Також для цих гравців залишимо формулу виплат один одному з минулої гри і змінні $p_1, \dots, p_4$, де в їх визначеннях гравець 1 просто змінюється на гравця 3, а гравець 2 – на гравця 4.

Для гравців 1 та 2 буде також по 2 стана та 2 дії:

- Стан 1 – якість процесорів хороша, продажі йдуть добре
- Стан 2 – якість процесорів погана, продажі йдуть погано
- Дія 1 – Продовжувати виготовлення процесорів за допомогою старих технологій
- Дія 2 – закупити нові технології виготовлення процесорів

Таким чином маємо 2 схожих по суті пари гравців, що полегшить розуміння результатів.

Загалом граф гри виглядатиме так:

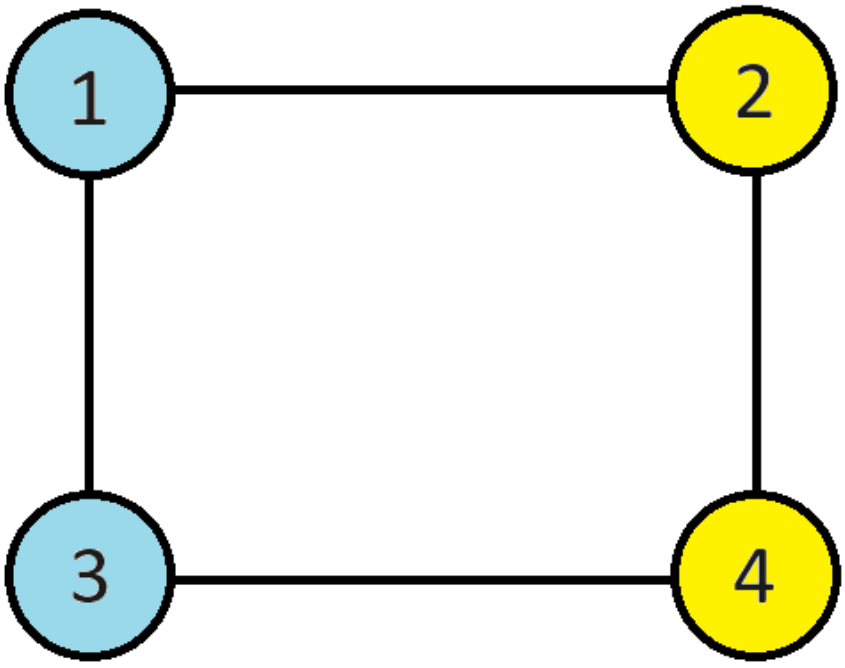


*Рисунок 3.8 Схема гри з чотирма гравцями*

Тепер потрібно визначити ймовірності переходів. Щоб не ускладнювати аналіз знову для початку створимо однакові умови для гравців. Так для початку визначимо ймовірності переходів для гравців 1 та 2:

| Стан гравця | Стан сусіда-ворога | Дія гравця | Ймовірність перейти в стан 1 | Ймовірність перейти в стан 2 |
|---|---|---|---|---|
| 1 | 1 | 1 | 0.4 | 0.6 |
| 1 | 1 | 2 | 0.6 | 0.4 |
| 1 | 2 | 1 | 0.5 | 0.5 |
| 1 | 2 | 2 | 0.7 | 0.3 |
| 2 | 1 | 1 | 0.3 | 0.7 |
| 2 | 1 | 2 | 0.4 | 0.6 |
| 2 | 2 | 1 | 0.4 | 0.6 |
| 2 | 2 | 2 | 0.5 | 0.5 |

Стани третього та четвертого гравців з коаліції не впливають на ймовірності, тому і не були додані в таблицю. Сусідом-ворогом тут для 1 гравця позначений гравець 2, а для другого – гравець 1.

Позначимо ймовірність переходів для гравця 3:

| Стан гравця 3 | Стан гравця 1 | Стан гравця 4 | Дія гравця 3 | Ймовірність перейти в стан 1 | Ймовірність перейти в стан 2 |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0.5 | 0.5 |
| 1 | 1 | 1 | 2 | 0.7 | 0.3 |
| 1 | 1 | 2 | 1 | 0.7 | 0.3 |
| 1 | 1 | 2 | 2 | 0.9 | 0.1 |
| 1 | 2 | 1 | 1 | 0.4 | 0.6 |
| 1 | 2 | 1 | 2 | 0.6 | 0.4 |
| 1 | 2 | 2 | 1 | 0.6 | 0.4 |

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 2 | 2 | 0.8 | 0.2 |
| 2 | 1 | 1 | 1 | 0.1 | 0.9 |
| 2 | 1 | 1 | 2 | 0.3 | 0.7 |
| 2 | 1 | 2 | 1 | 0.3 | 0.7 |
| 2 | 1 | 2 | 2 | 0.5 | 0.5 |
| 2 | 2 | 1 | 1 | 0 | 1 |
| 2 | 2 | 1 | 2 | 0.2 | 0.8 |
| 2 | 2 | 2 | 1 | 0.2 | 0.8 |
| 2 | 2 | 2 | 2 | 0.4 | 0.6 |

Для гравця 4 будемо використовувати такі ж ймовірності, з тієї лише відмінністю, що першими чотирма колонками будуть: стан гравця 4, стан гравця 2, стан гравця 3 та дія гравця 4.

Для гравців 3 і 4 зазначимо такі правила виплат:

- Якщо гравець 3 знаходиться в першому стані, а гравець 4 - в другому, сума виплати зменшиться на $d_1$
- Якщо гравець 3 знаходиться в другому стані, а гравець 4 - в першому, сума виплати збільшиться на $d_2$
- Якщо гравець 3 обрав дію два, сума виплати збільшиться на $d_3$
- Якщо гравець 4 обрав дію два, сума виплати зменшиться на $\mathrm{d}_4$

Почавши аналіз даної гри стало зрозуміло, що даний алгоритм не зможе впоратися з даним завданням так як число можливих стратегій для будь якої з двох коаліцій дорівнює $2^{32} = 4\,294\,967\,296$. Таке число можливих стратегій неможливо зберігати не в одній структурі і середовищі виконання виникає помилка нестачі пам’яті.

### 3.3 Аналіз гри з трьома гравцями

Аби все таки провести аналіз коаліційної гри, було прийнято рішення видалити одного з гравців станів, а саме четвертий. Таким чином отримуємо гру з трьома гравцями, два в першій коаліції проти одного в другій. Матимемо граф:

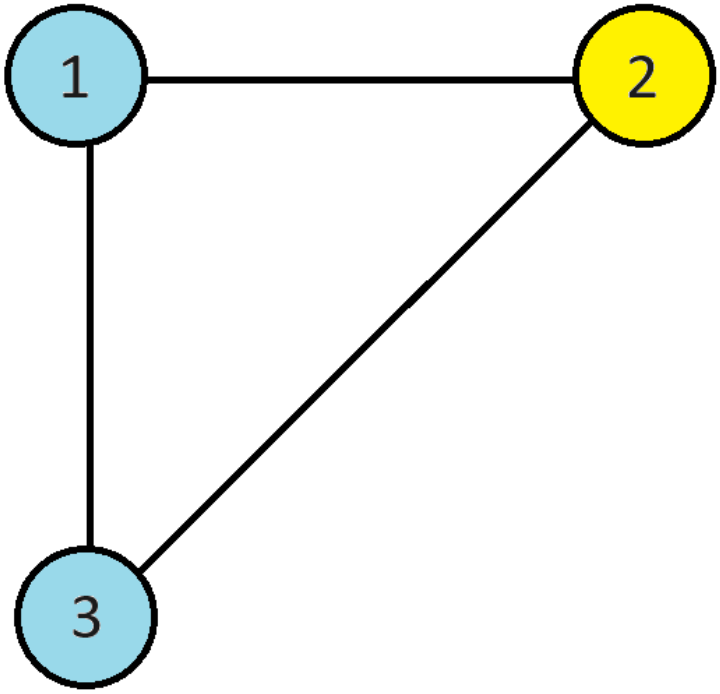


*Рисунок 3.9 Схема гри з трьома гравцями*

Будемо використовувати такі ж таблиці переходу ймовірностей, як були наведені для гри з чотирма гравцями, з тією лише відмінністю, що для ймовірностей переходу третього гравця замість стану гравця 4 буде враховуватися стан гравця 2. Також правила оплати для 3 і 4 гравця повністю переносяться на 3 і 2 відповідно.

Встановимо $p_1 = d_1 = p_2 = d_2 = 10$ і $p_3 = d_3 = p_4 = d_4 = 1$. Матимемо такий результат:

```
Iteration 2
c = 0,18518518518518518518518609990
strategy =
P0:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
P1:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
P2:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
```

*Рисунок 3.10 Результати виконання програми*

З результатів можна побачити, що всі гравці для всіх станів обирають придбати рекламу/технології. Бачимо, що перша коаліція має невелику перевагу.

Спробуємо виставити $p_4 = d_4 = 0$. Отримаємо такі результати:

```
Iteration 2
c = -1,8148148148148148148148139010
strategy =
P0:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
P1:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
P2:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
```

*Рисунок 3.11 Результати виконання програми*

Очевидно, так як другий гравець і так закупляв рекламу по ціні в одиницю він не перестане це робити якщо вона подешевшає, тому стратегія не змінилася. Також бачимо, що ціна гри знизилася на 2 одиниці, що логічно, так як другий гравець замовляє рекламу що так що так, але йому доводиться платити на 2 одиниці менше за крок. Спробуємо встановити $p_3 = d_3 = p_4 = d_4 = 5$. Матимемо:

```
Iteration 2
c = -1,33333333333333333333333333414
strategy =
P0:
000 = 0
001 = 0
010 = 0
011 = 0
100 = 0
101 = 0
110 = 0
111 = 0
P1:
000 = 0
001 = 0
010 = 0
011 = 0
100 = 0
101 = 0
110 = 0
111 = 0
P2:
000 = 0
001 = 0
010 = 0
011 = 0
100 = 0
101 = 0
110 = 0
111 = 0
```

*Рисунок 3.12 Результати виконання програми*

Як видно з результатів, хоч перша коаліція і мала перевагу при низькій ціні за рекламу, при високій ціні друга коаліція має перевагу. Зменшимо ціну за рекламу для першої коаліції до 3, отримаємо:

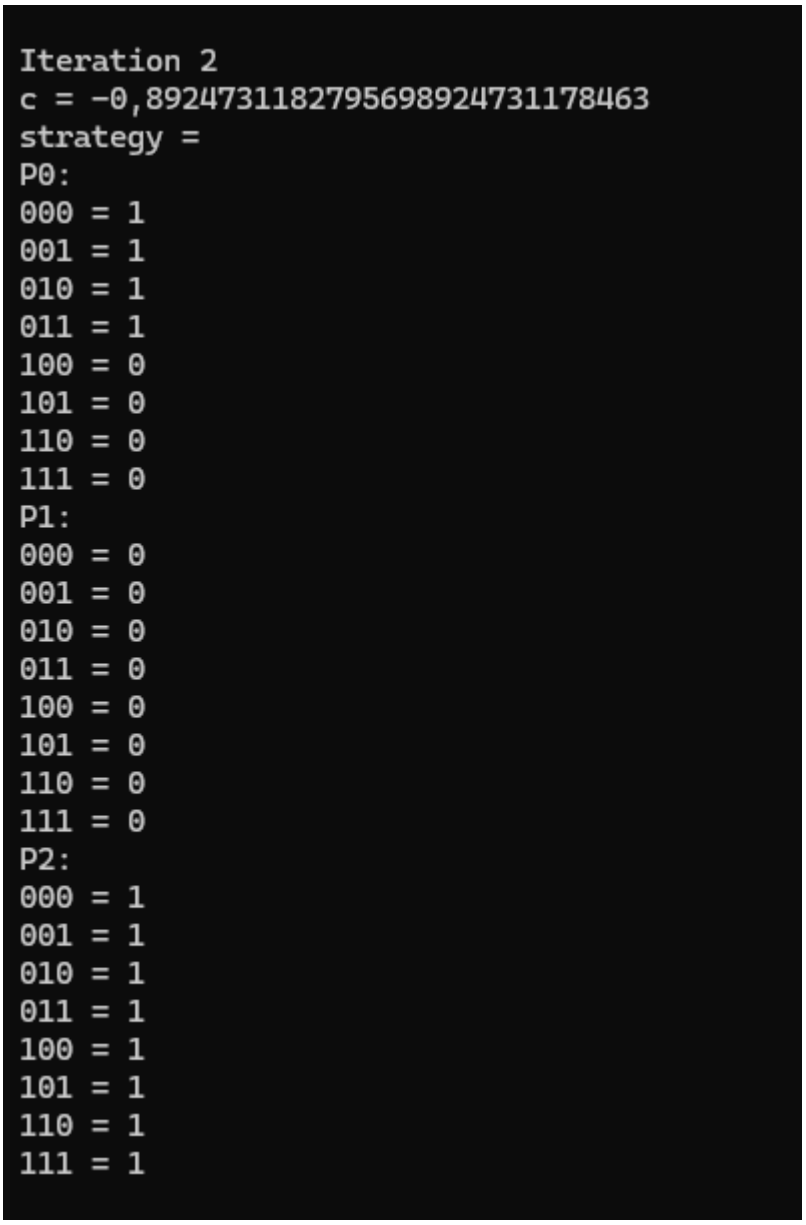


*Рисунок 3.13 Результати виконання програми*

Тобто навіть з перевагою в ціні реклами перша коаліція програє другій. Збільшимо суму виплати, яку отримують гравці з першої коаліції до 12, $p_2 = d_2 = 12$. Матимемо:

```
Iteration 2
c = 0,5422239395237803758339969413
strategy =
P0:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 0
101 = 0
110 = 0
111 = 0
P1:
000 = 0
001 = 0
010 = 0
011 = 0
100 = 0
101 = 0
110 = 0
111 = 0
P2:
000 = 1
001 = 1
010 = 1
011 = 1
100 = 1
101 = 1
110 = 1
111 = 1
```

*Рисунок 3.14 Результати виконання програми*

Після застосування даних змін перша коаліція матиме перевагу. Спробуємо виставити $p_2 = d_2 = 100$. Отримаємо:

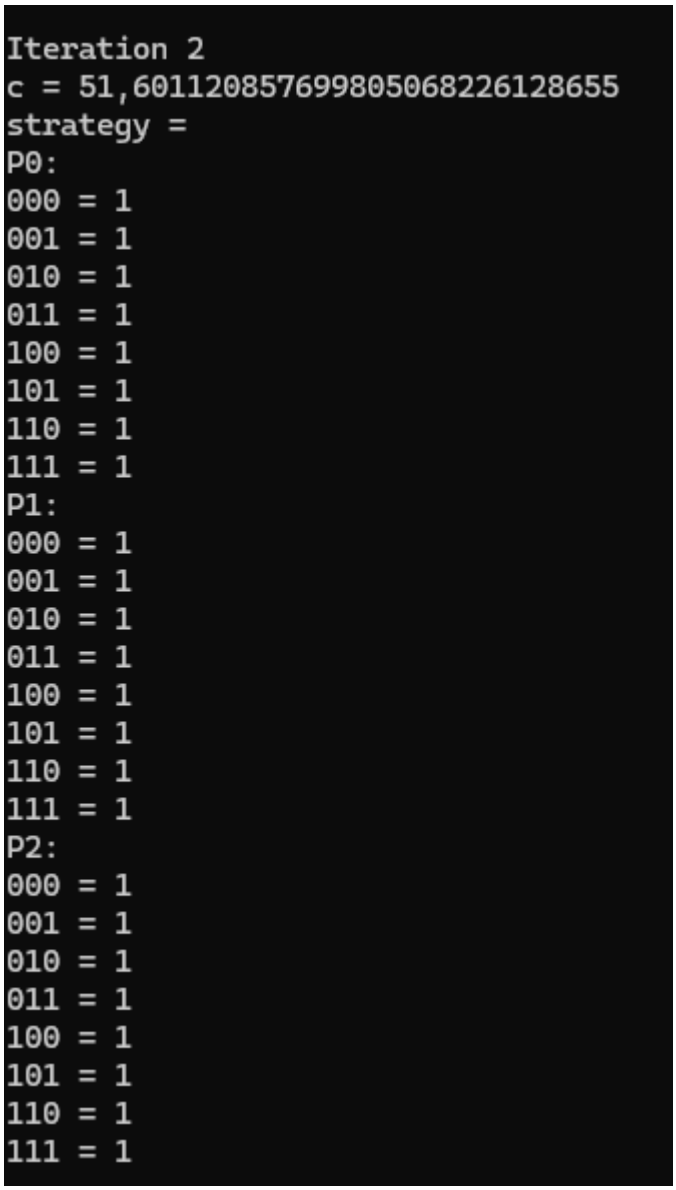


*Рисунок 3.15 Результати виконання програми*

Бачимо, що перший та третій гравці завжди замовляють рекламу, аби бути в кращому стані та отримувати великий прибуток, а другий гравець замовляє рекламу так як таким чином зменшує ймовірність першого та третього отримувати цей прибуток.

# ВИСНОВКИ

У ході виконання даної роботи були досліджені стохастичні коаліційні ігри з локальною структурою взаємодії. Під час роботи над даним дослідженням були зроблені декілька кроків, найголовніші з яких:

- Модифікація ітераційного методу [2], який в оригіналі використовувався для пошуку оптимальних стратегій для одного гравця. Після модифікації даний алгоритм може знаходити оптимальні стратегії для двох коаліцій
- Реалізація вищеназваного алгоритму за допомогою коду. В результаті отримана програма, при заданні початкових умов знаходить та повертає оптимальні стратегії для гравців, а також може симулювати гру на будь яку вказану кількість кроків.
- Проведення аналізу ігор з двома та трьома гравцями на основі раніше написаної програми. За допомогою розуміння концепцій коаліційних ігор з локальною структурою взаємодії, були протестовані різні початкові умови для ігор та проведено аналіз того, на скільки та чому зміняться оптимальні стратегії для гравців та ціна гри в залежності від них.

В подальшому можна розширити можливості алгоритму для вирішення задач з трьома та більше коаліціями та для неперервних множин станів та дій. Також, так як виникли проблеми з кількістю стратегій для коаліцій, важливо знайти спосіб обходу проблем з оптимізацією, що зробить можливим пошук оптимальних стратегій в іграх з великою кількістю гравців.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Chornei R., Daduna H., Knopov P. Stochastic games for distributed players on graphs / Math Meth Oper Res volume 60. – 2004. – 279 - 298 с.
2. Howard R. Dynamic Programming and Markov Processes / R. A. Howard., 1960. – 136 с.
3. Bellman equation [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bellman_equation
4. von Neumann J. Theory of Games and Economic Behavior / J. von Neumann, O. Morgenstern. – Princeton: Princeton University Press, 1944. – 625 с.
5. Equilibrium points in n-person games. / Proceedings of the National Academy of Sciences of the United States of America. – 1950. – №36. – С. 48 - 49.
6. Speyer J. Stochastic Processes, Estimation, and Control / J. L. Speyer, C. H. Walter. – Los Angeles: University of California, Los Angeles, 2008. – 383 с.