Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра математики

Магістерська робота

освітній ступінь – магістр

на тему: **«БАГАТОКРОКОВІ АНТАГОНІСТИЧНІ СТОХАСТИЧНІ ІГРИ»**

Виконав: студент 2-го року навчання
освітньо-наукової програми
«Системний аналіз»,
спеціальності 124 Системний аналіз

Чумак Віталій Віталійович

Керівник: Чорней Р. К.,
кандидат фіз.-мат. наук, доцент

Рецензент_____
(прізвище та ініціали)

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК _____
«____» _____ 20____ р.

Київ – 2022

Календарний план виконання магістерської роботи

Тема: БАГАТОКРОКОВІ АНТАГОНІСТИЧНІ СТОХАСТИЧНІ ІГРИ

Календарний план виконання роботи:

№ п/п	Назва етапу роботи	Термін виконання	Примітка
1	Отримання завдання	15.09.2021	
2	Огляд літератури за темою	10.10.2021	
3	Аналіз існуючої моделі гри	08.11.2021	
4	Аналіз алгоритмів пошуку оптимальних стратегій	23.11.2021	
5	Розробка власної моделі гри	12.12.2021	
6	Написання текстової частини роботи	25.04.2022	
7	Написання технічної частини роботи	10.06.2022	
8	Створення слайдів для доповіді	28.06.2022	
9	Остаточне оформлення роботи та слайдів	30.06.2022	

Студент _____

Керівник _____

“_____”_____

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
кандидат фіз.-мат. наук, доцент
_____ Р.К. Чорней
(підпис)
„____”_____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на магістерську роботу

студенту 2-го курсу, факультету інформатики__
Чумаку Віталію Віталійовичу_____
ТЕМА: **Багатокрокові антагоністичні стохастичні ігри**

Зміст ТЧ до магістерської роботи:

Зміст
Анотація
Вступ
1 Стохастичні ігри
2 Марківські ігри з неповною інформацією для двох гравців
3 Марківські ігри з неповною інформацією для багатьох гравців
4 Порівняння алгоритмів пошуку оптимальних стратегій
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2022 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Зміст

АНОТАЦІЯ

В цій роботі розглядається клас стохастичних ігор, досліджуються повторювані ігри з неповною інформацією на існування оптимальних стратегій для всіх учасників гри. Робота передбачає вивчення Марківської моделі гри з неповною інформацією для двох гравців. В результаті роботи запропонована модифікація моделі для багатьох гравців; алгоритми знаходження оптимальних стратегій для проінформованої та непроінформованої сторін. Технічна частина передбачає реалізацію алгоритмів пошуку оптимальних стратегій та їх порівняння з звичайним перебором всіх можливих поведінок гравців.

Ключові слова: Стохастичні ігри, повторювані ігри, Марківська модель, ігри з неповною інформацією, оптимальні стратегії, коаліційні ігри, теоретико-ігрові моделі.

ВСТУП

Тема повторюваних ігор з неповною інформацією залишається актуальною з моменту їх виникнення – під час холодної війни США та СРСР, модель створювалась як уніфікована система відповіді на дії опонента, яка спирається на додаткову (секретну) інформацію. Зараз такі моделі використовуються ще й на фінансових ринках (Модель фінансового ринку з асиметричною інформацією Де Мейєра та Салі [Додаток 1]).

Об’єктом дослідження роботи є стохастичні ігри, а саме Марківська модель гри з неповною інформацією для двох гравців. Предмет дослідження – алгоритми пошуку оптимальних стратегій в таких іграх. Мета цієї роботи – розглянути модель для двох гравців та існуючих алгоритмів пошуку оптимальних стратегій, а завдання – модифікувати модель та досліджувані алгоритми для випадку багатьох гравців. Метод дослідження це модифікація існуючої моделі та алгоритмів, їх реалізація для тестування на конкретному прикладі гри.

Структура роботи:

1) Розгляд класу стохастичних ігор, основні поняття.
2) Введення до моделі Марківських ігор з неповною інформацією для двох гравців.
3) Створення моделі для багатьох гравців, алгоритми пошуку оптимальних стратегій.
4) Реалізація алгоритмів, порівняння з класичним перебором.

У 1953 році Л. С. Шеплі ввів стохастичні ігри з додатними імовірностями зупинки [1] і показав, що їх значення існує і що обидва гравці володіють оптимальними стаціонарними стратегіями. Його результати відкрили область стохастичних ігор і, як відомо, були узагальнені в багатьох напрямках.

Кожен етап стохастичної гри імовірнісно залежить від попереднього та від дій гравців на цьому етапі. Стохастичні ігри узагальнюють Марківський

процес вирішування(МПВ, MDP – Markov Decision Process) та повторювані ігри. Адже, МПВ – це стохастична гра лише з одним гравцем, а повторювана гра – це стохастична гра з одним станом.

У більшості стратегічних взаємодій, гравці не повністю інформовані про параметри гри, як-от набори дій своїх опонентів і функції виплат, а іноді навіть про свою власну функцію виплат і дії опонентів. Це спостереження спонукає до вивчення ігор з неповною інформацією, яке було започатковано в п'ятдесятих роках. Наклавши обмеження, на поінформованість одного з гравців, було введено поняття ігор з неповною інформацією (SGLIOS – Stochastic Games with Lack of Information on One Side).

Харсаньї [8] представив модель Байєсівських ігор, які є одноетапними іграми з неповною інформацією. Ауман і Машлер [6, 9] досліджувавши повторювані ігри з неповною інформацією, надали елегантну характеристику ігрового значення та описали оптимальні стратегії для гравців. Надалі дослідженням таких ігор займалися Мелолідакіс[3], Солан і Рейнер[7] та ін.

Зазвичай у повторюваних іграх з неповною інформацією параметри гри залишаються незмінними протягом всієї гри, проте іноді ці параметри змінюються під час гри, незалежно від дій гравців. Наприклад, зміни на глобальних ринках впливають на місцевих споживачів і виробників, які, у свою чергу, мають незначний вплив на світовий ринок. Модель, яка враховує цю особливість — це модель ігор Маркова. Дослідження саме Марківської моделі гри з неповною інформацією проводилося Краусом та Рідером [5]. Вони показали існування ігрового значення та оптимальних стратегій для обох гравців, а також презентували скінченні алгоритми для знаходження таких стратегій. Використавши Баєсівський підхід, автори звели Марківську гру з неповною інформацією до неперервної стохастичної гри, де кожен ігровий стан заданий імовірнісним розподілом на множині станів, показали як такі ігри вирішуються за допомогою лінійного програмування, як знайти значення гри, та надали алгоритми для пошуку оптимальних стратегій для обох учасників.

РОЗДІЛ 1: Стохастичні ігри

Стохастичні ігри – повторювані ігри з імовірнісними переходами, розіграні одним або більше гравцями (Шеплі [1]). Такі ігри супроводжуються покроковими змінами станів, відповідно до імовірностей переходів. На кожному етапі гравці обирають певну дію з множини можливих, та, відповідно до стану і обраних дій, отримують певну нагороду. Наступний стан обирається з випадковим розподілом, який залежить від попереднього стану та дій гравців, обраних в ньому.

Покладемо N – скінченна кількість станів, m_k, n_k – скінченні кількості дій обох гравців відповідно, в кожному з станів. Якщо в стані k гравці обирають свої $i_{\text{-ту}}$ та $j_{\text{-ту}}$ дії відповідно, то з імовірністю $s^k_{ij} > 0$ – гра зупиниться, а з імовірністю p^{kr}_{ij} перейде в стан r.

Визначимо $s = \min\limits_{k,i,j} s^k_{i,j}$. Так як s – додатне, гра закінчиться з імовірністю 1 після скінченної кількості кроків, адже для будь якого t, імовірність, що гра не закінчилась після t кроків не більше ніж $(1 - s)^t$.

Підрахунок нагород здійснюється наступним чином: перший гравець отримує a^k_{ij} від другого в стані k та з вибраними діями i та j відповідно. Визначимо верхню границю нагороди як $M = \max\limits_{k,i,j} |a^k_{i,j}|$.

Тоді стає зрозуміло, що загальний очікуваний виграш становить не більше, ніж: $V_n = M + (1\text{-}s)M + (1\text{-}s)^2M + \ldots = M/s$ [1]

Сам процес гри складається з $N^2 + N$ матриць:

$P^{kl} = \left(p^{kl}_{ij} \middle| i = 1,2,\ldots,m_k; j = 1,2,\ldots,n_k\right)$ – матриці переходів між станами

$A^k = \left(a^k_{ij} \middle| i = 1,2,\ldots,m_k; j = 1,2,\ldots,n_k\right)$ – матриці можливих дій

де $k, l = 1,2,\ldots, N$ та $p^{kl}_{ij} \geq 0$, $\left|a^k_{ij}\right| \leq M$, $\sum_{l=1}^{N} p^{kl}_{ij} = 1 - s^k_{ij} \leq 1 - s < 1$ [1]

Вибравши стартову точку (перший стан), буде отримано певно гру Γ^k. Терміну «стохастична гра» тепер відповідає $\Gamma = \{\Gamma^k | \ k = 1, 2, \ldots, N\}$.

Кожен з гравців обирає свій план дій – ігрову стратегію – набір кроків, який враховує кожен етап гри, кожну можливу дію супротивника. Стратегія визначає відповідь гравця в будь який момент гри і для кожного можливого перебігу подій. Набір стратегій – це плани дій для кожного з учасників гри.

Чиста стратегія – алгоритм дій з повною визначеністю, яким чином гравець продовжує гру на тому чи іншому етапі. Вона визначає результат кожної можливої дії гравця під час гри.

Змішана стратегія – це чиста стратегія з імовірністю її вибору. Гравець обирає одну з чистих стратегій відповідно до її імовірностей, вказаних у змішаній стратегії.

Повні набори чистих і змішаних стратегій у стохастичних іграх громіздкі, оскільки вони враховують велику кількість інформації, яка виявляється часто виявляється неактуальною, тому вводиться поняття стаціонарних стратегій, визначених векторами з N-кортежами імовірнісних розподілів:

$\vec{x} = (x^1, x^2, \ldots, x^N,)$, де кожен $x^k = \left(x_1^k, x_2^k, \ldots, x_m^k,\right)$.

РОЗДІЛ 2: Марківські ігри з неповною інформацією для двох гравців

2.1. Повторювані ігри з неповною інформацією

Теоретико-ігрові моделі стають все більш популярними через розвиток систем взаємодій: між людьми, комп'ютерами, процесами. Це створює попит на моделі таких взаємодій, особливо моделі з великою кількістю учасників. Адже, майже кожна взаємодія може бути представлена у вигляді математичної гри, де є певні правила дій, нагороди за той чи інший результат, та ланцюг послідовності, що утворюється виборами відповіді на кожному етапі взяємодії.

Теорія повторюваних ігор з неповною інформацією виникла на початку шістдесятих років зі звітів Аумана і Машлера до Агентства США з контролю над озброєннями та роззброєння (Ауман і Машлер (1968) [6, 9]). Їхня мета полягала в розробці теоретичної ігрової системи для повторних переговорів про роззброєння між СРСР і США. Головною особливістю цієї взаємодії було стратегічне використання інформації в динамічній структурі, тобто при виборі дії сторона має дбати про інформацію, яку вона розкриває, та її вплив на майбутню поведінку опонента, адже розкрита інформація – шанс супернику дізнатися стан та можливості на поточному етапі. Початкова робота, виконана за контрактом з Агентством з контролю над озброєннями та роззброєння Сполучених Штатів, була спрямована на вирішення поступової проблеми роззброєння, в якій жоден гравець не знав, якою буде його власна виплата за будь-яку дану угоду, через невизначеність щодо арсеналу та технології виробництва зброї іншої сторони. Але незабаром дослідження стало набагато більш узагальненим, охоплюючи приховування і відкриття інформації, навчання, а також пов'язані з ними ідеї в будь-якій повторюваній конкурентній ситуації.

Виявилося, що найпростіший приклад гри з нульовою сумою, коли лише одна сторона має секретну інформацію, вже дуже нетривіальний. Такі ігри задані наступною моделлю:

$\boldsymbol{N} = \{1, \ldots, n\}$ – гравці

$\boldsymbol{a_i} \in A_i$, де a_i – конкретна дія $i_{\text{-го}}$ гравця; A_i – його множина можливих дій

$\boldsymbol{u_i} : A \rightarrow R$ – функція виплати

У таких іграх два гравці неодноразово грають в ту саму гру з нульовою сумою, яку обрано випадково перед першим етапом багаторазової взаємодії, відповідно до попереднього розподілу, відомого обом гравцям. Обрана гра повідомляється лише гравцю 1, тому він знає стан, в якому знаходиться, тоді як його опонент – ні. Багатоетапна конфігурація дає непроінформованому гравцеві можливість здогадатися, в якому стані він перебуває, спостерігаючи за попередніми діями гравця 1. У свою чергу, поінформований гравець повинен балансувати між двома іноді протилежними цілями: отримати вигоду від інформації на поточному етапі, але, при цьому, уникнути швидкого «відкриття всіх карт», щоб мати можливість отримати користь у майбутньому.

Застосування ігор з неповною інформацією зрозуміле: найцікавіші особливості соціально-економічних ситуацій обґрунтовуються асиметрією інформації залучених осіб. Модель повторних ігор була задумана для вивчення всіх аспектів, пов'язаних з взаємодіями.

Одним із способів подолати інформаційні труднощі в таких іграх є використання механізму оновлення, який використовує апостеріорні ймовірності в просторі станів з урахуванням історії до поточного етапу. Часто буває, що інформований гравець може обмежити свою увагу стратегіями, які залежать від історії лише через ці умовні ймовірності.

2.2. Модель Марківської гри з неповною інформацією

В свої роботі Шаплі доводить існування оптимальних стратегій для обох гравців в стохастичній грі. Розглянемо Марківську гру з нульовою сумою, з неповною інформацією для двох гравців. В такій грі другий гравець ніколи не інформується про поточний стан гри.

Марківська модель гри з неповною інформацією для двох гравців задається вектором **(S, A, B μ_0, p, r, β)**, де:

S – множина станів,

A, B – множини можливих дій для гравців 1,2 відповідно,

μ_0 ϵ P(s) – так звана апріорна інформація, де *P(S)* – множина всіх розподілів ймовірності на *S*

p – ймовірність переходу, тобто $p_{ij}(a, b)$ – ймовірність, що наступним станом після $i \epsilon S$ буде $j \epsilon S$ при діях *a*, *b* відповідно.

r_i : $A \times B \rightarrow R$ – функція нагороди для гравця *1* в стані $i \epsilon S$

β ϵR_+ – коефіцієнт дисконтування

Гра відбувається в *N* етапів. На початковому етапі вибирається стан i_0. Обидва гравці знають початкову інформацію μ_0. На етапах *{0…N-1}* гравець номер *1* інформований про поточний стан i_n, а гравець *2* – ні. Потім обидва гравці одночасно вибирають дії a_n та b_n відповідно, вони обидва проінформовані про вибір суперника і наступний стан обирається з множини станів з імовірністю $p_{i_n,i_{n+1}}(a_n, b_n)$. На кінцевому етапі *N* гравець *1* отримує від гравця *2* суму $\sum_{n=0}^{N-1} \beta^n r_{i_n}(a_n, b_n)$ нагороди. Позначимо цю гру як $\Gamma_N(\mu_0)$.

На етапі *n* гравець *1* приймає своє рішення в залежності від раніше зібраної ним інформації, тобто послідовності $h_n = (i_0, a_0, b_0, \ldots, i_{n-1}, a_{n-1}, b_{n-1})$, але частина інформації може бути втрачена або забута (Мелолідакіс [2]), тому вибір гравця *1* може залежати тільки від $h_n = (a_0, b_0, \ldots, a_{n-1}, b_{n-1}, i_n)$.

Визначимо певні випадкові стратегії для обох гравців:

Випадкова стратегія для гравця *1* – це послідовність функцій $\pi = (f_n)$

Де f_n: $(A \times B)^n \times S \rightarrow P(A)$

Випадкова стратегія для гравця *2* – це послідовність функцій $\sigma = (g_n)$

Де g_n: $(A \times B)^n \rightarrow P(B)$

Нехай $V_{N\pi\sigma}(\mu)$ – очікувана дисконтована нагорода за N етапів гри з апріорною інформацією μ та з стратегіями гри π та σ відповідно. Тоді V_N – числова функція, при чому $V_0 = 0$.

Відповідно до [5], обидва гравці мають оптимальні стратегії в даній моделі гри. Для їх знаходження введемо так звану скорочену гру, де стани розглядаються як апостеріорні імовірності на вихідному просторі станів S.

Для $\mu \in P(S)$, $f \in F$, $g \in G$ та v: $P(s) \rightarrow R$ мають місце наступні визначення:

$$\Phi(\mu, f, a, b)(j) = \frac{\sum_i \mu(i) f(i,a) p_{ij}(a,b)}{\sum_i \mu(i) f(i,a)}, j \in S \qquad (2.2.1)$$

$$L_{fg} v(\mu) = \sum_{i,a,b} \mu(i) f(i,a) g(b) [r_i(a,b) + \beta v(\Phi(\mu, f, a, b))] \qquad (2.2.2)$$

$$Uv(\mu) = \max_f \min_g (L_{fg} v)(\mu) \qquad (2.2.3)$$

Тут функція $\Phi(\mu, f, a, b)$ – апостеріорна інформація, а $\Phi(\mu, f, a, b)(j)$ – апостеріорна імовірність переходу в стан j, при початковій інформації μ, діях a і b, та функції вибору f; тобто вона трансформує апріорну інформацію в апостеріорну, відповідно до механізму Баєса.

Визначимо: $C \in R^S$ – фіксована скінченна підмножина, $D = B \times C^A$ та функція $T : S \times A \times B \times C^A \to R$ як

$$T(i, a, b, (c_k)) = r_i(a, b) + \beta \sum_j p_{ij}(a, b) c_a(j) \quad (2.2.4)$$

Для обчислення оптимальних стратегій використаємо алгоритми надані в [5]:

Оптимальна стратегія першого гравця:

0) Визначити $n = 0$, $\mu = \mu_0$
1) Знайти оптимальне рішення (z^*, x^*) для $D_{N\text{-}n}(\mu)$, див. (2.2.5)
2) Запам'ятати i_n та вибрати дію a_n базуючись на $f_n^*(a_0, b, \dots, a_{n-1}, b_{n-1}, i_n, a_n) = f^* = \frac{x^*_{i_n a_n}}{\mu(i_n)}$
3) Запам'ятати b_n
4) $n+=1$, $\mu = \Phi(\mu, f^*, a_n, b_n)$, див. (2.2.1)
5) Якщо $n==N$ – зупинитись, інакше – Крок 1.

Оптимальна стратегія другого гравця:

0) При $n = 0$, знайти оптимальне рішення (c^*, y^*) для $P_{N\text{-}n}(\mu_0)$, див. (2.2.6)
1) Вибрати дію b_n базуючись на $g_n^*(a_0, b_0, \dots, a_{n-1}, b_{n-1}, b_n) = g_{N\text{-}n, y^*}(b_n)$
2) Запам'ятати a_n
3) $c^{**} = d_{N\text{-}n, y^*}(a_n, b_n)$
4) $n+=1$, вибрати y^* такий, що (c^{**}, y^*) буде можливим рішенням $P_{N\text{-}n}(\mu_0)$
5) Якщо $n==N$ – зупинитись, інакше – Крок 1.

При цьому:

$$D_c = \begin{cases} z \to \max \\ \sum_{i,a} T(i, a, d) x_{ia} - z \geq 0, d \in D \\ \sum_{a \in A} x_{ia} = \mu(i), \ i \in S \\ z \in R, x_{ia} \geq 0 \end{cases} \quad (2.2.5)$$

$$P_c = \begin{cases} \sum_i \mu(i)c(i) \to min \\ \sum_{d\,\epsilon\,D}\ T(i,a,d)y_d\ \le c(i), (i,a) \in S\ x\ A \\ \sum_{d\,\in\,D} y_d = 1 \\ c(i)\epsilon\ R, \quad y_d \ge 0 \end{cases} \tag{2.2.6}$$

Отже, вирішивши задачу лінійного програмування, отримані алгоритми дозволяють знайти оптимальні стратегії дій для поінформованого та не поінформованого гравців. Кожна оптимальна стратегія складається з оптимальних рішень на кожному етапі гри, які, в свою чергу, базуються на історії (інформація з попередніх кроків гри) та апостеріорному розподілі.

РОЗДІЛ 3: Марківські ігри з неповною інформацією для багатьох гравців

3.1. Модель гри для багатьох гравців

В цьому розділі, на основі встановлення правил локальної взаємодії гравців, запропоновано модифіковану модель, яка являє собою модель для багатьох гравців.

У грі з багатьма гравцями учасники можуть діяти спільно або окремо один від одного. Встановивши правила локальної взаємодії гравців, розглянемо модель, де гравці діють кооперативно, тобто утворюють певні коаліції.

Модель Марківської гри з нульовою сумою з неповною інформацією для M гравців матиме наступний вигляд: **(S, A, μ_0, C, p, r, β)**, де

S – скінченна множина станів гри

A – скінченна множина можливих дій гравців, A = {A^i(s), s ϵ S, i ϵ {1, ... M}}

μ_0 ϵ $P(S)$, де $P(S)$ – імовірнісні розподіли на множині станів, а μ_0 – початкова інформація

V = {v_1, v_2} – коаліції гравців, де $v_i = \{j_1, j_2, \ldots, j_{M/2}\}$, $i \epsilon \{1, \ldots M\}$

p – імовірність переходу, $p_{ij}(a)$ – імовірність перейти зі стану $i \epsilon S$ в стан $j \epsilon S$ з вектором дій $a = (A^1(i), A^2(i), \ldots, A^M(i))$,

r_i (a) – функція нагороди для коаліції v_1 в стані і з вектором дій a.

β ϵR – коефіцієнт дисконтування

Деякі додаткові властивості коаліцій: $V = v_1 \cup v_2; v_1, v_2 \neq \emptyset; v_1 \cap v_2 = \emptyset$

Гра відбувається в N етапів, згідно з наступними правилами: на нульовому етапі, відповідно до μ_0, обирається стан s_0. Кожен гравець проінформований про значення μ_0. На етапі $n \epsilon \{0, 1, \ldots, N\}$ гравці з v_1 проінформовані про поточний стан s_n, тоді як гравці з v_2 – ні. Перебуваючи в

стані s, всі гравці обирають дію $A^i(s)$ одночасно, кожен гравець проінформований про вибір інших. Наступний стан s_{n+1} обирається відповідно до розподілу $p_{s_n s_{n+1}}(a)$. На етапі N коаліція v_1 отримує від v_2 $\sum_{n=0}^{N-1} \beta^n r_n(a_n)$ нагороду.

Розіб'ємо множину A на 2 підмножини: $A = \{A_1, A_2\}$, де A_1 – це множина можливих дій гравців з v_1, A_2 – гравців з v_2 відповідно. За таким принципом розіб'ємо вектор вибраних дій a на a^1, a^2.

На певному етапі n гравці з v_1 можуть робити вибір дії в залежності від інформації, яку вони зібрали на даний момент часу: $(s_0, a_0, \ldots, s_{n-1}, a_{n-1}, s_n)$. Тоді довільна стратегія коаліції v_1 це послідовність $\pi = (f_n)$ функцій, де $f_n : (A)^n \times S \rightarrow P(A_1)$. Множину всіх можливих таких стратегій позначимо як Δ, а множину всіх функцій f_n позначимо F.

Для гравців з v_2: довільна стратегія – це послідовність $\sigma = (g_n)$ функцій, де $g_n : (A)^n \rightarrow P(A_2)$. Позначимо Σ – всі можливі стратегії для v_2 та G – множина всіх функцій g_n.

Нехай $V_{N\pi\sigma}(\mu)$ – очікувана дисконтована нагорода за N етапів гри з апріорною інформацією μ та з стратегіями гри π та σ відповідно. Тоді V_N – числова функція, при чому $V_0 = 0$.

Для $\mu \in P(S)$, $f \in F$, $g \in G$ та $v: P(s) \rightarrow R$ мають місце наступні визначення:

$$\Phi(\mu, f, a)(j) = \frac{\sum_i \mu(i) f(i,a^1) p_{ij}(a)}{\sum_i \mu(i) f(i,a^1)}, j \in S \qquad (3.1.1)$$

$$L_{fg} v(\mu) = \sum_{i,a,b} \mu(i) f(i,a^1) g(a^2) [r_i(a) + \beta v(\Phi(\mu, f, a))] \qquad (3.1.2)$$

$$Uv(\mu) = \max_f \min_g (L_{fg} v)(\mu) \qquad (3.1.3)$$

Тут функція $\Phi(\mu, f, a)$ – апостеріорна інформація, а $\Phi(\mu, f, a)(j)$ – апостеріорна імовірність переходу в стан j, при початковій інформації μ, вектору дій a, та функції вибору f; тобто вона трансформує апріорну інформацію в апостеріорну, відповідно до механізму Баєса.

Визначимо: $C \in R^S$ – фіксована скінченна підмножина, $D = A_1 \times C^{A_1}$ та функція $T : S \times A_1 \times A_2 \times C^{A_1} \to R$ як

$$T(i, a, (c_k)) = r_i(a) + \beta \sum_j p_{ij}(a) c_{a_1}(j) \qquad (3.1.4)$$

3.2. Алгоритми пошуку оптимальних стратегій

Для пошуку оптимальних стратегій використаємо алгоритми з [5], підставивши замість першого та другого гравців, першу та другу коаліції гравців. Така підстановка збільшить кількість обчислень на кожному кроці, для її реалізації скористаємось виведеними формулами з пункту 3.1.

$$D_c = \begin{cases} z \to \max \\ \sum_{i,a^1} T(i,a^1,d)x_{ia^1} - z \geq 0, d \in D \\ \sum_{a^1 \in a} x_{ia^1} = \mu(i),\ i \in S \\ z \in R, x_{ia^1} \geq 0 \end{cases} \quad (3.2.1)$$

$$P_c = \begin{cases} \sum_i \mu(i)c(i) \to min \\ \sum_{d \in D} T(i,a^1,d)y_d \leq c(i), (i,a^1) \in S\ x\ A_1 \\ \sum_{d \in D} y_d = 1 \\ c(i) \in R, \quad y_d \geq 0 \end{cases} \quad (3.2.2)$$

Використавши надані вище задачі лінійного програмування, побудуємо алгоритми знаходження оптимальних стратегій для обох коаліцій:

Оптимальна стратегія першої коаліції:

0) Визначити $n = 0$, $\mu = \mu_0$
1) Знайти оптимальне рішення (z^*, x^*) для $D_{N\text{-}n}(\mu)$, див. (3.2.1)
2) Запам'ятати i_n та вибрати вектор дій a^1 базуючись на $f_n^*(a_0, \dots, a_{n-1}, i_n, a_n) = f^* = \frac{x^*_{i_n a_n^1}}{\mu(i_n)}$
3) Запам'ятати a^2
4) $n+=1$, $\mu = \Phi(\mu, f_n^*, a)$, див. (3.1.1)
5) Якщо $n==N$ – зупинитись, інакше – Крок 1.

Оптимальна стратегія другої коаліції:

0) *При* $n=0$ знайти оптимальне рішення (c^*, y^*) для $P_{N-n}(\mu_0)$, див. (3.2.2)
1) Вибрати вектор дій a^2 базуючись на $g_n^*(a_0, \ldots, a_{n-1}) = g_{N-n,y^*}(a^2)$
2) Запам'ятати a^1
3) $c^{**} = d_{N-n,y^*}(a^1, a^2)$
4) $n+=1$, вибрати y^* такий, що (c^{**}, y^*) буде можливим рішенням $P_{N-n}(\mu_0)$
5) Якщо $n==N$ – зупинитись, інакше – Крок 1.

РОЗДІЛ 4: Порівняння алгоритмів пошуку оптимальних стратегій

4.1. Реалізація повного перебору можливих дій гравців

4.1.1. Постановка задачі, параметри гри

Для оцінки роботи алгоритмів, було вибрано варіант порівняння з набором всіх можливих ігрових шляхів. Для цього було розглянуто конкретний приклад Марківської гри з неповною інформацією з наступними параметрами:

$M = 4$ – к-сть гравців (по 2 на кожну коаліцію)

$N = 2$ – к-сть етапів гри

$S = [1, 2]$ – множина станів гри

$A = [(0,1); (0,1); (0,1); (0,1)]$ – множини можливих дій для кожного гравця

$\mu_0(1) = \mu_0(2) = 0.5$ – початковий розподіл

$\beta = 1$ – коефіцієнт дисконтування

$$p_{i,j}(a) = \delta_{i,j} = \begin{cases} 1, \text{якщо } i == j \\ 0, \text{якщо } i \neq j \end{cases}$$

$r_1(a)$ – функція нагороди для гравця 1, в стані 1 та з вектором дій a – обраховується як різниця між 1 та 0 у векторі дій ($r_1([1,0,0,0]) = -2$)

$r_2(a)$ – функція нагороди для гравця 1, в стані 2 та з вектором дій a – обраховується як різниця між 0 та 1 у векторі дій ($r_2([1,0,0,0]) = 2$)

4.1.2. Структура програмного застосунку

Для реалізації було використано мову програмування *Python* версії *3.8*, середовище – *PyCharm*. Для написання було використано бібліотеки *numpy* та *itertools*.

Програма написана в процедурному стилі та складається з трьох частин: введення початкових значень гри, функції обчислення (*r1(), r2(), p(), init_stage(), choose_state(), reward(), get_all_actions()*), та перебіг самої гри.

Після запуску відбувається повний перебір симуляцій проходження гри. Перебір відбувається по можливих станах гри, діях гравців, функціях виплати.

В результаті виконання програми користувач отримує перебір всіх можливих варіантів гри, значення нагород для кожної коаліції та порядок дій кожного гравця в кожному варіанті гри.

4.1.3. Представлення отриманих результатів

Після виконання програми було отримано 512 різних варіантів гри з повним описом дій, нагород та станів гри, а також значення максимальної (для першої коаліції) та мінімальної(для другої коаліції) нагороди.

```
Game path  510
Game's states:  [2, 2]
Coalition1's acts:  [[0, 0], [0, 0]]
Coalition2's acts:  [[0, 0], [1, 0]]
Coalition1 reward:  6
Coalition2 reward:  -6
Game path  511
Game's states:  [2, 2]
Coalition1's acts:  [[0, 0], [0, 0]]
Coalition2's acts:  [[0, 0], [0, 1]]
Coalition1 reward:  6
Coalition2 reward:  -6
Game path  512
Game's states:  [2, 2]
Coalition1's acts:  [[0, 0], [0, 0]]
Coalition2's acts:  [[0, 0], [0, 0]]
Coalition1 reward:  8
Coalition2 reward:  -8
Max reward for Coalition1: game - 1, reward =  8
Max reward for Coalition2: game - 256, reward =  8
```

Зображення 1

Максимальну нагороду перша коаліція отримала за наступних умов:

```
Game path  1
Game's states:  [1, 1]
Coalition1's acts:  [[1, 1], [1, 1]]
Coalition2's acts:  [[1, 1], [1, 1]]
Coalition1 reward:  8
Coalition2 reward:  -8
```

Зображення 2

Гру було зіграно двічі в *стані 1*, і всі гравці обирали *дію 1* в обох етапах. Очевидно, що найкращий варіант для другої коаліції – протилежний: обрати *дію 0* на обох етапах гри.

Щодо випадку, коли гра гралася в *стані 2*, результати обернені: найкращим варіантом для першої коаліції буде вибір *дії 0* на обох етапах гри.

Отже, зробивши повний перебір варіантів перебігу гри, легко побачити, що відповідно до стану гри, а отже, й до функції нагороди, коаліціям вигідно вибрати або всі *1,* або всі *0* для отримання максимальної виплати. Оскільки перша коаліція володіє інформацією про поточний стан гри, вона діятиме саме за такою стратегією.

Такий висновок є очевидним в 2-етапнй грі з невеликою кількістю учасників, бо перебір є швидким і невеликим. Як же варто реагувати на гру з сотнею етапів та тисячами гравців в кожній з коаліцій. З такими параметрами гри повний перебір зробити практично неможливо, або це займе «вічність». Для цього розглянемо та реалізуємо алгоритми з попереднього розділу, а також порівняємо їх з результатами повного ігрового перебору.

4.2. Реалізація алгоритмів пошуку оптимальних стратегій

4.2.1. Структура програмного застосунку

Для реалізації було використано мову програмування *Python* версії *3.8*, середовище – *PyCharm*. Для написання було використано бібліотеки *numpy* та *cvxpy*. Остання – бібліотека для вирішення задач лінійного програмування, де створюється завдання, додається цільова функція та функції обмеження. Приклад використання для вирішення задачі D_C (Формула 3.2.1):

```
_x = cp.Variable(shape=(len(coeffs), 1), name="x")
_A = np.array(coeffs)
_constraints = [cp.matmul(_A, _x) >= 0, cp.sum(_x) == _mu, _x >= 0]
_objective = cp.Maximize(cp.matmul(_A, _x))
_problem = cp.Problem(_objective, _constraints)
_solution = round(_problem.solve(), 3)
```

Зображення 3

Програма написана в процедурному стилі та складається з трьох частин: введення початкових значень гри, функції обчислення (*r1(), r2(), p(), init_stage(), choose_state(), T(),D(), f_star(), choose_action(), choose_best_a2(), F()*) та перебіг самої гри.

В результаті виконання програми користувач отримує матриці можливих дій на обох етапах гри для першої коаліції, співставленні з можливими діями гравців з *Коаліції 2*.

4.2.2. Представлення отриманих результатів

Реалізувавши описані вище алгоритми оптимальних стратегій, отримуємо результати для вибору оптимальних дій кожного з гравців в будь якому стані гри. Результати подано у наступному форматі:

[дія коаліції 2] : [перелік корисності можливих відповідей коаліції 1]

Таких можливих відповідей матимемо чотири: *[0, 0] [0, 1] [1, 0] [1, 1]*.

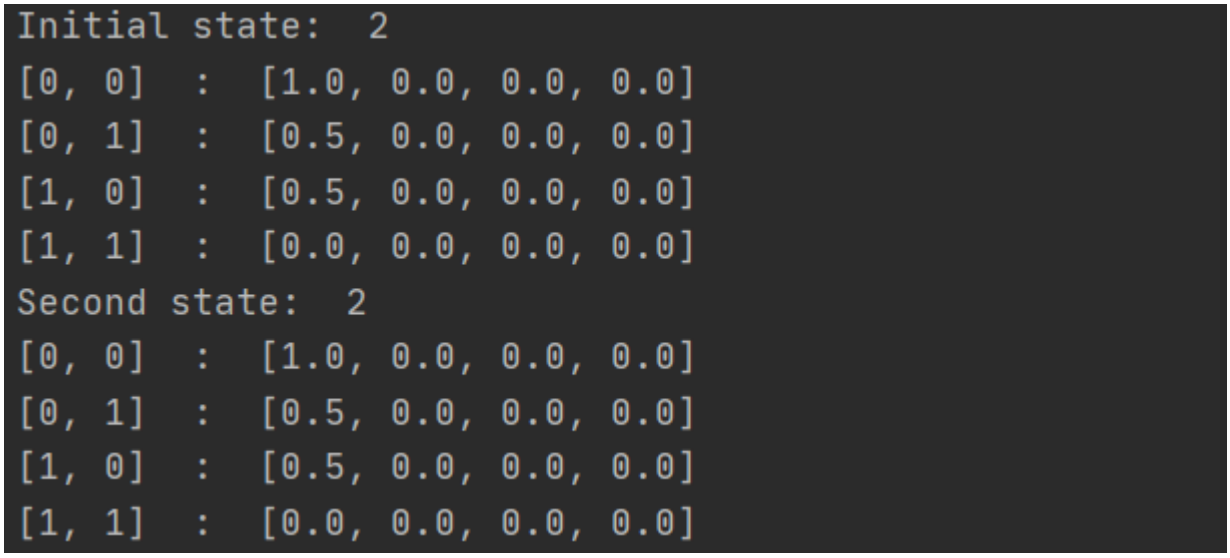
Після виконання програми було отримано:

```
Initial state:  1
[0, 0]  :  [0.0, 0.0, 0.0, 0.0]
[0, 1]  :  [0.0, 0.0, 0.0, 0.5]
[1, 0]  :  [0.0, 0.0, 0.0, 0.5]
[1, 1]  :  [0.0, 0.0, 0.0, 1.0]
Second state:  1
[0, 0]  :  [0.0, 0.0, 0.0, 0.0]
[0, 1]  :  [0.0, 0.0, 0.0, 0.5]
[1, 0]  :  [0.0, 0.0, 0.0, 0.5]
[1, 1]  :  [0.0, 0.0, 0.0, 1.0]
```

Зображення 4

Отож, якщо гра почалася в стані 1, тому буде застосовано функцію нагороди *r1(a)* – обраховується як різниця між 1 та 0 у векторі дій (*r1([1,0,0,0]) = -2*), тому першій коаліції оптимально вибирати останню дію ([1,1]) при будь яких діях коаліції номер 2.

Наступний випадок – гра почалася в стані 2:



Зображення 5

В даному випадку початковий стан – 2, відповідно буде застосовано функцію нагороди $r_2(a)$ – обраховується як різниця між 0 та 1 у векторі дій ($r_1([1,0,0,0]) = 2$), тому першій коаліції оптимально обирати першу відповідь [0, 0] при будь яких діях другої коаліції гравців.

ВИСНОВКИ

Отже, в роботі було проаналізовано клас стохастичних ігор, досліджено підвид ігор з неповною інформацією, а саме Марківських ігор на прикладі моделі для двох гравців, розглянуто існуючі алгоритми пошуку оптимальних стратегій.

Після проведених досліджень, встановивши обмеження локальної взаємодії гравців, шляхом введення коаліцій, було розроблено модель Марківської гри з неповною інформацією для багатьох гравців. Базуючись на існуючих шляхах пошуку оптимальних стратегій для моделі з двома гравцями, було розроблено алгоритми пошуку оптимальних стратегій, реалізовані та протестовані на конкретному прикладі в розділі 4.

Результати, надані алгоритмом, збігаються з інтуїтивно правильним вибором, але, що важливіше, отримавши всі можливі варіанти розвитку гри прямим перебором, стає очевидно, що найкращий варіант той, що в результаті виконання пропонує алгоритм.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. SHAPLEY L. S. STOCHASTIC GAMES. PRINCETON, 1953. 6 p. URL: https://www2.cs.siu.edu/~hexmoor/classes/CS539-F10/Shapley.pdf

2. Melolidakis C. (1984) On stochastic games with lack of information on one side. PhD Thesis, University of California, Los Angeles

3. Melolidakis C. (1989) On stochastic games with lack of information on one side. Intemat. J. Game Theory 18:1-29

4. Melolidakis C. (1991) Stochastic games with lack of information on one side and positive stop probabilities. In: Raghavan TES Stochastic Games with Lack of Information on One Side and Positive Stop Probabilities

5. ALEXANDER KRAUSZ and ULRICH RIEDER (1997) Markov Games with Incomplete Information Markov games with incomplete information

6. Aumann R., Maschler M. (1995) Repeated games with incomplete information. Cambridge Repeated Games with Incomplete Information

7. Rainer C. , Solan E. (2019) Solving Two-State Markov Games with Incomplete Information on One Side. HAL Solving Two-State Markov Games with Incomplete Information on One Side

8. Harsanyi J. C. (1967) Games with incomplete information played by “Bayesian” players, I–III Part I. The basic model, Management science

9. Aumann, R. J., Maschler M. B. (1968) Repeated games of incomplete information: The zero-sum extensive case, in Report of the U.S. Arms Control and Disarmament Agency ST-143, Washington, D.C.

10. De Meyer B, Saley H M (2003) On the strategic origin of Brownian motion in finance. Int J Game Theory

11. Ruslan K. Chornei, Hans Dadunay , Pavel S. Knopov (2004) Stochastic games for distributed players on graphs

12. Tony A. R. Solving Linear Programming problems in Python using cvxpy library. OpenGenus IQ: Computing Expertise & Legacy. URL: https://iq.opengenus.org/solving-linear-programming-in-python/

ДОДАТКИ

Додаток 1

Модель фінансового ринку з асиметричною інформацією (Де Мейер [10]):

$S = \{0, 1\}$

$A = B = [0, 1]$

$r_i(a, b) = sgn(a - b)\ (i - max\{a, b\})$

Суть гри полягає у наступному: стан i є ліквідаційною вартістю ризикованого активу, і тільки *гравець 1* – інсайдер, знає i. На кожному етапі гри обидва гравці пропонують свої ціни a і b для активу, і гравець з вищою ціною купує одну одиницю активу у свого опонента за цю ціну. У гравців достатньо активів і грошей. Мета обох гравців — максимізувати очікуваний прибуток після N раундів торгів

Додаток 2

```
from itertools import *
import numpy as np

M = 4
S = [1, 2]
A = [[0, 1], [0, 1], [0, 1], [0, 1]]  # , [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1]]
mu0s = [0.5, 0.5]
beta = 1
N = 2


def p(_i, _j, _a, ):
    if _i == _j:
        return 1
    return 0


def r1(_a):  # 1 0 0 0 1 0 0 0 1 1
    _res_a = []
    for ia in _a:
        if ia == 0:
            _res_a.append(-1)
        else:
            _res_a.append(1)
    res = sum(_res_a)
    return res
```

```
def r2(_a):
    _res_a = []
    for ia in _a:
        if ia == 1:
            _res_a.append(-1)
        else:
            _res_a.append(1)
    res = sum(_res_a)
    return res


def init_stage(_S, _mu0s):
    return np.random.choice(_S, p=_mu0s)


def choose_state(_in, _S, _a):
    probs = [p(_in, _si, _a, ) for _si in _S]
    return np.random.choice(_S, p=probs)


def reward(_In, _a):
    if _In == 1:
        return r1(_a)
    else:
        return r2(_a)
```

```
def get_all_actions():
    pr = product('10', repeat=M * 2)
    pr = list(pr)
    res = []
    for p in pr:
        p2 = []
        for sym in p:
            p2.append(int(sym))
        res.append(p2)
    # print(res)
    return res



In = 1  # init_stage(S, mu0s)
pathA = []
pathB = []
path_state = []
i = 1
coal1_rewards = []
coal2_rewards = []
actions = get_all_actions()
for In in range(1, 3):
    for act in actions:
        acts = act[:len(act)//2], act[len(act)//2:]
        v1 = acts[0][:len(acts[0])//2]
        v2 = acts[0][len(acts[0])//2:]
        reward1 = 0
        reward2 = 0
```

```
        for step in range(0, 2):
            path_state.append(In)
            reward1 += reward(In, v1+v2)
            reward2 -= reward(In, v1+v2)
            In = choose_state(In, S, v1+v2)
            pathA.append(v1)
            pathB.append(v2)
            v1 = acts[1][:len(acts[0]) // 2]
            v2 = acts[1][len(acts[0]) // 2:]
        print("Game path ", i)
        i += 1
        print("Game's states: ", path_state)
        print("Coalition1's acts: ", pathA)
        print("Coalition2's acts: ", pathB)
        print("Coalition1 reward: ", reward1)
        coal1_rewards.append(reward1)
        print("Coalition2 reward: ", reward2)
        coal2_rewards.append(reward2)
        pathA = []
        pathB = []
        path_state = []
print("Max reward for Coalition1: game - " +
str(coal1_rewards.index(max(coal1_rewards))+1) + ", reward = ",
max(coal2_rewards))
print("Max reward for Coalition2: game - " +
str(coal2_rewards.index(max(coal2_rewards))+1) + ", reward = ",
max(coal2_rewards))
```

Додаток 3

```
import cvxpy as cp

import numpy as np


M = 4

S = [1, 2]

A = [[0, 1], [0, 1], [0, 1], [0, 1]]  # , [0, 1], [0, 1], [0, 1], [0, 1], [0, 1], [0, 1]]

a1 = [[0, 0], [0, 1], [1, 0], [1, 1]]  # possible first team acts in one state

a2 = [[0, 0], [0, 1], [1, 0], [1, 1]]  # possible first team acts in one state

mu0s = [0.5, 0.5]

beta = 1

N = 2



def p(_i, _j, _a, ):
    if _i == _j:
        return 1
    return 0



def r1(_a):  # 1 0 0 0 1 0 0 0 1 1
    _res_a = []
    for ia in _a:
```

```
        if ia == 0:
            _res_a.append(-1)
        else:
            _res_a.append(1)
    res = sum(_res_a)
    return res



def r2(_a):
    _res_a = []
    for ia in _a:
        if ia == 1:
            _res_a.append(-1)
        else:
            _res_a.append(1)
    res = sum(_res_a)
    return res



def init_stage(_S, _mu0s):
    return np.random.choice(_S, p=_mu0s)
```

```
def choose_state(_in, _S, _a):
    probs = [p(_in, _si, _a, ) for _si in _S]
    return np.random.choice(_S, p=probs)



def T(_i, _a1, _a2, _c):
    _res = 0
    if _i == 1:
        _res = r1(_a1 + _a2)
    else:
        _res = r2(_a1 + _a2)
    _sum = 0
    for j in range(1, N + 1):
        _sum += p(_i, j, _a1 + _a2) * _c
    return _res + beta * _sum



def D(_N, _n, _mu, _i, _a2i):
    # X1(0,0) X1(0,1) X1(1,0) X1(1,1) X2(0,0) X2(0,1) X2(1,0) X2(1,1)
    coeffs = []
    for i in range(1, _N + 1):
```

```
        for act1 in a1:
            # for act2 in a2:
            coeffs.append(T(i, act1, _a2i, _N - _n))
    _x = cp.Variable(shape=(len(coeffs), 1), name="x")
    _A = np.array(coeffs)
    _constraints = [cp.matmul(_A, _x) >= 0, cp.sum(_x) == _mu, _x >= 0]
    _objective = cp.Maximize(cp.matmul(_A, _x))
    _problem = cp.Problem(_objective, _constraints)
    _solution = round(_problem.solve(), 3)
    # print("Z= ", _solution)
    res = [round(_val[0], 2) for _val in _x.value]
    for i in range(0, len(res)):
        if res[i] == -0:
            res[i] = 0
    # print("Vals: ", res)
    return [res[:len(res) // 2], res[len(res) // 2:]]


def f_star(_Xs, _In):
    res = []
    _current_Xs = _Xs[_In - 1]
    for _x in _current_Xs:
```

```
        res.append(_x / 0.5)
    return res



def choose_action(_fst_s):
    res = [0, 0, 0, 0]
    for _fst in _fst_s.values():
        for i in range(0, len(res)):
            res[i] += _fst[i]
    return res.index(max(res))



def choose_best_a2(_In):
    if _In == 1:
        return [1, 1]
    if _In == 2:
        return [0, 0]



def F(_mui, _f_sts, _a, _best_a2):
    probs = []
    sum1 = 0
```

```
    sum2 = 0
    for j in range(len(S)):
        for i in range(len(S)):
            res = _mui * sum(_f_sts[str(_best_a2)])
            sum1 += res * p(i, j, _a)
            sum2 += res
        probs.append(sum1 / sum2)
    return max(probs)



mu = 0.5
In = init_stage(S, mu0s)
print("Initial state: ", In)
for n in range(0, N):
    f_sts = {}
    for a2i in a2:
        Xs = D(N, n, mu, In, a2i)
        f_st = f_star(Xs, In)
        f_sts[str(a2i)] = f_st
    action = choose_action(f_sts)
    best_a2 = choose_best_a2(In)
    a = a1[3] + best_a2
```

```
mu = F(mu, f_sts, a, best_a2)
for key, val in f_sts.items():
    print(key, " : ", val)
print("Second state: ", In)
```