МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

ВИПУСКНА РОБОТА

на тему:

«Комп'ютерна реалізація чисельних методів розрахунку матричних ігор мовою програмування Python»

Завідувач випускаючої кафедри — Довбиш А. С.

Керівник роботи — Шаповалов С. П.

Студента групи ІНз – 63-8с — Любченко О. Ю.

СУМИ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Центр заочної, дистанційної і вечірньої форм навчання
Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2020 р.

ЗАВДАННЯ

до випускної роботи

Студента п'ятого курсу, групи ІНз-63-8С спеціальності "Інформатика" заочної форми навчання **Любченка Олександра Юрійовича**

Тема:"«Комп'ютерна реалізація чисельних методів

розрахунку матричних ігор мовою програмування Python»

Затверджена наказом по СумДУ

№_____ от _____2020 р.

Зміст пояснювальної записки: 1) аналітичний огляд методів розрахунку матричних ігор; 2) постановка завдання й формування завдань дослідження; 3) опис основних положень, математичних моделей і алгоритмів, що використовуються для рішення поставленого завдання; 5) розробка інформаційного й програмного забезпечення; 6) аналіз результатів моделювання.

Дата видачі завдання "_____"_____2020 р.

Керівник випускної роботи _____ Шаповалов С. П.

Завдання приняв до виконання _____ Любченко О. Ю.

РЕФЕРАТ

Записка: 44 стор., 8 рис., 5 табл., 3 додаток, 8 джерел інформації.

Об’єкт дослідження — моделі теорії ігор.

Мета роботи — застосування чисельних методів для розв’язку задач теорії ігор.

Методи дослідження — математичне моделювання, комп’ютерна реалізація алгоритмів на ЕОМ.

Результати — розроблено інформаційне та програмне забезпечення чисельного розв’язання задач теорії ігор. Застосовані алгоритми Монте-Карло, Брауна-Робінсона, направленого перебору. Комп’ютерна реалізація алгоритмів проведена за допомогою алгоритмічної мови програмування Python.

ТЕОРІЯ ІГОР, МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ, АЛГОРИТМ МОНТЕ-КАРЛО, АЛГОРИТМ БРАУНА-РОБІНСОНА, ПОРІВНЯЛЬНИЙ КОМП’ЮТЕРНИЙ АНАЛІЗ, МОВА ПРОГРАМУВАННЯ PYTHON

ЗМІСТ

ВСТУП..... 5

1 АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ 7

1.1Огляд існуючих моделей в теорії ігор 7

1.2 Постановка задачі..... 16

2 МАТЕМАТИЧНА ПОСТАНОВКА ЗАВДАННЯ ТА ВИБІР МЕТОДУ ЇЇ РІШЕННЯ. 17

2.1 Короткий огляд відомих рішень..... 17

2.2 Численне рішення методом Монте-Карло 19

2.3 Ітераційні методи рішення..... 23

3 КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ ТА ТЕСТОВІ РОЗРАХУНКИ……..…….27

3.1 Вибір мови програмування 27

3.2 Комп'ютерна реалізація алгоритмів 29

3.3 Тестові розрахунки 33

ВИСНОВКИ..... 36

СПИСОК ЛІТЕРАТУРИ..... 37

ДОДАТОК А..... 38

ДОДАТОК Б..... 41

ДОДАТОК В 43

ВСТУП

Теорія ігор – це інструментарій «стратегічного мислення» в будь-яких взаємовідносинах, який передбачає математичне моделювання ситуацій взаємодії двох або більше учасників та намагається за допомогою їх рішення надати учасникам рекомендацій задля одержання максимальної для себе вигоди. Самі ситуації взаємодії називаються конфліктами, а застосовність саме інструментарію теорії ігор обумовлено саме вигідними результатами які одержуються гравці (учасники конфлікту), коли надані рекомендації будуть вжиті [1-4].

Міжнародна енциклопедія з соціальних і поведінських наук надає наступне визначення грі: «Гра - це будь - яка ситуація, в якій два бо більша кількість гравців, які не завжди мають однакові інтереси, намагаються досягти виграшу».

У минулому, розроблений в основному для військових цілей і оборони, ігровий інструментарій також використовувався як альтернативний та взаємодоповнюючий підхід для досліджень а умовах невизначеностей в багатьох сферах, таких як економіка, інженерія, інформатика. Основні питання, якими займалися фахівці з теорії ігор в 1950-60-е, були пов'язані із зовнішньою політикою, зокрема ядерним стримуванням і гонкою озброєнь.

Однак останнім часом теорія ігор набирає позиції в системах та інженерії управління, здебільшого в інженерних системах із залученням людей, де існує тенденція використовувати теоретичні інструменти для ігор для розробки протоколів, які сприятимуть співпраці людей. Наприклад, вчені, як правило, використовують ігрові теоретичні інструменти для проектування оптимальних потоків руху або прогнозування або уникнення відключень в електромережах або перевантажень в керованих кібер-фізичними системами керованих мережах [1-4]. Такий зріст «нових» застосувань теорії ігор зумовив новий «виток» її популярності і зріст нових досліджень в цих застосуваннях [1-7].

Ігрове моделювання почалось навіть використовується в мережевих дослідженнях, в яких раніше й не мріялось засновникам теорії ігор, як теоретичний інструментарій прийняття рішень в задачах маршрутизації, контролю заторів, обмін ресурсами тощо. Порівняно новими є дослідження, що стосуються, на-

приклад, вираховуванням взаємодій між користувачами та постачальниками мережевих технологій в мережі 5G, щоб покращити прийняття рішень. Мережам 5G доведеться обробляти дані (збір, зберігання, видобуток, аналіз тощо), зібрані з дуже різноманітних набір таких джерел, як трафік, погода, інциденти із безпекою, натовпи тощо. Аналіз даних та управління мережею необхідне для розгортання 5G. IoT включає датчики і мобільні пристрої, які збирають дані та здійснюють обробку даних для передбачення певні обставини, включаючи поведінку людини.

Отже, теорія ігор – це математична теорія аналізу стратегічної поведінки учасників конфліктів будь якого спрямування. ЇЇ основним завданням є надання цим учасникам порад які можливості з тих, що має кожний з учасників застосувати, щоб одержати для себе найкращий результат. В результаті дій всіх учасників виникає ситуація рівноваги, яка і є результатом гри. Саме пошук рівноваги та вибір стратегій гравців для її здобуття являються шуканими результатами для дослідження.

1 АНАЛІЗ ПРОБЛЕМИ. ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Огляд існуючих моделей в теорії ігор

Теорія ігор передбачає розгляд різноманітних ситуацій зіткнення інтересів, дій, які гравці роблять в подібних ситуаціях і тих результатів, які реалізуються в результаті. При цьому передбачається, що у кожного гравця є певні переваги по відношенню до наслідків, тобто він може порівнювати результати і вирішувати, які для нього більш, а які менш переважні. Таким чином, якщо відволіктися від взаємодії гравця з іншими гравцями, а обмежитися лише його власними діями і виникаючими наслідками, то тут не обійтися без аналізу індивідуального вибору, тобто без аналізу того, який вибір зробить гравець, якщо йому буде запропонований деякий набір альтернатив. Це завдання про індивідуальний прийнятті рішень є основою всіх міркувань в теорії ігор.

Математичний опис гри передбачає три моделі подання конфлікту: нормальна форма, розгорнута форма (інакше позиційна форма) та ігри, представлені характеристичними функціями. Причому основні труднощі, як правило, полягають не в виконанні розрахунків за побудованими моделями теорії ігор, а в побудові самих моделей, адекватних реальній ситуації [1]. В її основі лежать припущення про те, що розглянуті суб'єкти мають деякі цілі, є в певному сенсі раціональними і при прийнятті власних рішень мислять стратегічно, тобто враховують внутрішню структуру ситуації і те, які рішення можуть приймати інші.

Модель гри в нормальній формі (принаймні для двох гравців, N={1,2}) що якнайкраще представляється платіжною матрицею або таблицею, в якій кожен рядок і відповідає вибору гравцем 1 однієї з множини своїх стратегій (дій), а стовпчик j відповідає вибору гравцем 2 своєї стратегії в цій грі (табл. 1.1). Елемент матриці α_{ij}, що знаходиться на перетині і-рядку та j-стовпчику визначає результат (наслідок) їх вибору. Якщо гра не антагоністична, то цей результат буде складатися з двох платежів, розділених, наприклад, комою, тобто і для гравця 1 і для гравця 2, якщо гра антагоністична, то в матриці достатньо відобразити тільки виграші одного гравця (див. табл. 1.1). Модель гри, яка подається в нормальній формі найчастіше називають матричною грою.

Тому кожен гравець стикається з питанням про те, які дії є для нього найкращими. Оскільки гравці взаємозалежні, то відповідь на це питання буде залежати не тільки від власних уподобань гравця, але і від того, як діють інші гравці. Отже, гравець повинен будувати припущення з приводу того, що будуть робити інші гравці.

Таблиця 1.1 Платіжна матриця гри в нормальній формі

Стратегії гравця 1	Стратегії гравця 2						
		B_1	B_2	…	B_j	…	B_n
	A_1	α_{11}	α_{12}	…	α_{1j}	…	α_{1n}
	A_2	α_{21}	α_{22}	…	α_{2j}	…	α_{2n}
	…	…	…	…	…	…	…
	A_i	α_{i1}	α_{i2}	…	α_{ij}	…	α_{in}
	…	…	…	…	…	…	…
	A_m	α_{m1}	α_{m2}	…	α_{mj}	…	α_{mn}

Гра в розгорнутій (позиційній) формі (рис.1. 1) задається наступною сукупністю об'єктів:G_R ={T, K_i, t_i^m, і ∈N},де N={1,2,…,n} - множина гравців,Т – дерево гри, K_i – функції виграшу гравців в фінальних позиціях (в кінцевих вузлах (листках) дерева), t_i^m – право ходу гравця і в позиції m.

Дерево гри T={M,f},в якому М – множина позицій (вершин), а функція f:M→ M задає для кожної вершини т попередню їй f(m), причому повинні бути виконані дві умови: 1. початкова позиція:∃!m_0∈M:f (m_0)= m_0; 2. відсутність циклів. Функцію виграшу кожного гравця (кількісну міру) можна представити як відображення множини фінальних позицій M_F на множину дійсних чисел - K_i: $M_F \rightarrow R$. Право ходу (порядок ходів) задається відображенням t_i^m:M_l→ N, де M_l –множина внутрішніх вузлів графа.

Розглянемо подання гри в розгорнутій формі на конкретному прикладі – змоделюємо наші дії на грі з «природою» - «Вибір відпочинку влітку».

Гравцями в цій грі є особисто ви самі, а інший гравець – це оточуюча природа, яка може допомагати у вирішенні вашим цілям, а може й заважати. Ваші цілі в цій грі – обрати один з трьох видів відпочинку: 1) відпочивати вдома; 2) відпочивати на морі; 3) відпочивати в Європі.

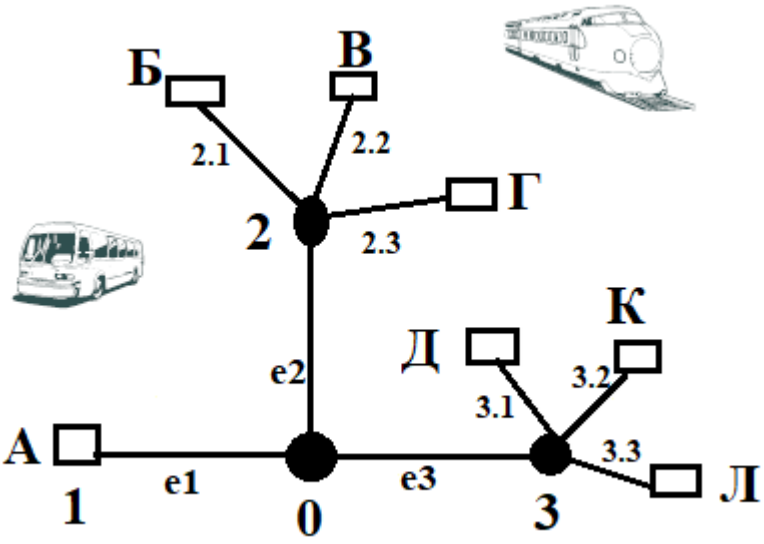

Рисунок 1.1 – Дерево гри «Відпочинок влітку»

Модель гри з природою, що представлена деревом гри (рис. 1. 1) має такі компоненти:

1) Внутрішні вузли дерева – 1, 2, 3 – в яких знаходиться гравець А й його право вибору ходу (вибір стратегії із набору можливих). Наприклад, будучи в вершині 2 гравець може обрати 2.1 чи 2.2 чи 2.3. Його вибір буде означати, чи він обирає подорожувати по країнах Європи автобусом (стратегія 2.1), чи потягом (стратегія 2.2), чи знаходитись в одному й тому же місті.

2) Листки дерева – А, Б, В, Г, Д, К, Л – кінцеві вузли дерева, в яких надаються платежі гравця А за вибір тієї чи іншої стратегії. В даному разі це будуть грошові витрати в подорожі.

3) Стратегії гравця А надані ребрами графа – e1, e2, e3 і вказують його вибір відпочинку влітку: e1 – залишитися вдома й відпочивати на дачі, e2 – відпочивати в Європі, подорожуючи, e3 - відпочивати на морі.

4) «Природа» обирає в цій грі як непередбачувані стратегії, наприклад, в 2020 році обрала стратегію «епідемія коронавірусу», що очевидно вплине на вибір гравця А, так і передбачувані як то вплив на здоров’я, чи дощову погоду, яка не вщухає, що теж впливає вочевидь.

Відмінність гри з природою від гри двох «свідомих» гравців полягає в тому, що в ній раціонально грає тільки один гравець, в більшості випадків зва-

ний гравцем А. Інший учасник гри (природа) грає не раціонально і по-суті не має за мету максимізувати свою корисність у грі. Тому термін «природа» характеризує якусь об'єктивну дійсність, яку не слід розуміти буквально, хоча цілком можуть зустрітися ситуації, в яких «гравцем» В дійсно може бути природа (наприклад, обставини, пов'язані з погодними умовами або з епідеміями).

В подальшому гравців будемо нумерувати, як А та В, та розглянемо скінчену гру, яку зручно представити в нормальній формі матрицею (таблицею) $A=\{a_{ij}\}$, з кількістю рядків m, що дорівнює числу стратегій гравця А ($A_1,A_2,\ldots A_m$) і кількістю стовпчиків n, що дорівнює числу стратегій гравця В ($B_1,B_2,\ldots B_n$); елемент матриці a_{ij} – виграш гравця А при застосуванні ним стратегії A_i, якщо при цьому гравець В застосував стратегію B_j .

Зрозуміло, що матриця гри гравця В буде складатися з тих же елементів, але протилежних по знаку. Будемо вважати, що при $a_{ij}>0$ гравець А виграє, а гравець В програє величину a_{ij}. Якщо $a_{ij}<0$, то, навпаки, виграє гравець В і програє гравець А. В такому разі замість програшу будемо говорити про від'ємний виграш гравця А.

Тому однієї матриці виграшів (табл. 1. 1) гравця А достатньо для представлення скінченої гри.

Введемо поняття рівноваги за Нешем.

Означення. Рівновагою Неша (РН) в грі в нормальній формі називається такий профіль стратегій s* всіх гравців, що

$$K_i(s^*) \geq K_i(s_i,s^*_{-i}),\forall\ i\in N,\forall\ s_i\in S_i\ . \qquad (1.1)$$

За цим означенням, s* – найкраща відповідь на s^*_{-i} для кожного гравця і. Це така ситуація у грі, коли нікому з гравців не вигідно відхилятися від неї, якщо всі інші її дотримуються.

Припустимо спочатку, що гра складена таким чином, що існує її рішення в чистих стратегіях. Спочатку визначимо найкращу з стратегій гравця А, тобто найкращу з A_1, A_2, ..., A_m з урахуванням того, що на будь-яку стратегію A_i гравець В відповість стратегією B_j, мінімізуючи свій програш. Щоб знайти цю

стратегію Bj, треба в рядку платіжної матриці, що відповідає стратегії A_i (рядку з номером i), знайти мінімальне з чисел a_{ij}. Позначимо його, α_i тобто

$$\alpha_i = \min_j a_{ij} \ , j = \overline{1,n},$$

де мінімум визначається шляхом перебору всіх номерів стовпців.

При зміні стратегій гравця А відповідне кожної з цих стратегій число αi теж буде змінюватись. Природно, що гравцю А вигідніше всього зупинитися на такій стратегії Ai, для якої значення αi буде максимальним. Позначимо це максимальне значення α, тобто $\alpha = \max_i \min_j a_{ij}$.

Величину α прийнято називати нижньою ціною гри або максимінним виграшем (скорочено максиміном). Стратегію гравця А, якій відповідає максимін, назвемо максимінною стратегією. Якщо гравець А буде дотримуватися цієї стратегії, то йому при будь-якій поведінці гравця В гарантований виграш, в будь якому разі, не менший ніж α. Тому величину α називають нижній ціною гри, тобто це той гарантований мінімум, який отримає гравець А в цій грі.

Аналогічно можна визначити найкращу з стратегій гравця В. Він прагне мінімізувати свій програш. Тобто він уявляє, що на кожну обрану ним стратегію гравець А обирає для себе найкращу (найгіршу для В) дію, тобто він знаходить значення βj такі, що $\beta_j = \max_i a_{ij} \ , \overline{i = 1,m}$,

де максимум визначається шляхом перебору всіх номерів рядків.

Природно, що гравець В буде обирати стратегію за правилом : $\beta = \min_j \max_i a_{ij}$.

Стратегія гравця В, що забезпечує йому в будь-якому випадку програш не більше β і, відповідно, виграш гравцю А також не більше β. У теорії ігор принцип обережності, що рекомендують гравцям дотримуватися максімінной і мінімаксной стратегії, називається принципом мінімаксу. Він випливає з припущення про обережності гравців або з бажання вирішити конфліктну ситуацію найкращим для всіх що беруть участь у ній сторін чином.

Ситуацію з α та зручно зрозуміти за рис. 1. 2.

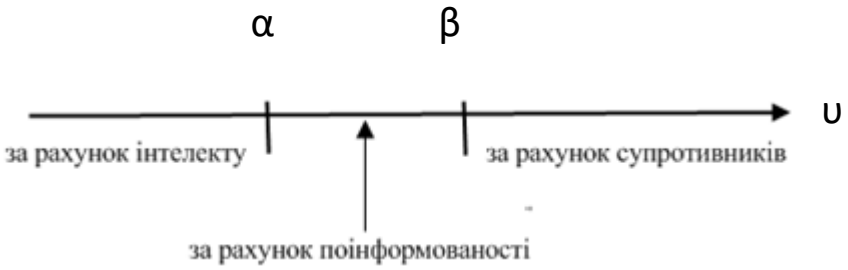

Рисунок 1. 2 – Шкала виграшу гравця А

Цей рисунок э роз'ясненням того, що виграш менше α гравець А одержить тільки тоді, коли буде вести себе неадекватно (нерозумно) або азартно. Якщо у нього є інформація про переваги в діях супротивника, то він може розраховувати до збільшення свого виграшу в інтервалі [α, β], але гравець В має можливість не дати йому виграти більше за β. Виграти більше за β гравець А зможе при нерозумних діях В. Зрозуміло, що при розумних діях А та В виграш гравця А (позначимо цей виграш K_i літерою υ) буде знаходитись в проміжку $\alpha \leq \upsilon \leq \beta$.

Якщо верхня ціна гри збігається з нижньою ціною $\alpha = \beta = \upsilon$, то їх загальне значення називається чистою ціною гри. Пара чистих стратегій дає оптимальне рішення тоді і тільки тоді, коли відповідний їй елемент a_{ij} є одночасно найбільшим в своєму стовпці і найменшим у своєму рядку. Ця пара стратегій A_i та B_j й буде утворювати рівновагу Неша – РН(A_i ,B_j).

В цьому випадку говорять, що гра має сідлову точку.

У загальному випадку при $\alpha \neq \beta$ виявилось, що одержати РН в чистих стратегіях неможливо, потрібно знайти було новий механізм пошуку рівноваги.

Було винайдено механізм багатократного повторення гри, при якому оптимальні рішення досягаються шляхом випадкового застосування гравцями своїх початкових чистих стратегій. Цей механізм дозволив, по-перше, забезпечити найбільшу скритність вибору стратегій (стратегії вибираються випадковим чином); по-друге, при розумній побудові процесу вибору стратегій, останні виявляються оптимальними.

Означення. Випадкова величина, значення якої являють собою стратегії гравця, називають його змішаною стратегією.

Тим самим, завдання змішаної стратегії гравця складається в визначенні ймовірностей, з якими обираються його початкові чисті стратегії. Змішана стратегія гравця *i*– це ймовірнісна міра μ_i на множині його чистих стратегій :

$$\mu_i(s_i) \geq 0, \sum_{s_i \in S_i} \mu_i(s_i) = 1. \qquad (1.2)$$

Так як гравець А має *m* чистих стратегій, то його змішана стратегія $S_A(p_1,p_2,\ldots p_m)$, де p_i - ймовірність з якою обирається стратегія A_i. Аналогічно змішана стратегія гравця В – $S_B\ (q_1,q_2,\ldots q_n)$, де q_j - ймовірність вибору стратегії B_j.

Зауваження 1. Кожна чиста стратегія є частковим випадком змішаної стратегії. Наприклад, чиста стратегія A_i є така $S_A(p_1,p_2,\ldots p_m)$, в якій $p_i=1$, $p_к=0$ ($к \neq i$).

Зауваження 2. В антагоністичній грі без сідлової точки гравець, що грає по визначеній (детермінованій чистій) стратегії, виявляється в гіршому положенні в порівнянні з гравцем, який грає змішані стратегії.

Повернемось до гри, коли гравці грають в змішаних стратегіях. Оскільки матриця виграшів гравця А стає ймовірнісною, тобто кожна ситуація гри $\{A_i, B_к\}$ представляє собою випадкову подію, що відбувається з ймовірністю $p(\{A_i, B_к\}) = p_i \times q_k$, то математичне сподівання виграшу гравця А(середній виграш) в умовах гри в змішаних стратегіях становить

$$E(A,S_A,S_B) = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} p_i q_j, \quad \sum_{i=1}^{m} p_i = 1, \quad \sum_{j=1}^{n} q_j = 1\ . \qquad (1.3)$$

Означення. Стратегії $S^*_A(p^*_1,p^*_2,\ldots p^*_m)$, $S^*_B\ (q^*_1,q^*_2,\ldots q^*_n)$ являються оптимальними змішаними стратегіями гравців А і В в антагоністичній грі , якщо виконується співвідношення:

$$E(A,S_A,S^*_B) \leq E(A,S^*_A,S^*_B) \leq E(A,S^*_A,S^*_B) \qquad (1.4)$$

Величина $\upsilon = E(A,S^*_A,S^*_B)$ називається ціною гри, а набір $\{ S^*_A, S^*_B, \upsilon\}$ – рішенням антагоністичної гри в змішаних стратегіях. Природно, виникає запитання – як знайти $\{ S^*A, S^*B, \upsilon\}$, якщо воно існує?

В [1] доведенs теореми існування рішень, наведемо дві основні теореми (без доведення), що ведуть до визначення { S*A ,S*$_B$, υ}.

Теорема 1. Теорема Неша. Кожна скінчена антагоністична гра має рішення в змішаних стратегіях.

Теорема 2. Теорема про активні стратегії. Якщо один з гравців дотримується своєї оптимальної змішаної стратегії, то його виграш не менше ціни гри, незалежно від того, що робить інший гравець.

За теоремою 2 маємо, що використання оптимальної змішаної стратегії гравцем А має забезпечувати виграш на рівні, не меншому, ніж ціна гри за умови вибору гравцем В будь-яких стратегій. Математично ця умова записується так:

$$\sum_{i=1}^{m} a_{ij} p_i \geq \nu, \quad j = \overline{1,n} \quad . \qquad (1.5)$$

З іншого боку, використання оптимальної змішаної стратегії гравцем В має забезпечувати за будь-яких стратегій гравця А програш, що не перевищує ціну гри u, тобто:

$$\sum_{j=1}^{n} a_{ij} q_j \leq \nu, \quad i = \overline{1,m} \quad . \qquad (1.6)$$

Співвідношення (1.3),(1.5),(1.6) і являють собою математичну модель антагоністичної гри, яка потребує розрахунку.

Зауважимо, що розраховані оптимальні стратегії гравців за даною моделлю будуть стійкими, тобто якщо один з гравців дотримується своєї оптимальної змішаної стратегії, то його виграш буде не меншим за ціну гри незалежно від того, яку із можливих змішаних стратегій обере інший гравець.

Принципово, будь-яка антагоністична гра може бути розв’язана за нерівностями. Але встає питання як розв’язати цю систему?

Якщо стратегій дуже значна кількість, то на допомогу може прийти «принцип домінування», що дозволить зменшити їх кількість.

Означення. Стратегія $A_к$ гравця А строго домінує над стратегією A_l, якщо, $a_{kj}>a_{lj}$ для всіх $j=\overline{1,n}$; стратегія $B_к$ гравця В строго домінує над стратегією B_l ,якщо $a_{ik}<a_{il}$,для всіх $i=\overline{1,m}$.

За цим означенням, кожний раціональний гравець може виключити з матриці гри ті стратегії, що домінуються, таким чином зменшивши розмір матриці для подальшого аналізу.

Зауваження. Оптимальні змішані стратегії в грі з матрицею, одержаній з вихідної за рахунок видалення домінованих стратегій гравців А та В, співпадуть з оптимальним рішенням вхідної задачі; ті чисті стратегії гравців, що в рішенні не беруть участі в кінцевому результаті слід прийняти такими, що дорівнюють нулеві.

Існує також поняття не строгого домінування (слабе домінування), коли в означені знак «більше», «менше» заміняється «не менше» і «не більше».

Зауваження. Виключення домінованих (не строго) стратегій може привести до втрати деяких рішень. Якщо виключати тільки строго домінові стратегії, то множина рішень не зміниться. (Зауваження стосується більше щодо біматричних ігор, що будуть розглядатись далі).

В перетвореннях вихідної матриці гри корисним також є правило, що формулюється нижче.

Афінне правило. Оптимальні стратегії гравців у матричних іграх, елементи матриць А та С яких пов'язані рівностями :

$c_{ik}=\lambda a_{ik}+\mu, \quad i=\overline{1,m}, \quad k=\overline{1,n}$, де $\lambda>0$, μ-довільне,

мають однакові рівновісні ситуації (або в чистих, або в змішаних стратегіях), а їх ціни задовольняють слідкуючій умові:

$\nu_c=\lambda\nu_A+\mu$.

Підсумовуючи вище приведене, маємо наступний алгоритм рішення матричної гри:

- Спрощення платіжної матриці шляхом виключення домінованих стратегій, якщо такі маємо.

- Визначення нижньої ціни α, верхньої ціни β, і відповідно перевірка існування сідлової точки. Якщо $\alpha = \beta$, то загальне значення є ціною гри ν, рішення в чистих стратегіях знайдено, гра вирішена.
- Перевірка умов застосовності змішаних стратегій (повторюваності гри й того, що гравцям відома загальна інформація щодо гри). При невиконанні хоча б однієї з умов рішення гри відповідає знайденому в п.2, тобто maxmin й minmax, як гарантованим виграшу й програшу гравців.
- Знаходимо рішення в змішаних стратегіях за одним із способів, що приведемо нижче.

Проаналізувавши дослідження в теорії ігор на основі [1-8], встановлено наступне:

- ✓ Головною проблемою є перехід від реальної конфліктної ситуації, що досліджується до моделі теорії ігор, що її буде описувати.
- ✓ В подальшому потрібно описати обрану модель гри математично (скласти математичну модель гри).
- ✓ Розробити алгоритм та комп'ютерну реалізацію для вирішення складеної математичної постановки задачі.
- ✓ Одержати розрахунки конфлікту та проаналізувати їх на предмет розробки рекомендацій для конфліктуючих сторін.

1.2 Постановка задачі

Поставимо наступне завдання для дослідження.

Для заданої матричної гри знайти виграші гравців та визначити які оптимальні стратегії їм застосовувати для вирішення цього. Для досягнення мети знайти рішення підзадач:

1. *Створити комп'ютерну реалізацію алгоритму Монте-Карло.*
2. *Створити комп'ютерну реалізацію ітераційного алгоритму.*
3. *Провести порівняльний аналіз цих алгоритмів на тестових завданнях.*

2 МАТЕМАТИЧНА ПОСТАНОВКА ЗАВДАННЯ ТА ВИБІР МЕТОДУ ЇЇ РІШЕННЯ

2.1 Короткий огляд відомих рішень

Загальний алгоритм рішення матричної гри, представлений в п. 1.1 реалізований в багатьох моделях з використанням як перевірених способів, як то приведенням задачі до задачі лінійного програмування з подальшим застосуванням симплекс-методу, так і новітніх реалізаціях, з застосуваннях чисельних методів та еволюційних алгоритмів.

Представимо декілька варіантів рішення.

Покажемо, що модель будь-якої матричної гри може бути зведена до задачі лінійного програмування.

Будемо вважати, що платіжна матриця гри $A=\{a_{ij}\}$ нам відома, причому $a_{ij} \geq 0$ (цього можна домогтися за афінним правилом (див. п. 1. 1)). Оптимальні стратегії гравців будемо знаходити в змішаних стратегіях : $S^*_A(p_1,p_2,\ldots,p_m)$; $S^*_B(q_1,q_2,\ldots,q_n)$.

Розглянемо гру спочатку з позицій гравця А. Впровадження ним оптимальної стратегія повинно забезпечувати йому виграш не менше ν при будь-якому поводженні гравця В та виграш, рівний ν, при оптимальному поводженні гравця В.

Припустимо далі, що гравець А застосовує свою оптимальну змішану стратегію S^*_A (поки нам невідому), а гравець В – тільки свою будь-яку чисту стратегію B_j. Тоді середній виграш гравця А складе:

$$a_j = p_1 a_{1j} + p_2 a_{2j} + \ldots + p_m a_{mj}$$

Але кожне із чисел a_j не може бути менше ν відповідно до теореми теорії ігор про активні стратегії (див. п. 1. 1). Звідси одержуємо в загальному випадку n нерівностей (див. 2. 1)

Ця система нерівностей визначає математичну модель матричної гри, рішення якої приведе до шуканого результату.

$$\begin{cases} p_1 a_{11} + p_2 a_{21} + ... + p_m a_{m1} \geq \nu, \\ p_1 a_{12} + p_2 a_{22} + ... + p_m a_{m2} \geq \nu, \\ ..... \\ p_1 a_{1n} + p_2 a_{2n} + ... + p_m a_{mn} \geq \nu, \end{cases} \quad (2.\ 1)$$

Уведемо нові позначення: $x_i = p_i / \nu$.

Тоді після ділення всіх нерівностей на ν одержимо:

$$\begin{cases} x_1 a_{11} + x_2 a_{21} + ... + x_m a_{m1} \geq 1, \\ x_1 a_{12} + x_2 a_{22} + ... + x_m a_{m2} \geq 1, \\ ..... \\ x_1 a_{1n} + x_2 a_{2n} + ... + x_m a_{mn} \geq 1, \end{cases} \quad (2.\ 2)$$

де за умовою всі x_i – додатні змінні, що задовольняють умові: $x_1 + x_2 + ... + x_m = 1/\nu$

Гравець А прагне зробити свій гарантований виграш ν максимальним. Це значить, що величина $1/\nu$ повинна прийняти мінімальне значення. Таким чином, моделювання гри звелося до наступної задачі лінійного програмування: визначити невід'ємні значення змінних x_1, x_2, …, x_m, які надають функції цілі мінімального значення:

$$z = x_1 + x_2 + ... + x_m = 1/\nu \to \min \quad (2.\ 3)$$

за умови, що виконуються зазначені обмеження (2. 1).

Розглянемо тепер гру з позицій гравця В. Впровадження ним оптимальної стратегії повинно забезпечувати йому програш не більший за ν при будь-якому поводженні гравця А та програш, рівний ν, при оптимальному поводженні гравця А.

В результаті аналогічних математичних перетворень одержимо наступну модель:

$$\begin{cases} y_1 a_{11} + y_2 a_{12} + ... + y_n a_{1n} \le 1, \\ y_1 a_{21} + y_2 a_{22} + ... + y_n a_{2n} \le 1, \\ ..... \\ y_1 a_{m1} + y_2 a_{m2} + ... + y_n a_{mn} \le 1, \end{cases}$$
$$\bar{z} = y_1 + y_2 + ... + y_n = 1/\nu \to \max \qquad (2.\ 4)$$

Де введені позначення: $y_j = q_j / \nu$.

Замість вимоги мінімізації функції тепер використається вимога максимізації функції цілі, тому що гравець В на відміну від гравця А прагне мінімізувати ν (програти якнайменше).

Отже, рішення гри m×n звелося до рішення двоїстої задачі лінійного програмування. Відповідно до теорем теорії ігор, це рішення завжди існує.

2.2 Чисельне рішення методом Монте-Карло

Метод Монте-Карло (в подальшому МС) відносять до чисельних методів рішення проблем в різноманітних сферах. Його ще часто називають метод статистичних випробувань, бо він тісно пов'язаний з теорією ймовірностей й опирається на Закон великих чисел, що є одним з її основних прикладних законів.

Незважаючи на простоту, він потужний і має деякі цікаві властивості, які роблять його дуже привабливим для вирішення різних завдань, включаючи життєві проблеми (див. рис. 2.1).

Рисунок 2. 1 - Застосування методу Монте-Карло в спортивних змаганнях при жеребкуванні

Розглянемо простий приклад задля показу загальної концепції застосування МС. Нехай нам потрібно обчислити площу криволінійної фігури, поданої рисунком 2.1. Заключимо фігуру в рамки прямокутника, як це демонструється

на рисунку. Оскільки границі прямокутника відомі, площу цього прямокутника неважко знайти –

$$S = ab \times ac.$$

А надалі включаємо випадковість на допомогу – накидаємо на область прямокутника наперед задану достатньо значну кількість точок. Деякі з них потраплять, окрім прямокутника одночасно й в область криволінійної фігури.

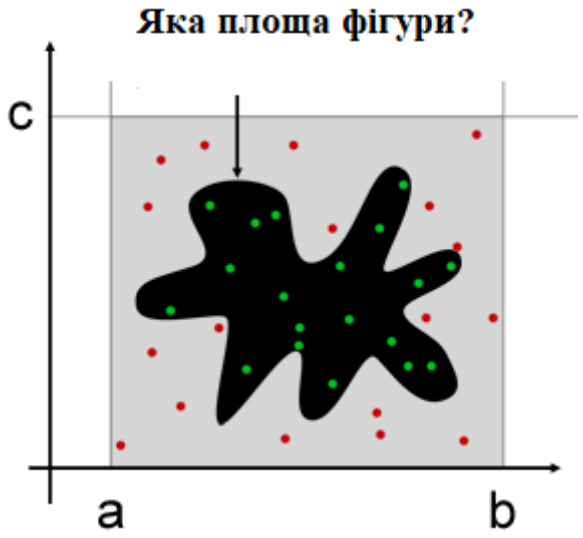

Рисунок 2. 2 – Застосування методу Монте-Карло для визначення площі

Підрахуємо кількість таких точок, що потрапляють в область криволінійної фігури з загальної кількості N і позначимо її літерою N_1. Надалі напишемо наближену формулу

$N_1 / N \approx S_1 / S$, де S_1 – шукана площа криволінійної фігури. З останньої формули й знайдемо S_1.

Загальну схему застосовності алгоритму Монте-Карло можна подати наступне (див. рис. 2. 3).

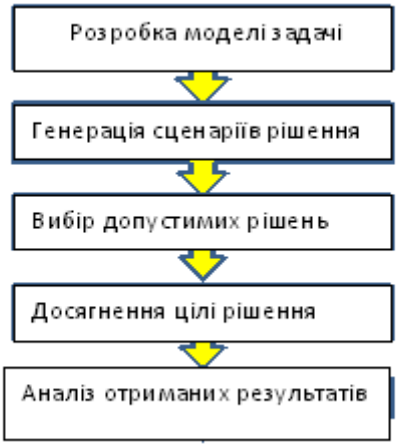

Рисунок 2. 3 – Загальна схема застосовності алгоритму Монте-Карло

Сучасний стан розвитку високошвидкісних цифрових комп'ютерів дозволяє використовувати вибірки досить великого розміру, щоб забезпечити задовільну точність в більшості практичних задач.

Це важливо розуміти, тому що саме по собі, будучи досить простий ідеєю, використовувати МС без допомоги комп'ютера досить утомливо, якщо не сказати непридатно до вирішення будь-яких проблем. Комп'ютер може виконати всі розрахунки для нас, тому, незважаючи на свою низьку швидкість збіжності, МС набрав популярності. Ми просто дозволяємо комп'ютерів виконувати тяжку роботу за нас.

Адаптуємо МС для рішення моделей теорії ігор.

Використання МС для задач теорії ігор є природним, так як він по своїй суті сам нібито розігрує гру, моделюючи випадковим чином той чи інший сценарій гри. За загальним сценарієм, МС нібито розігрує варіанти гри, обираючи за гравців випадковим чином стратегії з множини можливих стратегій.

Наприклад, для гравців А та В ми розігруємо деякий сценарій гри, генеруючи випадкові змішані їх стратегії p і q (випадкові числа з інтервалу [0, 1]). Підставляючи цю пару до системи (2.2) ми відсіюємо ті, що нам не підходять (не задовольняють нерівностям), а з тих, що задовольняють всі нерівності системи, обираємо рішення таке, щоб функція цілі (2.5) та (2.6) мали найбільші значення. Це й буде рішенням гри. Будь яку гру можна розрахувати цим методом з заданої точністю обчислення ε.

$$W(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}\ p_i q_j \rightarrow \max \qquad (2.5)$$

$$W(B) = \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij}\ p_i q_j \rightarrow \max \qquad (2.6)$$

Суть адаптації методу Монте-Карло для вирішення поставленої задачі полягає в наступному: для задоволення екстренуму цільових функцій (2.5), (2.6) генеруємо випадкової величини з інтервалу(0,1), задовольняємо систему нерівностей (2.2), а потім на їх основі розраховуємо необхідні значення, що надають функціям цілі найбільших значень.

Нарешті, завершаючи адаптацію МС до розв'язку моделей ігор повернемось до питання генерації випадкових чисел. Щоб запустити алгоритм МС, нам спочатку потрібно мати можливість генерувати випадкові числа (зазвичай із заданим розподілом ймовірності). З цієї причини розробка алгоритмів для генерації таких «випадкових» чисел (вони здаються випадковими, але, як правило, вони не є «по-справжньому» випадковими, тому ці алгоритми називають генератором псевдовипадкових чисел), є важливою областю досліджень в області обчислювальної техніки.

Так як для комп'ютерної реалізації алгоритмів обрано мову програмування Python, означимо тонкості застосувань МС на мові Python.

Python, одна з найбільш застосовних мов для МС, породжує випадкові числа на основі формули, так що вони не насправді випадкові, а, як кажуть, псевдовипадкові [6]. Цей спосіб зручний для більшості додатків.

Основні процедури:

random.seed([X], version=2) - ініціалізація генератора випадкових чисел. Якщо X не вказано, використовується системний час;

random.getstate () - внутрішній стан генератора;

random.setstate (state) - відновлює внутрішній стан генератора. Параметр state повинен бути отриманий функцією getstate ();

random.randrange (start, stop, step) - повертає випадково вибране число з послідовності;

random.randint (A, B) - випадкове ціле число N, $A \leq N \leq B$;

random.random () - випадкове число від 0 до 1;

random.uniform (A, B) - випадкове число з плаваючою точкою, $A \leq N \leq B$ (або $B \leq N \leq A$)4

random.normalvariate (mu, sigma) - нормальний розподіл. mu - середнє значення, sigma - стандартне відхилення.

Якщо задатись питанням про точність методу Монте-Карло, то чим більше моделювань ситуацій гри використовується, то тим ближче метод МС підходить до реального вирішення, оскільки ми використовуємо випадкові вибірки, метод МС також може «просто» випадковим чином натрапити на точне рішення по чистій випадковості. Проте, в більшості випадків це не так, але усереднення всіх результатів симуляцій гри все одно призведе до точного рішення (про це хоча б говорить Закон великих чисел теорії ймовірностей).

2.3 Ітераційні методи рішення

Опишемо один з відомих в теорії ігор чисельних методів рішення – так званий метод ітерацій (інакше – метод Брауна - Робінсона), що відображує деяку «реальну ситуацію накопичення досвіду» в результаті багатьох повторень конфлікту [1-3].

Ідея його в наступному – розігрується "уявна гра", в якій гравці (нехай це будуть А та В) по черзі обирають друг проти друга свої стратегії, прагнучи виграти побільше (програти поменше). Кожен з гравців слідкує за діями супротивника і старається відповісти на них найбільш вигідним для себе чином.

Гра імітується з ряду "партій". Починається вона з того, що один із гравців (скажімо, А) робить свій вибір, допустимо він грає стратегію A_i.

Перший хід гравцю А доцільно вибрати за принципом $\mathrm{Arg}_i\{\max_i \min_j (a_{ij})\}$ (обираємо той рядок, що забезпечує нижню ціну гри). Супротивник (В) відповідає на цей хід своїм вибором – він обирає зі своїх стратегій B_j , яка найвигідніша для нього, тобто обертає свій програш при стратегії A_i в мінімум – $\mathrm{Arg}_j \{ \min_j (a_{ij})\}$ (обирає j – стовпчик, що забезпечує мінімальний програш при вибраній гравцем А стратегії).

Надалі настає черга А реагувати на дії гравця В – гравець А відповідає гравцеві В такою своєю стратегією A_k, яка дає максимальний виграш при стратегії B_j гравця В – $Arg_i \{ \max_i (a_{ij})\}$.

Надалі – знову черга супротивника. Він відповідає своєю стратегією, яка мінімізує програш вже не відносно останньої, застосованої гравцем А стратегії A_k, а відносно накопичуваного виграшу гравця А – $a_{k1}' = a_{i1} + a_{к1}$,$a_{k2}' = a_{i2} + a_{к2}$, … ,$a_{км}' = a_{ім} + a_{км}$, як сумарного виграшу при застосуванні стратегій A_i, A_k (гравець В обирає стовпчик $j = Arg_j \{ \min_j (a_{ij}')\}$).

І так далі, на кожному кроці ітераційного процесу кожен гравець відповідає на черговий хід іншого своєю стратегією, яка є оптимальною щодо накопичуваного виграшу (програшу).

Хід гри зручно відобразити в таблиці (див. табл. 2. 1). В таблиці представлені, окрім описаних, також нижня оцінка ціни гри U, що обчислюється як найменший середній виграш гравця А : $U_N = \frac{\min_k a_{ik}'}{N}$, а також верхня оцінка ціни гри W, як найбільший середній програш гравця В : $W_N = \frac{\max_k a_{ki}'}{N}$, ν =(U+W)/2 - наближена оцінка ціни гри, як середнє арифметичне між U та W.

Таблиця 2. 1 Ітераційний метод Брауна – Робінсона

№ гри	Стратегія гравця А	Накопичуваний виграш гравця А				Стратегія гравця В	Накопичуваний програш гравця В				U	W	ν
1	A_i	a_{i1}	a_{i2}	…	a_{im}	B_j	a_{1j}	a_{2j}	…	a_{nj}	U_1	W_1	ν_1
2	A_k	a_{k1}'	a_{k2}'	…	a_{km}'	B^l	a_{1l}'	a_{2l}'	…	a_{nl}'	U_2	W_2	ν_2
…	…	…	…	…	…	…	…	…	…	…	…	…	…
N	A_p	a_{p1}'	a_{p2}'	…	a_{pm}'	B_t	a_{1t}'	$a_{2t}/$	…	a_{nt}'	U_N	W_N	ν_N

Сформулюємо правило зупину ітераційного процесу.

Нехай задано число ε >0. Процес ітерацій зупинимо на кроці n, коли вперше виконається нерівність $W(n) - U(n) \le \varepsilon$.

В цьому разі ціна гри стане рівною $\nu \approx (W(n) + U(n))/2$, *змішані* стратегії гравців $S_A^*(p_1^*, p_2^*,..., p_m^*), S_B^*(q_1^*, q_2^*,..., q_n^*)$ знаходимо за формулами статистичного визначення ймовірностей :

$$p_1^* = \frac{n_{A_1}}{N}, \quad p_2^* = \frac{n_{A_2}}{N},..., \quad p_m^* = \frac{n_{A_m}}{N},$$
$$q_1^* = \frac{n_{B_1}}{N}, \quad q_2^* = \frac{n_{B_2}}{N},..., \quad q_n^* = \frac{n_{B_n}}{N}, \qquad (2.11)$$

де $n_{A_i}, \quad n_{B_j}$ - загальна кількість використання гравцем А стратегії A_i та відповідно гравцем В стратегії B_j .

Сам алгоритм ітераційного процесу запропонував Браун, але Робінсон довів ключову теорему, яка є основою методу, тому його прізвище теж ввійшло до назви методу.

Теорема Робінсона. В методі Брауна $\lim_{k\to\infty} U(k) = \lim_{k\to\infty} W(k) = \upsilon$, а будь-які граничні значення p*, q* послідовностей {p(k)}, {q(k)} являють собою оптимальні змішані стратегії гравців.

Доведення теореми можна знайти в [1]. Там же вказано, що швидкість збігання послідовностей {U(k)}, {W(k)} до значення ціни гри ν можна оцінити як O((1/k)1/(m+n-1)). Практично швидкість збігання суттєво вища.

✍ Розглянемо приклад . Нехай гра задана платіжною матрицею

$$A = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 0 & 3 \\ -1 & 3 & -3 \end{pmatrix}$$

. Знайти оптимальні стратегії гравців та ціну гри.

Рішення. Маємо гру без сідлової точки, бо α =0, β=2. Тобто $0 \le \nu \le 2$.

Проведемо рішення ітераційним методом Брауна - Робінсона (див. табл. 2. 1). Процес вибору стратегій демонструє рис. 2.2.

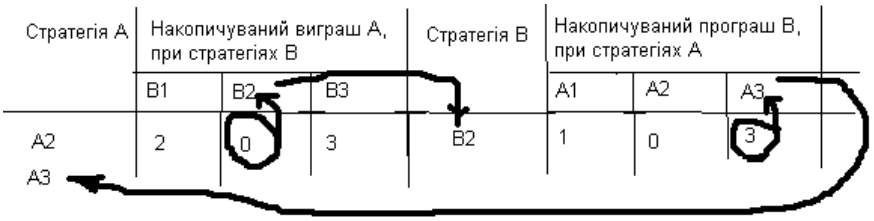

Рис. 2. 4 – Вибір гравцями стратегій на першому кроці

Таблиця 2. 2 Таблиця ітерацій методу Брауна-Робінсона

№ гри	Стратегія гравця А	Накопичуваний виграш гравця А, при стратегіях В			Стратегія гравця В	Накопичуваний програш гравця В, при стратегіях А			U	W	ν
		B_1	B_2	B_3		A_1	A_2	A_3			
1	A_2	2	0	3	B_2	1	0	3	0	3	1.5
2	A_3	1	3	0	B_3	1	3	0	0	1.5	0.75
3	A_2	3	3	3	B_3	1	6	-3	1	2	1.5
4	A_2	5	3	6	B_2	2	6	0	0.75	1.5	1.175
5	A_2	7	3	9	B_2	3	6	3	0.6	1.2	0.9
6	A_2	9	3	12	B_2	4	6	6	0.5	1	0.75
7	A_2	11	3	15	B_2	5	6	9	0.43	1.3	0.86
8	A_3	10	6	12	B_2	6	6	12	0.75	1.5	1.175
9	A_3	9	9	9	B_3	6	9	9	1	1	1

Знаходимо відповідь: ν=1, p1=0, p2=6/9=2/3, p3 =3/9=1/3, q1=0, q2=2/3, q3=1/3.

3 КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ ТА ТЕСТОВІ РОЗРАХУНКИ

3.1Вибір мови програмування

Вибір мови програмування для комп'ютерної реалізації проекту опирається на відповіді до запитань: 1) Чи є ця мова рейтинговою і бажано об'єктно орієнтованою? 2) Чи передбачені в цій мові операції над тими структурами даних, які будуть використовуватись в комп'ютерній реалізації? 3) Чи передбачені в цій мові реалізації парадигм програмування, які використовуються в алгоритмах? У програмуванні головне – зрозуміти принцип вирішення завдань і складання алгоритмів, а не знання конкретних команд мови. Справа в тому, що не потрібно витрачати час на розуміння принципів, і залишиться тільки розібратися, як потрібна нам річ реалізована в конкретній мові.

Наш вибір – це мова програмування Python [6]. Чому саме вона?

Звернемось до світового рейтингу мов на 2020 рік (див. рис. 3.1). В цьому рейтингу Python знаходиться на 4 місці.

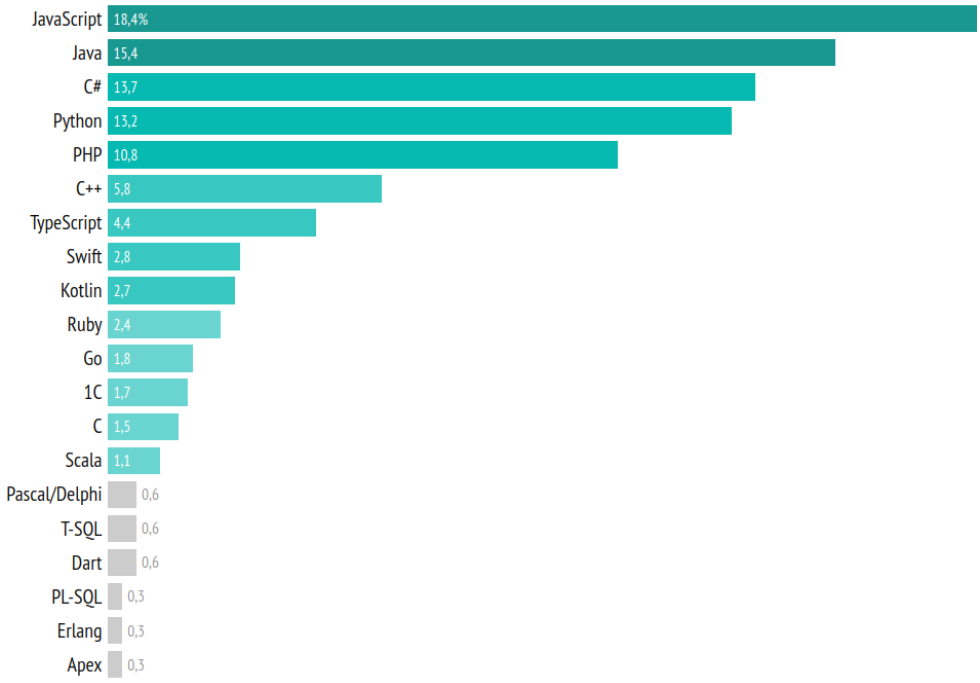

Рисунок 3.1 – Світовий рейтинг мов програмування 2020 року

Але, якщо використати індекс популярності мови програмування (PYPL), який створюється шляхом аналізу частоти пошуку навчальних посібників з мови в Google, то тут картина на нашу користь (див. табл. 3.1).

Чим більше підручник з мови шукається, тим більш популярним вважається мову. Це провідний індикатор. Необроблені дані надходять з Google Trends. Якщо ви вірите в колективну мудрість, індекс популярності мови програмування PYPL допоможе вам вирішити, яку мову вивчати або який використовувати в проекті, який розробляється.

Таблиця 3.1 Індекс популярності мов програмування PYPL

Worldwide, Jun 2020 compared to a year ago:

Rank	Language	Share	Trend
1	Python	31.6 %	+4.3 %
2	Java	17.67 %	-2.4 %
3	Javascript	8.02 %	-0.2 %
4	C#	6.87 %	-0.4 %
5	PHP	6.02 %	-0.9 %

Python в даний час є найбільш популярною мовою для викладання вступних курсів з інформатики в провідних американських департаментах. Зокрема, вісім з 10 кращих відділів CS (80%) і 27 з 39 кращих (69%) викладають Python на вступних курсах CS0 або CS1.

Python підтримує кілька парадигм програмування, включаючи структурованість (зокрема, процедурність), об'єктно-орієнтованість і функціональне програмування. Python часто описується як мова «з батарейками» через його обширну стандартну бібліотеку, в якій зібрані операції над багатьма структурами даних, включаючи ті структури даних, на яких ми записуємо наші числові алгоритми.

Python володіє чітким і послідовним синтаксисом, продуманою модульністю та масштабованістю, завдяки чому вихідний код написаних на Python програм зручний й не перенапружений.

І основна зручність, яка реалізовується для досліджуваних алгоритмів – це можливості стандартної бібліотеки Python в реалізації чисельних методів розрахунків (в п. 2.2 ми частково описали операції для реалізації методу МС).

3.2 Комп'ютерна реалізація алгоритмів

3.2.1. Алгоритм Монте – Карло

Таким чином, сутність методу Монте-Карло в застосувань до задач теорії ігор буде полягати в тому, що замість аналітичного рішення моделі гри (наприклад, тим же симплекс-методом), ми будемо здійснювати значну серію "розіграшів" гри, як випадкового процесу, який відбувається шляхом спеціально організованої процедури, в основі якої лежить генератор випадкових чисел.

В результаті такого "розіграшу" здійснюється кожного разу нова, відмінна від інших реалізація ситуація в грі. Цю множину реалізацій можна використати як деякий штучно отриманий статистичний матеріал, з якого ми й обираємо ту реалізацію, яка найкращим чином нам піддійте. Цільовою функцією будуть слугувати виграші гравців

$$W(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}\ p_i q_j \rightarrow \max \qquad (3.\ 1)$$

$$W(B) = \sum_{i=1}^{m} \sum_{j=1}^{n} b_{ij}\ p_i q_j \rightarrow \max \qquad (3.\ 2)$$

Програмна реалізація такого алгоритму наступна

```
for i in range(N_iterations):
new_random_x = np.random.uniform(0, 1/max_min,
size=(matrix.shape[0],))
if not are_ineq_satisfied(matrix, new_random_x, lambda x_dot_a:
x_dot_a
>=1 ):
continue
L = np.sum(new_random_x)
```

```
if L_min > L:
L_min = L
values = new_random_x
print("L:")
print(L_min)
print("x_vals:")
print(values)
print("V:")
v = 1 / L_min
print(v)
print("p_vals:")
print(values * v)
```

Використання методу Монте-Карло – чисельного методу рішення є природним, так як він по своїй суті сам нібито розігрує гру, моделюючи випадковим чином той чи інший сценарій гри. Наприклад, для гравця А за нерівностями (2. 2) та функцією (2. 3), генеруємо випадкові числа x_i з інтервалу [0, 1/α]. Підставляючи ці числа до системи (2. 2) ми відсіюємо ті, що нам не підходять (не задовольняють нерівностям), а з тих, що задовольняють всі нерівності системи, обираємо рішення таке, щоб функція цілі (2. 3) мала мінімальне значення. Це й буде рішенням гри.

Будь яку гру можна розрахувати цим методом з заданої точністю обчислення ε.

3.2.2. Ітераційні алгоритми

Для застосування алгоритмів ітераційної парадигми, повернемось до математичної моделі (див. п. 2.1)

$$\begin{cases} p_1 a_{11} + p_2 a_{21} + \ldots + p_m a_{m1} \geq v, \\ p_1 a_{12} + p_2 a_{22} + \ldots + p_m a_{m2} \geq v, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ p_1 a_{1n} + p_2 a_{2n} + \ldots + p_m a_{mn} \geq v, \end{cases} \qquad (3.\ 3)$$

В цій системі нерівностей: коефіцієнти $\{a_{ij}\}$ – платіжна матриця гри, υ – виграш, який має нижню та верхню границі: $\alpha \leq \upsilon \leq \beta$, а p_i – змішані стратегії гравця. Так як гравець прагне отримати максимально можливий виграш у грі, то побудуємо наступний алгоритм його дій.

Алгоритм простого перебору.

1) Приймемо $\upsilon = \beta$.
2) Підставимо поточне значення υ в систему (3. 3), та шляхом простого перебору на полі допустимих значень p_i ($\sum_{i=1}^{n} p_i$) перевіряємо виконуваність системи. Якщо так, то той набір p_i є рішенням для гравця, а поточне значення υ є його виграшом. Кінець алгоритму. Рішення отримане.
3) Якщо ні, то переходимо до п. 3.
4) Відступимо на крок $\Delta\beta$ (його можна обрати заздалегідь), тобто поточне значення виграшу приймемо $\upsilon = \beta - \Delta\beta$. Якщо $\beta \geq \alpha$ переходимо до п. 2, якщо ні – кінець алгоритму. Рішення не можливе.

Програмна реалізація такого алгоритму наступна

```
while alpha <= v <= beta:
n_iterations = 2000000
        for iter_index in range(n_iterations):
        p_random_vals = build_random_p_values(matrix.shape[0])
        if not are_ineq_satisfied(matrix, p_random_vals, lambda p_dot_a:
p_dot_a >= v):
continue
return p_random_vals, v
        v -= main_itr_step
        def get_max_min(game_matrix):
        rows_mins = np.min(game_matrix, axis=1)
        return np.max(rows_mins)


def get_min_max(game_matrix):
```

```
        columns_maxs = np.max(game_matrix, axis=0)


def build_random_p_values(size):
        result = np.zeros(shape=size)
        high_border = 1.0
        for i in range(size-1):
        new_random_p = np.random.uniform(0, high_border, 1)
        result[i] = new_random_p
        high_border -= new_random_p
        result[-1] = high_border
        return result
```

Алгоритм Брауна-Робінсона(див. п. 2. 3).

Програмна реалізація такого алгоритму наступна

```
def get_curr_cumm_vector(matrix, old_vector, next_index, is_row=True):
        next_vector = None
        if is_row:
        next_vector = matrix[next_index]
        else:
        next_vector = matrix[:, next_index]
        return next_vector + old_vector
        scale = 0.01
        cost_matrix = get_cost_matrix("matrix")
        print(cost_matrix)
        start_index = get_start_strat_index(cost_matrix)
        first_strategies = list()
        second_strategies = list()
        cumm_first = np.zeros(cost_matrix.shape[1], dtype=cost_matrix.dtype)
        cumm_second = np.zeros(cost_matrix.shape[0],
dtype=cost_matrix.dtype)
        next_first_srtategy = start_index
```

```
next_second_strategy = None
W = None
U = None
counter = 1
while True:
first_strategies.append(next_first_srtategy)
cumm_first = get_curr_cumm_vector(cost_matrix, cumm_first,
next_first_srtategy)
next_second_strategy = np.argmin(cumm_first)
second_strategies.append(next_second_strategy)
cumm_second = get_curr_cumm_vector(cost_matrix, cumm_second,
next_second_strategy,False)
next_first_srtategy = np.argmax(cumm_second)
U = cumm_first.min()/counter
W = cumm_second.max()/counter
if W - U < scale:
break
counter += 1
v = (W+U)/2
_, cnts = np.unique(first_strategies, return_counts=True)
for i in range(len(cnts)):
print("p_{} = {}".format(i+1, cnts[i] / len(first_strategies)))
print("\n\n")
_, cnts = np.unique(second_strategies, return_counts=True)
for i in range(len(cnts)):
print("q")
```

3.3 Тестові розрахунки

Проведемо тестові розрахунки та порівняємо результати розрахунків. Задамо вхідну платіжну матрицю гри.

	B			
A	6	9	11	6
	13	6	8	10
	9	7	6	12
	6	11	8	7
	11	8	7	9

Рисунок 3. 2 – Матриця гри тестового розрахунку

Проведемо попередні дослідження: MAX_MIN = 7 MIN_MAX = 11

Сідлова точка відсутня, тому тому зводимо да задачі лінійного програмування в змішаних стратегіях.

$S_A = (p_1, p_2, p_3, p_4, p_5)$. $S_B = (q_1, q_2, q_3, q_4)$.

Математична модель для гравця А:

$$L(x)=x_1+x_2+ x_3+x_4+x_5 \rightarrow \min$$
$$6\ x_1+13\ x_2+9\ x_3+6\ x_4+11\ x_5 \geq 1$$
$$9\ x_1+6\ x_2+7\ x_3+11\ x_4+8x_5 \geq 1$$
$$11\ x_1+8\ x_2+6\ x_3+8x_4+7\ x_5 \geq 1$$
$$6\ x_1+10\ x_2+12\ x_3+7x_4+9\ x_5 \geq 1$$
$$x_i \geq 0$$

Математична модель для гравця В:

$$L(y)=y_1+ y_2+ y_3+ y_4 \rightarrow \max$$
$$6\ y_1+ 9y_2+11\ y_3+6\ y_4 \leq 1$$
$$13\ y_1+6\ y_2+8\ y_3+10\ y_4 \leq 1$$
$$9\ y_1+7\ y_2+6\ y_3+12y_4 \leq 1$$
$$6\ y_1+11\ y_2+8\ y_3+7y_4 \leq 1$$
$$11\ y_1+ 8\ y_2+ 7\ y_3+ 9y_4 \leq 1$$
$$y_i \geq 0$$

Вхідний файл містить матрицю :

6 9 11 6

13 6 8 10

9 7 6 12

6 11 8 7

11 8 7 9

Результати розрахунків представимо таблицею 3.2. Для оцінювання точності рішення в таблиці дані також результати за симплекс-методом.

Таблиця 3.2 Розрахунки тестового прикладу

Метод Монте-Карло	Метод Брауна-Робінсона	Метод направле-ного перебору	Симплекс-метод
$\upsilon = 8{,}36$	$\upsilon = 8{,}425$	$\upsilon = 8{,}40$	$\upsilon = 8,\ 427$
$p_1 = 0{,}258$ $p_2 = 0{,}252$ $p_3 = 0{,}146$ $p_4 = 0{,}252$ $p_5 = 0{,}092$	$p_1 = 0{,}262$ $p_2 = 0{,}273$ $p_3 = 0{,}172$ $p_4 = 0{,}293$ $p_5 = 0$	$p_1 = 0{,}258$ $p_2 = 0{,}282$ $p_3 = 0{,}16$ $p_4 = 0{,}29$ $p_5 = 0{,}01$	$p_1 = 0{,}26$ $p_2 = 0{,}27$ $p_3 = 0{,}17$ $p_4 = 0{,}30$ $p_5 = 0$
$q_1 = 0{,}072$ $q_2 = 0{,}293$ $q_3 = 0{,}325$ $q_4 = 0{,}310$	$q_1 = 0{,}081$ $q_2 = 0{,}301$ $q_3 = 0{,}304$ $q_4 = 0{,}314$	$q_1 = 0{,}073$ $q_2 = 0{,}302$ $q_3 = 0{,}312$ $q_4 = 0{,}313$	$q_1 = 0{,}08$ $q_2 = 0{,}3$ $q_3 = 0{,}31$ $q_4 = 0{,}31$

Порівняльний аналіз одержаних результатів.

1) *Всі чисельні методи з точністю до 0,1 надають однакові результати.*
2) *Метод Брауна-Робінсона найбільш наблизився до рішення, одержаного за симплекс-методом.*
3) *Для гравця А невигідно грати стратегію 5, про це говорить той факт, що* $p_5 = 0$.
4) *Для гравця В невигідно грати стратегію 1, про це говорить той факт, що* $q_1 = 0{,}08$.

ВИСНОВКИ

В ході виконання випускної роботи було досліджено застосування чисельних методів розв’язання моделей теорії ігор, заданих платіжними матрицями.

Результатами виконання є наступне:

1) Адаптовано метод Монте-Карло для розв’язку гри та виконана йото комп’ютерна реалізація.
2) Виконана комп’ютерна реалізація ітераційних методів розв’язку.
3) Порівняльний аналіз алгоритмів показав, що рішення задач збігаються, по часовим витратам методи мають однакові показники.

СПИСОК ЛІТЕРАТУРИ

1. Corchyn L. C., Marini M. A. Handbook of Game Theory and Industrial Organization,Volume I, Theory. – Edward Elgar Publishing Limited, 2018. – 533 p.
2. Corchyn L. C., Marini M., A. Handbook of Game Theory and Industrial Organization,Volume II, Applications. – Edward Elgar Publishing Limited, 2018. – 533 p.
3. Frahm G. Rational Choice and Strategic Conflict. The Subjectivistic Approach to Game and Decision Theory. – Berlin/Boston: « Walter de Gruyter GmbH», 2019. – 340 p.
4. Bauso D. Game Theory: Models, Numerical Methods and Applications. Foundations and Trends in Systems and Control, vol. 1, no. 4, 2014. - pp. 379–522.
5. Clark R. Meaningful Games: Exploring Language with Game Theory. - The MIT Press, 2012. - 376 p.
6. Lee K. D., Hubbard S. Data Structures and Algorithms with Python. – Springer, 2015. – 363 p.
7. Диксит А. Стратегические игры. Доступный учебник по теории игр / А. Диксит, С. Скит, Д. Рейли-младший. – Москва: Манн, Иванов и Фербер (МИФ), 2017. – 880 с.
8. Mazhdrakov M. The Monte Carlo Method: Engineering Applications/ Mazhdrakov M. , Benov D. , Valkanov N. – ACMO Academic Press, 2018. – 250 p.

ДОДАТОК А

Метод фіктивного розігрування

```
import numpy as np
    def get_cost_matrix(filename):
        matrrix = list()
        with open(filename, "tr") as matrix_file:
        for line in matrix_file:
        numbers = line.split(" ")
        current_row = list()
            for number in numbers:
            current_row.append(int(number))
            matrrix.append(current_row)
        return np.array(matrrix)

    def get_start_strat_index(matrix):
        rows_mins = matrix.min(axis=1)
        result_index = np.argmax(rows_mins)
        _, el_count =np.unique(rows_mins, return_counts=True)
        if el_count[-1] == 1:
        return result_index
        max_val = np.amax(rows_mins)
        indexes = np.where(rows_mins == max_val)
        rows_to_choose = matrix[indexes]
        index_in_indexes_array = np.argmax(rows_to_choose.max(axis=1))
        return indexes[0][index_in_indexes_array]

    def get_curr_cumm_vector(matrix, old_vector, next_index, is_row=True):
        next_vector = None
        if is_row:
        next_vector = matrix[next_index]
```

```
        else:
        next_vector = matrix[:, next_index]
        return next_vector + old_vector
        scale = 0.01
        cost_matrix = get_cost_matrix("matrix")
        print(cost_matrix)
        start_index = get_start_strat_index(cost_matrix)
        first_strategies = list()
        second_strategies = list()
        cumm_first = np.zeros(cost_matrix.shape[1], dtype=cost_matrix.dtype)
        cumm_second = np.zeros(cost_matrix.shape[0],
dtype=cost_matrix.dtype)
        next_first_srtategy = start_index
        next_second_strategy = None
        W = None
        U = None
        counter = 1
        while True:
        first_strategies.append(next_first_srtategy)
        cumm_first = get_curr_cumm_vector(cost_matrix, cumm_first,
        next_first_srtategy)
        next_second_strategy = np.argmin(cumm_first)
        second_strategies.append(next_second_strategy)
        cumm_second = get_curr_cumm_vector(cost_matrix, cumm_second,
        next_second_strategy,False)
        next_first_srtategy = np.argmax(cumm_second)
        U = cumm_first.min()/counter
        W = cumm_second.max()/counter
        if W - U < scale:
        break
```

```
counter += 1
v = (W+U)/2
_, cnts = np.unique(first_strategies, return_counts=True)
for i in range(len(cnts)):
print("p_{} = {}".format(i+1, cnts[i] / len(first_strategies)))
print("\n\n")
_, cnts = np.unique(second_strategies, return_counts=True)
for i in range(len(cnts)):
print("q")
```

ДОДАТОК Б

Метод Монте-Карло

```
import numpy as np
import sys
    def get_cost_matrix(filename):
        matrrix = list()
        with open(filename, "tr") as matrix_file:
        for line in matrix_file:
        numbers = line.split(" ")
        current_row = list()
        for number in numbers:
        current_row.append(int(number))
        matrrix.append(current_row)
        return np.array(matrrix)

    def are_ineq_satisfied(matrix, x_val, inequality):
        for column_index in range(matrix.shape[1]):
        column = matrix[:, column_index]
        x_dot_a = np.dot(column, x_val)
        if not inequality(x_dot_a):
        return False
        return True

    def get_max_min(game_matrix):
        rows_mins = np.min(game_matrix, axis=1)
        return np.max(rows_mins)
        matrix = get_cost_matrix("matrix")
        max_min = get_max_min(matrix)
        N_iterations = 30000000
        values = np.zeros(shape=matrix.shape[0])
```

```
        for i in range(N_iterations):
        new_random_x = np.random.uniform(0, 1/max_min,
        size=(matrix.shape[0],))
        if not are_ineq_satisfied(matrix, new_random_x, lambda x_dot_a:
x_dot_a
        >=1 ):
        continue
        L = np.sum(new_random_x)
        if L_min > L:
        L_min = L
        values = new_random_x
        print("L:")
        print(L_min)
        print("x_vals:")
        print(values)
        print("V:")
        v = 1 / L_min
        print(v)
        print("p_vals:")
        print(values * v)
```

ДОДАТОК В

Метод направленого перебору

```
import numpy as np


        def get_cost_matrix(filename):
                matrrix = list()
                with open(filename, "tr") as matrix_file:
                for line in matrix_file:
                numbers = line.split(" ")
                current_row = list()
                for number in numbers:
                current_row.append(int(number))
                matrrix.append(current_row)
                return np.array(matrrix)


def calc_p_vals(matrix):
        beta = get_min_max(matrix)
        alpha = get_max_min(matrix)
        main_itr_step = 0.1
        v = beta
        while alpha <= v <= beta:
        n_iterations = 2000000
                for iter_index in range(n_iterations):
                p_random_vals = build_random_p_values(matrix.shape[0])
                if not are_ineq_satisfied(matrix, p_random_vals, lambda p_dot_a:
        p_dot_a >= v):
        continue
        return p_random_vals, v
                v -= main_itr_step
                def get_max_min(game_matrix):
```

```
        rows_mins = np.min(game_matrix, axis=1)
        return np.max(rows_mins)

def get_min_max(game_matrix):
    columns_maxs = np.max(game_matrix, axis=0)

def build_random_p_values(size):
    result = np.zeros(shape=size)
    high_border = 1.0
    for i in range(size-1):
    new_random_p = np.random.uniform(0, high_border, 1)
    result[i] = new_random_p
    high_border -= new_random_p
    result[-1] = high_border
    return result

def are_ineq_satisfied(matrix, x_val, inequality):
    for column_index in range(matrix.shape[1]):
    column = matrix[:, column_index]
    x_dot_a = np.dot(column, x_val)
    if not inequality(x_dot_a):
    return False
    return True
    matrix = get_cost_matrix("matrix")
    p_vals, v_val = calc_p_vals(matrix)
    finish = dt.now()
    print("V : ")
    print(v_val)
    print("P values : ")
    print(p_vals)
```