

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ПРИКЛАДНОГО
СИСТЕМНОГО АНАЛІЗУ**

Кафедра математичних методів системного аналізу

До захисту допущено:

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2024 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системний аналіз і управління»

спеціальності 124 «Системний аналіз»

**на тему: «Інформаційна технологія, моделі і методи біматричної гри з
урахуванням впливу «природи» як третього гравця»**

Виконала:

студентка IV курсу, групи КА-05

Ровенко Ольга Михайлівна

Керівник:

доцент, к.ф.-м.н.

Барановська Леся Валеріївна

Консультант з економічного розділу:

Доцент кафедри економічної кібернетики

к.е.н. Рощина Надія Василівна

Консультант з нормоконтролю:

К.ф.-м.н.

Статкевич Віталій Михайлович

Рецензент:

доцент кафедри СП КПІ ім.І.Сікорського, к.т.н., доцент

Харченко Костянтин Васильович

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів без
відповідних посилань.

Студентка _____

Київ – 2024 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Оксана ТИМОЩУК

«__» _____ 2024 р.

ЗАВДАННЯ

на дипломну роботу студентці

Ровенко Ользі Михайлівні

1. Тема роботи «Інформаційна технологія, моделі і методи біматричної гри з урахуванням впливу «природи» як третього гравця», керівник роботи Барановська Леся Валеріївна, доцент кафедри ММСА, затверджені наказом по університету від «31» травня 2024 р. № 2240-с
2. Термін подання студентом роботи 11.06.2024.
3. Вихідні дані до роботи: біматрична гра, теорія ігор, третій гравець, генетичні алгоритми.
4. Зміст роботи: теоретичні основи, математичне моделювання, практичне застосування.
5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) – презентація, основні результати, матриця вигравів гравця А та гравця В,

результати змін функції пристосованості в залежності від кількості поколінь для біматричної гри не враховуючи вплив «природи».

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Доцент, к. е. н. Рощина Надія Василівна	5.02.2024	29.05.2024

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Формулювання тематики (напрямку) дослідження.	05.02.2024	Виконано
2	Аналіз актуальності задач стосовно тематики дослідження	04.05.2024	Виконано
3	Аналіз відомих результатів стосовно тематики дослідження	10.05.2024	Виконано
4	Формулювання задач дослідження	12.05.2024	Виконано
5	Уточнення теми дипломної роботи	12.05.2024	Виконано
6	Збір статичних даних, попередній аналіз даних	14.05.2024	Виконано
7	Розробка програмного продукту для виконання обчислювальних експериментів	16.05.2024	Виконано
8	Виконання обчислювальних експериментів, аналіз та оформлення результатів	20.05.2024	Виконано
9	Оформлення пояснювальної записки у цілому	29.05.2024	Виконано
10	Підготовка презентації для захисту	03.06.2024	Виконано
11	Попередній захист дипломної роботи	05.06.2024	Виконано

Студентка

Ольга РОВЕНКО

Керівник

Леся БАРАНОВСЬКА

РЕФЕРАТ

Дипломна робота: 87 с., 12 табл., 12 рис., 5 додатків, 26 джерел.

ГЕНЕТИЧНІ АЛГОРИТМИ, ПРИРОДА, ДИНАМІЧНІ СЕРЕДОВИЩА, БІМАТРИЧНІ ІГРИ, СИСТЕМИ ПРИЙНЯТТЯ РІШЕНЬ.

Об'єкт дослідження – біматричні ігри та їхні розв'язки з урахуванням впливу природи як третього гравця.

Предмет дослідження – методи та моделі розв'язання біматричних ігор з урахуванням третього гравця (природи) з використанням генетичного алгоритму.

Мета роботи – дослідити вплив природи як третього гравця на результат біматричної гри двох гравців, провести адаптацію генетичного алгоритму для розв'язання даного завдання, створити комп'ютерну реалізацію та провести тестові випробування.

Методи дослідження – теоретичний аналіз біматричних ігор та генетичних алгоритмів, комп'ютерне моделювання, тестування алгоритмів на прикладних задачах.

Актуальність – традиційно для розв'язання біматричних ігор розглядаються лише два гравці. Вплив природи на результати гри є новим підходом, який може мати важливе значення для багатьох прикладних задач, де враховується невизначеність або зовнішні фактори.

Результати роботи – проведено адаптацію генетичного алгоритму для розв'язання біматричних ігор з урахуванням третього гравця (природи). Створено математичну модель та алгоритм розв'язання поставленого завдання. Реалізовано комп'ютерну програму та проведено тестування, результати якого показали ефективність запропонованого підходу.

Шляхи подальшого розвитку предмету дослідження – інтеграція з методами штучного інтелекту, дослідження генетичних алгоритмів у динамічних середовищах, та впровадження розроблених методів у реальні системи прийняття рішень.

ABSTRACT

Thesis: 87 p., 12 tabl., 12 fig., 4 appendices, 26 references.

GENETIC ALGORITHMS, NATURE, DYNAMIC ENVIRONMENTS,
BIMATRIX GAMES, DECISION-MAKING SYSTEMS.

The object of research – bimatrix games and their solutions considering the influence of nature as a third player.

The subject of research – methods and models for solving bimatrix games considering a third player (nature) using a genetic algorithm.

The aim of the work – to investigate the influence of nature as a third player on the outcome of a bimatrix game between two players, adapt the genetic algorithm to solve this problem, create a computer implementation, and conduct test trials.

Research methods – theoretical analysis of bimatrix games and genetic algorithms, computer modeling, algorithm testing on applied problems.

Relevance – traditionally, only two players are considered in solving bimatrix games. The influence of nature on game outcomes is a new approach that can be significant for many applied problems where uncertainty or external factors are considered.

Results of the work – the genetic algorithm was adapted to solve bimatrix games considering a third player (nature). A mathematical model and solution algorithm were created for the task. A computer program was implemented, and testing showed the effectiveness of the proposed approach.

Ways of further development of the research subject – integration with artificial intelligence methods, studying genetic algorithms in dynamic environments, and implementing developed methods in real decision-making systems.

ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ.....	10
1.1 Вступ до теорії ігор.....	10
1.2 Використання теорії ігор.....	13
1.3 Біматричні ігри двох гравців.....	15
1.4 Гра з природою.....	18
1.5 Постановка задачі.....	19
1.6 Висновки до розділу 1.....	19
РОЗДІЛ 2 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ.....	21
2.1 Математична модель.....	23
2.2 Характеристика генетичного алгоритму.....	24
2.3 Використання алгоритму.....	28
2.4 Висновки до розділу 2.....	30
РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ.....	31
3.1 Вступ.....	31
3.2 Вибір платформи та мови програмування.....	31
3.3 Розрахунки.....	34
3.4 Висновки до розділу 3.....	40
РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ.....	41
4.1 Постановка задачі проектування.....	41
4.2 Обґрунтування функцій програмного продукту.....	42
4.3 Обґрунтування системи параметрів програмного продукту.....	45

	7
4.4 Аналіз експертного оцінювання параметрів.....	47
4.5 Аналіз рівня якості варіантів реалізації функцій.....	51
4.6 Економічний аналіз варіантів розробки ПП.....	53
4.7 Вибір кращого варіанту ПП техніко-економічного рівня.....	58
4.8 Висновки до розділу 4.....	59
ВИСНОВКИ.....	61
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	63
ДОДАТОК А.....	66
ДОДАТОК Б.....	68
ДОДАТОК В.....	75
ДОДАТОК Г.....	81
ДОДАТОК Д.....	88

ВСТУП

У сучасному світі, що швидко розвивається та глобалізується, прийняття ефективних рішень в умовах невизначеності та конфлікту інтересів стає все більш важливим. Теорія ігор є потужним інструментом для аналізу та моделювання стратегічних взаємодій між раціональними агентами в різних сферах, таких як економіка, політика, соціологія та інші.

Особливої актуальності набувають задачі прийняття рішень в умовах невизначеності та неповноти інформації, коли класичні методи теорії ігор можуть бути недостатньо ефективними. В таких ситуаціях перспективним є застосування методів штучного інтелекту, зокрема генетичних алгоритмів, які дозволяють знаходити наближені оптимальні рішення в складних просторах пошуку.

Біматричні ігри є важливим класом ігор, які моделюють стратегічну взаємодію двох гравців з урахуванням їх індивідуальних переваг та виграшів. Вони знаходять широке застосування в моделюванні економічної конкуренції, міжнародних відносин, військових конфліктів та інших сфер.

Врахування впливу природи на процес прийняття рішень в біматричних іграх є актуальною науковою проблемою. Реальні ігрові ситуації часто відбуваються в динамічному та невизначеному середовищі, яке може суттєво впливати на вибір стратегій гравців та їх виграші.

Застосування генетичних алгоритмів для вирішення біматричних ігор з урахуванням впливу середовища дозволяє знаходити адаптивні та стійкі рішення в умовах невизначеності. Генетичні алгоритми моделюють процеси природної еволюції, такі як спадковість, мутації та природний відбір, для пошуку оптимальних рішень в складних просторах.

Розробка ефективних програмних реалізацій генетичних алгоритмів для вирішення біматричних ігор є важливим прикладним завданням. Це вимагає

вибору відповідних структур даних, операторів та параметрів алгоритму, а також проведення ґрунтового тестування та аналізу результатів.

Дослідження ефективності та адаптивності генетичних алгоритмів для вирішення біматричних ігор з урахуванням впливу середовища є актуальним науковим напрямком. Порівняння результатів роботи генетичних алгоритмів з класичними методами теорії ігор дозволяє оцінити переваги та обмеження еволюційних підходів в цьому контексті.

Отже, тема застосування генетичних алгоритмів для вирішення біматричних ігор з урахуванням впливу природи є актуальною та перспективною. Вона поєднує в собі теоретичні аспекти теорії ігор та штучного інтелекту, а також має важливе прикладне значення для моделювання та оптимізації процесів прийняття рішень у майже всіх сферах людської діяльності.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ

1.1 Вступ до теорії ігор

Теорія ігор є відносно дуже новою галуззю математики, яка вивчає стратегічну взаємодію серед гравців-конкурентів у конфліктних ситуаціях, коли результат залежить від дій кожного з них. Вона надає формальні методи для аналізу процесів прийняття рішень в умовах конфлікту інтересів та невизначеності. Станом на сьогодні теорія ігор стала дуже популярною і вже навіть широко застосовується в економіці, політології, психології, біології та інших галузях.

Започаткував теорію ігор Джон фон Нейман (рис. 1.1), американський математик угорського походження. Хоча початковою ціллю вченого була аксезматизація іншої науки, а саме економіки, щоб вона стала такою ж точною наукою як математика чи фізика [1].



Рисунок 1.1 – Джон фон Нейман

Саме Джон фон Нейман винайшов метод мінімаксу, який дозволяє розв'язувати ігри з нульовою сумою, де один гравець виграє, а інший програє. Однак цей метод лише краплина у морі великої кількості задач, що досліджуються в теорії ігор на сьогоднішній день. Наприклад, гра "камінь - ножиці - папір" (рис. 1.2) є однією з таких задач.

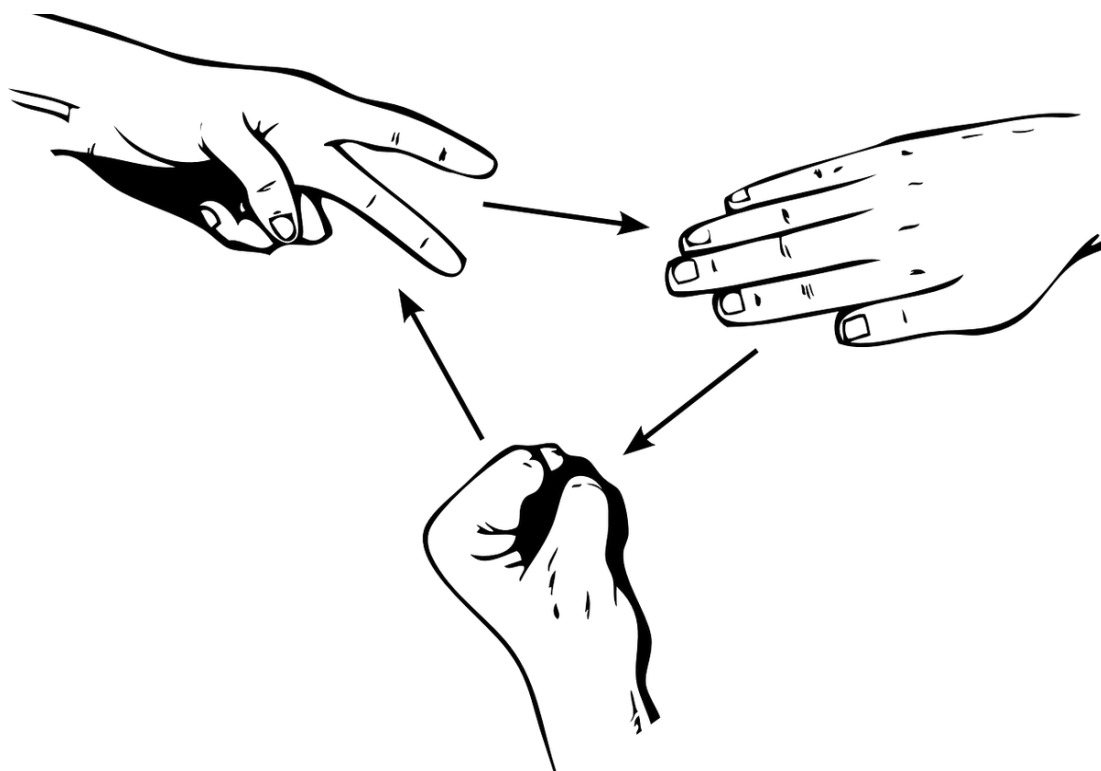


Рисунок 1.2 – Гра "камінь ножиці папір"

У 1944 році Джон фон Нейман разом із Оскаром Моргенштерном, американським економістом німецького походження (див.рис. 1.3) описали основи теорії ігор у своїй книзі "Теорія ігор та економічна поведінка" [2].



Рисунок 1.3 – Оскар Моргенштерн

Основними поняттями теорії ігор є гравці, стратегії, виграші та рівновага. Гравцями можуть бути індивіди, групи, компанії чи будь-які інші суб'єкти, які приймають рішення. Стратегії - це можливі дії, які гравці можуть обрати в процесі гри. Виграші визначають результати, які гравці отримують залежно від обраних стратегій всіма учасниками гри.

Простими словами, теорія ігор – це наука про стратегії, які використовують люди або організації для досягнення своїх цілей, враховуючи можливі дії своїх опонентів чи партнерів.

Ігри можуть бути класифіковані за різними критеріями, такими як кількість гравців, сума виграшів, наявність кооперації, повнота інформації тощо. Наприклад, ігри з нульовою сумою - це ігри, в яких виграш одного гравця дорівнює програшу другого. З іншого боку, ігри з ненульовою сумою передбачають можливість взаємовигідної співпраці між гравцями [27].

Теорія ігор пропонує різні методи вирішення ігрових ситуацій. Одним з найпоширеніших підходів є пошук рівноваги шляхом аналізу найкращих відповідей гравців на стратегії інших учасників. Інші методи включають домінантні стратегії, змішані стратегії, ітеративне виключення домінованих стратегій тощо [3].

Одним з викликів, з якими стикається теорія ігор, є врахування ірраціональності та емоцій гравців. Класична теорія ігор базується на припущенні про повну раціональність агентів, однак в реальності люди часто діють під впливом емоцій, упереджень та обмеженої інформації. Розробка моделей, які враховують ці фактори, є активною областю досліджень [4].

Та звісно, теорія ігор не залишилася поза увагою критиків, і ось основні претензії до неї. По-перше, критики вказують на те, що вона ґрунтується на припущенні, що усі гравці в ній раціональні. Однак у реальності не всі приймають раціональні рішення в кожній ситуації, оскільки на них впливають культурні звички, упередження та особисті установки. Тому можливо індивід зробив би інший вибір, якщо б знаходився в іншому контексті. Друга важлива критика полягає в тому, що теорія ігор передбачає, що кожна людина, після ознайомлення з правилами гри, автоматично застосує оптимальну стратегію. Проте це не завжди відбувається, оскільки перші кроки в навчанні будь-чого включають ознайомлення з правилами та експериментування з можливостями, а не завжди безпосереднє використання оптимальних стратегій.

1.2 Використання теорії ігор

Теорія ігор є потужним інструментом для моделювання та аналізу стратегічної взаємодії між раціональними агентами в різних сферах, таких як економіка, політика, психологія та комп'ютерні науки. Вона дозволяє аналізувати ситуації, в яких результат залежить від дій кількох гравців, кожен з яких має свої цілі та інтереси.

У економіці теорія ігор застосовується для моделювання ринкової конкуренції, аукціонів, переговорів та інших економічних процесів. Вона дозволяє аналізувати стратегічну поведінку фірм та споживачів.

У політології теорія ігор використовується для дослідження процесів прийняття рішень, формування коаліцій, міжнародних відносин та інших політичних явищ. Вона допомагає зрозуміти стратегічну взаємодію між політичними акторами [5].

У психології теорія ігор застосовується для вивчення процесів прийняття рішень, соціальних дилем, кооперації та конкуренції між індивідами. Вона дозволяє моделювати та аналізувати поведінку людей в різних ситуаціях.

У комп'ютерних науках теорія ігор використовується для розробки алгоритмів штучного інтелекту, моделювання поведінки агентів у мультиагентних системах, оптимізації мережевих протоколів та інших задач.

Теорія ігор також знаходить застосування в сфері кібербезпеки та захисту інформації. Ігрові моделі використовуються для аналізу стратегій нападу та захисту в контексті конфлікту інтересів та асиметрії інформації між зловмисниками та захисниками.

Важливим напрямком досліджень в теорії ігор є вивчення еволюційних ігор, які моделюють процеси адаптації та навчання в популяціях агентів. Еволюційні ігри дозволяють аналізувати динаміку стратегій в довгостроковій перспективі та вивчати еволюцію співпраці та конкуренції в біологічних та соціальних системах [6].

Теорія ігор також тісно пов'язана з теорією прийняття рішень та теорією соціального вибору. Вона надає інструменти для аналізу процесів колективного прийняття рішень, таких як голосування, розподіл ресурсів, формування коаліцій тощо.

Одним з перспективних напрямків розвитку теорії ігор є дослідження ігор з неповною інформацією, коли гравці не мають повного знання про характеристики інших учасників або про правила гри. Такі ситуації часто зустрічаються в реальному житті, і їх аналіз вимагає розробки нових методів та моделей.

Теорія ігор також знаходить застосування в сфері штучного інтелекту та робототехніки. Ігрові моделі використовуються для розробки алгоритмів прийняття рішень та координації поведінки автономних агентів в динамічних та невизначених середовищах [7].

Важливим аспектом застосування теорії ігор є експериментальна перевірка теоретичних результатів. Експериментальна економіка та поведінкова теорія ігор вивчають реальну поведінку людей в ігрових ситуаціях та порівнюють її з передбаченнями теоретичних моделей.

Розвиток теорії ігор супроводжується створенням спеціалізованого програмного забезпечення для моделювання та аналізу ігрових ситуацій. Такі програми дозволяють будувати та досліджувати складні ігрові моделі, знаходити рівноваги та оптимальні стратегії, візуалізувати результати та проводити аналіз чутливості.

1.3 Біматричні ігри двох гравців

Біматричні ігри двох гравців є підкласом ігор з ненульовою сумою, де кожен гравець має обмежений набір чистих стратегій. У цих іграх основна форма опису моделей представляється через платіжну матрицю, яка показує виграші кожного гравця у залежності від обраних ними стратегій. (приклад платіжної матриці наведений на рисунку 1.4)

Основними елементами біматричної гри є гравці, стратегії, виграші та платіжні матриці. Гравцями можуть бути індивіди, групи або організації. Стратегії визначають можливі дії гравців, а виграші - результати, які отримують гравці в залежності від обраних стратегій.

$$\mathbf{A} = \begin{bmatrix} c & d & e \\ f & g & h \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} p & q & r \\ s & t & u \end{bmatrix}$$

	<i>X</i>	<i>Y</i>	<i>Z</i>
<i>V</i>	<i>c, p</i>	<i>d, q</i>	<i>e, r</i>
<i>W</i>	<i>f, s</i>	<i>g, t</i>	<i>h, u</i>

A payoff matrix converted from **A** and **B** where player 1 has two possible actions *V* and *W* and player 2 has actions *X*, *Y* and *Z*

Рисунок 1.4 – Платіжна матриця біматричної гри перетворена з матриць виграшів **A** і **B**, де гравець 1 має дві можливі дії *V* і *W*, а гравець 2 має три можливі дії *X*, *Y* і *Z*

Ключовою концепцією біматричних ігор є рівновага Неша, яка представляє собою стан гри, в якому жоден з гравців не має стимулу змінювати свою стратегію за умови, що інші учасники дотримуються своїх стратегій. Рівновага Неша існує в будь-якій скінченній грі з ненульовою сумою.

Складові елементи біматричної моделі : $\langle N, SN, UN \rangle$, що представляють собою:

N – гравці ну як або учасники гри (у подальшому розглядатимемо, що $N=2$).

SN – набір можливих дій або виборів, доступних кожному гравцю під час гри.

UN – функції виграшів гравців: $UN : X \times Y \rightarrow R$, де $X, Y \in SN$

Розв'язання гри ґрунтується на пошуку рівноваги, де кожен гравець не має вигоди від виходу з обраного набору стратегій, коли всі інші гравці залишаються в цій же стратегії.

Однією з найбільш привабливих концепцій в теорії ігор є поняття рівноваги Неша, за яке Джон Форбс Неш був удостоєний Нобелівської премії з економіки. При закінченні навчання в університеті його викладач написав лише одне речення у характеристиці: «Ця людина – математичний геній!» [8].

У грі $G_{bi} = \{S_i, U_i, i \in N\}$, стратегії $(s_1^*, s_2^*, \dots, s_N^*)$ створюють рівновагу Неша, якщо для кожного гравця i стратегія s_i^* є його найкращою відповіддю на набір стратегій s_{-i}^* інших гравців. Тобто, s_i^* вирішує задачу.

$$\max_{s_i \in S_i} U_i(s_1^*, \dots, s_{i-1}^*, s_i, s_{i+1}^*, \dots, s_n^*) \quad (1.1)$$

Теорема Неша. [24] *У будь-якій грі з кінцевою кількістю гравців, де кожен гравець має скінченну кількість стратегій, існує принаймні одна рівновага Неша.*

Це означає, що існує набір стратегій, при якому жоден гравець не може покращити свій виграш, змінюючи свою стратегію в односторонньому порядку, за умови, що стратегії інших гравців залишаються незмінними (див. рис. 1.5).

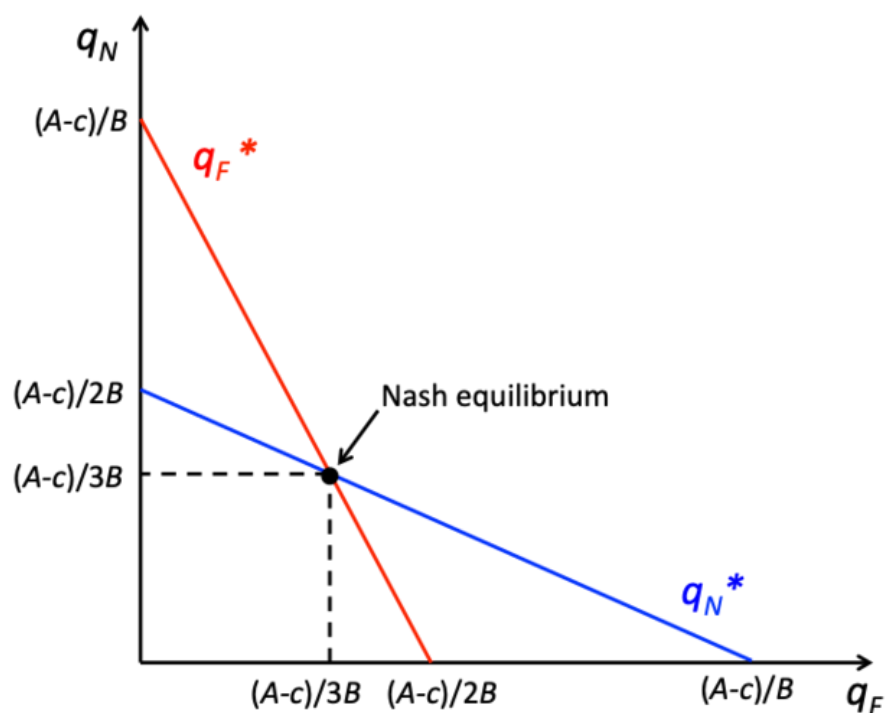


Рисунок 1.5 – Ілюстрація концепції рівноваги Неша для двох гравців.

1.4 Гра з природою

Гра із природою включає одного гравця, який максимізує свій вигравш, враховуючи невизначеність зовнішніх факторів. Природа в цих моделях представляє випадкові події або умови, такі як попит на ринку або погодні умови. "Природа" в цьому контексті не є активним учасником гри, але впливає на результат через випадкові фактори.

При аналізі ігор з природою рішення можуть прийматися за двома основними критеріями [9]:

- 1) умови ризику: рішення приймаються, коли відомі ймовірності кожного можливого стану природи;
- 2) умови невизначеності: рішення приймаються без інформації про ймовірності виникнення різних станів природи.

Аналіз економічних сценаріїв, оцінка ефективності рішень та вибір оптимальних альтернатив в умовах невизначеності часто здійснюються за

допомогою ігор з природою. Ці методи широко застосовуються для моделювання різних економічних ситуацій і визначення найкращих рішень у нестабільних умовах. Наприклад, компанія, що планує випуск нового продукту на ринок, може скористатися моделлю гри з природою для оцінки попиту і прийняття стратегічних рішень.

1.5 Постановка задачі

Перейдемо до постановки задачі.

Потрібно дослідити вплив «природи» у ролі третього гравця у результаті біматричної гри двох гравців.

Для цього необхідно провести такі дослідження:

1. Провести аналіз теоретичних основ теорії ігор, біматричних ігор та генетичного алгоритму.
2. Розробити математичну модель біматричної гри з урахуванням впливу природи, як третього гравця.
3. Реалізувати генетичний алгоритм для розрахунку оптимальних стратегій гравців без урахування та з урахуванням природи.
4. Реалізувати комп'ютерну програму для моделювання і аналізу біматричних ігор із впливом природи.
5. Провести аналіз отриманих результатів і зробити висновки.

1.6 Висновки до розділу 1

У першому розділі дипломної роботи було проведено вступ до теорії ігор, де було розглянуто основні поняття та принципи цієї науки, а також його історичний аспект та важливість у сучасному світі. Наступним кроком було проаналізовано практичні застосування теорії ігор у різних сферах діяльності, включаючи економіку, політику, бізнес тощо. Після цього було детально розглянуто

біматричні ігри двох гравців як основний об'єкт дослідження, з визначенням основних понять, таких як стратегія, профіль стратегій, матриця виграшів тощо. Також було проведено аналіз різних типів біматричних ігор та їх характеристик. Заключним етапом дослідження було розглянуто ігри з природою, де природа виступає як третій гравець, проаналізовано їх вплив на стратегії гравців та розвиток гри, а також розглянуті приклади таких ігор та їх можливі застосування в різних областях.

РОЗДІЛ 2 МАТЕМАТИЧНЕ МОДЕЛЮВАННЯ

Давайте розглянемо моделювання рішень гравців, які прагнуть знайти оптимальні стратегії поведінки в конфліктних ситуаціях. Стратегія являє собою набір правил, що визначають вибір дій кожним гравцем у конкретних ситуаціях. Найкраща стратегія гравця максимізує його виграш у грі. Якщо гра повторюється багаторазово та включає як індивідуальні, так і випадкові ходи, оптимальна стратегія гарантує максимальний середній виграш.

У цій грі беруть участь два учасники, яких назовемо "гравець А" і "гравець В". Учасники — це конкуруючі компанії, які змагаються за ринкові частки, або будь-які інші суб'єкти в економічній чи управлінській діяльності.

Цими учасниками можуть бути конкуруючі компанії, що змагаються за ринки збуту, або будь-які інші агенти в економічній чи управлінській діяльності.

У кожному раунді гри гравець А може вибрати одну з m доступних стратегій, тоді як гравець В має можливість обрати одну з n альтернативних стратегій. Це створює ситуацію, у якій кожен гравець має певні виграші. Виграш гравця А в ситуації, коли він використовує i -ту стратегію, а гравець В - j -ту стратегію, позначається як (a_{ij}) , тоді як виграш гравця В за таких самих умов позначається як (b_{ij}) . Набір таких ситуацій та виграшів складає платіжну матрицю гри, відому як матриця біматричної гри (G_{bi}) .

$$G_{bi} = \begin{pmatrix} a_{11}, b_{11} & a_{12}, b_{12} & \dots & a_{1n}, b_{1n} \\ a_{21}, b_{21} & a_{22}, b_{22} & \dots & a_{2n}, b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1}, b_{m1} & a_{m2}, b_{m2} & \dots & a_{mn}, b_{mn} \end{pmatrix} \quad (2.1)$$

У цій матриці кожен подвійний елемент представляє виграш кожного гравця в кожній з можливих ситуацій гри. Математичну модель гри можна описати так [10]:

$$U_A(A, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n a_{ij} p_j q_j, U_B(B, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n b_{ij} p_j q_j, \\ \sum_{i=1}^m p_i = 1, \sum_{j=1}^n q_j = 1 \quad (2.2)$$

2.1 Математична модель

У рамках теорії ігор математичне моделювання відіграє ключову роль в аналізі стратегічної взаємодії між гравцями. Модель біматричної гри, яку розглядаємо, включає трьох учасників: гравця А, гравця В та природу як третього гравця. Природа визначає зовнішні фактори, які впливають на результати гри. Загальна математична модель включає набір співвідношень (2.1)-(2.6). Для вирішення біматричної гри без впливу природи необхідно вирішити рівняння (2.1)-(2.3). Щоб моделювати гру, де кожен гравець взаємодіє лише з природою, слід вирішити рівняння (2.4)-(2.6).

Опишемо основні елементи математичної моделі.

1. Гравці та їхні стратегії: гравці А і В мають набір можливих стратегій, які вони можуть обирати для досягнення своїх цілей. Стратегія кожного гравця визначає його поведінку в грі та реакцію на дії інших учасників.

2. Виграші гравців: виграш гравця залежить від обраної стратегії як його власної, так і стратегії суперника. Виграші представляються у вигляді матриць, де кожен елемент матриці відображає виграш гравця за конкретної комбінації стратегій.

3. Роль природи: у нашому випадку природа діє як третій гравець, що вносить невизначеність у гру. Її вплив може бути представлений у вигляді випадкових подій, таких як зміни попиту на ринку або погодні умови, які впливають на результати рішень гравців.

4. Інформаційні стани гри: інформаційні стани визначають, які саме умови враховуються гравцями при прийнятті рішень. Це може включати інформацію про стратегії суперників та стан природи.

Одним з важливих аспектів моделювання є врахування змішаних стратегій, де гравці можуть обирати свої дії з певною ймовірністю. Це дозволяє уникнути передбачуваності та збільшити шанси на виграш. Однією з ключових характеристик рівноваги Неша у змішаних стратегіях є те, що кожна дія, яка

підтримує рівноважну змішану стратегію, є оптимальною для гравця. Це означає, що будь-який вибір у рамках цієї стратегії забезпечує гравцю однаковий виграш.

2.2 Характеристика генетичного алгоритму

Генетичний алгоритм (ГА) є еволюційною методикою обчислень, що імітує процес природної еволюції для пошуку оптимальних та адаптивних стратегій гравців в умовах невизначеності та мінливості середовища. Ця ідея була запропонована Джоном Голландом у 1970-х роках. Він зацікавився властивостями процесів природної еволюції, зокрема тим, що еволюціонують хромосоми, а не самі живі істоти. Голланд створив і реалізував у вигляді комп'ютерної програми алгоритм, який вирішував складні задачі шляхом еволюції, імітуючи природні процеси відбору та спадковості [12].

ГА можуть бути застосовані для розв'язання задач, які раніше вирішувалися тільки нейронними мережами. Наприклад, вони використовуються для розв'язання задачі комівояжера, оптимізації параметрів у системах штучного інтелекту, підтримки нейронних мереж або в рамках гібридних систем.

ГА працюють з популяцією хромосом, кожна з яких представляє можливе рішення задачі у вигляді послідовності символів. Початкова популяція зазвичай створюється випадковим чином. Кожна хромосома оцінюється за допомогою функції пристосованості, яка визначає, наскільки добре конкретне рішення вирішує поставлену задачу. Хромосоми з найвищою оцінкою відбираються для створення нових хромосом через процеси схрещування і мутації.

Схрещування полягає в комбінуванні генетичного матеріалу пари батьків для створення нових хромосом-нащадків. Це дозволяє створювати нові комбінації генів, що може покращити пристосованість наступних поколінь. Мутація, в свою чергу, вносить випадкові зміни в гени окремих хромосом, підтримуючи генетичну різноманітність у популяції.

Генетичні алгоритми складаються з кількох основних етапів [26], кожен з яких відіграє важливу роль у процесі пошуку оптимального рішення (таблиця 2.1).

Таблиця 2.1– Основні етапи генетичного алгоритму

Етап	Опис
Ініціалізація	Генерація початкової популяції випадкових рішень
Оцінка пристосованості	Обчислення функції придатності для кожного рішення
Відбір	Вибір найбільш пристосованих рішень для створення нового покоління
Схрещування	Обмін генетичною інформацією між відібраними рішеннями
Мутація	Випадкова зміна деяких генів в хромосомах нащадків
Формування нової популяції	Заміна старої популяції новими рішеннями
Перевірка умов зупинки	Досягнення заданої кількості поколінь і рівня пристосованості. Відсутність значних змін у значеннях пристосованості протягом певної кількості поколінь.

Розглянемо детально ці етапи [13]:

1. Ініціалізація популяції. На цьому етапі створюється початкова група хромосом, які представляють можливі розв'язки задачі. Зазвичай популяція формується випадковим чином, щоб забезпечити широку різноманітність початкових рішень. Кожна хромосома кодує потенційне рішення задачі у вигляді послідовності генів.

2. Оцінка пристосованості. Кожна хромосома в популяції оцінюється за допомогою функції пристосованості (фітнес-функції). Ця функція визначає, наскільки добре конкретне рішення задовольняє вимоги задачі. Значення фітнес-функції визначає ймовірність того, що хромосома буде обрана для подальшої репродукції. Чим вища пристосованість, тим краще рішення.

3. Відбір (селекція). Відбір передбачає вибір хромосом для створення нової популяції на основі їх пристосованості.

4. Схрещування (кросовер). Схрещування є процесом комбінування генетичного матеріалу двох батьківських хромосом для створення дочірніх хромосом. Цей етап сприяє генетичному різноманіттю і дозволяє створювати нові комбінації генів. Існує кілька методів схрещування:

5. Мутація. Мутація полягає у випадковій зміні одного чи декількох генів у хромосомі. Цей етап допомагає підтримувати генетичне різноманіття в популяції і запобігає передчасному збіженню до локальних оптимумів. Мутація може відбуватися з певною малою ймовірністю.

6. Формування нової популяції. Після виконання відбору, схрещування та мутації створюється нова популяція хромосом, яка замінює стару популяцію. Нове покоління хромосом стає основою для наступного циклу алгоритму.

7. Перевірка умов зупинки. Генетичний алгоритм продовжує свою роботу, аж поки умова зупинки не буде виконаною. Умови зупинки можуть включати:

Вкажемо декілька умов зупинки для генетичного алгоритму.

1. Досягнення заданої кількості поколінь.
2. Досягнення заданого рівня пристосованості.
3. Відсутність значних змін у значеннях пристосованості протягом певної кількості поколінь.
4. Обмеження за часом виконання.
5. Досягнення заданої кількості оцінок функції пристосованості.

Для кращого розуміння розглянемо блок-схему генетичного алгоритму, зображену на рисунку 2.1 [14].

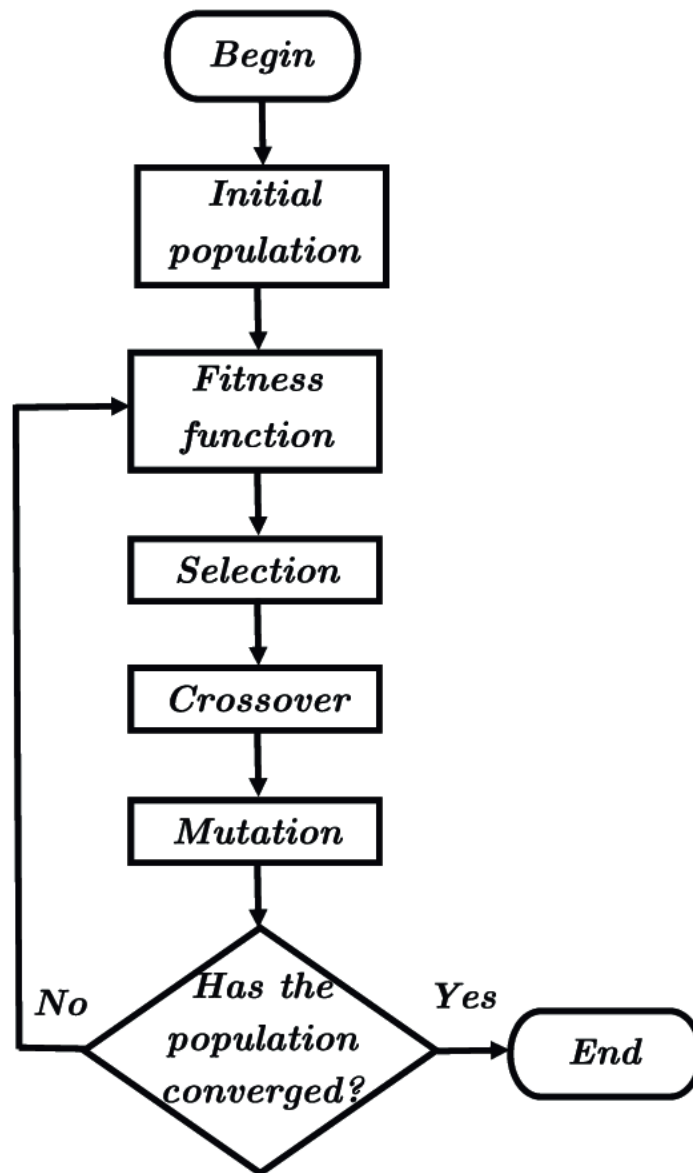


Рисунок 2.1 – Блок-схема генетичного алгоритму

Генетичний алгоритм (ГА) нині є одним із найпопулярніших методів стохастичної оптимізації, широко застосовуваним для розв'язання складних задач у різних сферах.

Цілком логічним здається, що ГА має ряд переваг порівняно з іншими методами оптимізації. Він не вимагає знання градієнта цільової функції, може працювати з великими та складними просторами пошуку, та здатний знаходити глобальний оптимум, уникаючи застрягання в локальних оптимумах (див. табл. 2.2).

Однак ГА також має деякі недоліки. Він може бути обчислювально затратним для задач з великою розмірністю, і його ефективність залежить від вибору параметрів, таких як розмір популяції, ймовірності схрещування та мутації тощо [15].

Таблиця 2.2 - Переваги та недоліки генетичного алгоритму

Переваги	Недоліки
Не вимагає знання градієнта цільової функції	Може бути обчислювально затратним
Працює з великими та складними просторами пошуку	Ефективність залежить від вибору параметрів
Здатний знаходити глобальний оптимум	Не гарантує знаходження точного оптимуму
Легко адаптується до різних задач оптимізації	Результати можуть бути детермінованими

2.3 Використання алгоритму

Створимо адаптацію генетичного алгоритму для виконання поставленого завдання.

1. Ініціалізація початкової популяції:

Генеруємо початкову популяцію випадкових рішень, де кожне рішення (хромосома) представляє набір змішаних стратегій для гравців А і В та можливих станів природи.

$$S_A(p_1, p_2, p_3, \dots, p_n); \quad S_B(q_1, q_2, q_3, \dots, q_n).$$

Кожна хромосома містить імовірності вибору стратегій для кожного гравця та станів природи.

Будемо дотримуватися властивості при генерації

$$\sum_{i=1}^m p_i = 1, \quad \sum_{j=1}^n q_j = 1$$

2. Оцінка пристосованості:

Функція пристосованості визначає виграші гравців А і В на основі їх стратегій та впливу природи. Виграші обчислюються як зважена сума, де ваги відображають відносну важливість цілей кожного гравця.

Використовуємо рівняння для обчислення виграшів гравців:

$$U_A(A, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n a_{ij} p_i q_j, \quad U_B(B, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n b_{ij} p_i q_j,$$

де p_i і q_j — імовірності вибору стратегій гравців А і В відповідно.

3. Селекція:

Відбираємо хромосоми з найвищими значеннями пристосованості для створення нового покоління.

4. Схрещування:

Вибираємо пари хромосом з високою пристосованістю для схрещування. Обираємо точку розділу в хромосомах і комбінуємо генетичний матеріал обох батьків для створення нащадків. Схрещування дозволяє створювати нові рішення з комбінацією стратегій, які можуть бути ефективнішими.

5. Мутація:

Для хромосом, що не брали участі в схрещуванні, спершу визначають ген з найбільшим значенням і переносять його на відповідне місце в нащадка. Решта генів зазнають мутації: випадково обирається один ген, до якого додається 0,05, а від іншого це число віднімається. Оновлені гени потім інтегруються в нащадкову хромосому, забезпечуючи генетичне різноманіття в популяції.

6. Формування нової популяції:

Заміна старої популяції новими хромосомами, отриманими після етапів схрещування та мутації. Нове покоління має бути більш пристосованим до умов гри з урахуванням впливу природи.

7. Умови зупинки:

Критерієм зупинки алгоритму оберемо фіксовану кількість змін поколінь під час його виконання. Якщо цей критерій ще не виконано, ми переходимо до етапу схрещування та мутації, а потім направляємо нове покоління на селекцію.

2.4 Висновки до розділу 2

У другому розділі дипломної роботи було проаналізовано та змодельовано біматричну гру з урахуванням впливу третього гравця, яким є природа. Адаптований генетичний алгоритм був запропонований для визначення оптимальних стратегій гравців у заданих умовах.

Розроблено модель генетичного алгоритму, що враховує вплив навколишнього середовища на результати гри, та показано, як генетичний алгоритм може бути застосований для пошуку оптимальних стратегій гравців у біматричній грі. Створено математичну модель біматричної гри, яка включає функції виграшу для кожного гравця та описує, як змінні стану навколишнього середовища впливають на вибір стратегій гравців.

Також було розглянуто основні етапи генетичного алгоритму, такі як ініціалізація популяції, оцінка пристосованості, селекція, схрещування, мутація, формування нової популяції та умови зупинки алгоритму. Для кожного з цих етапів було визначено відповідні математичні формули та принципи їх застосування у контексті біматричної гри.

Аналіз результатів моделювання підтвердив гіпотезу про значний вплив третього гравця на результати гри. Ці результати демонструють потенціал використання генетичних алгоритмів для моделювання складних стратегічних взаємодій у ситуаціях з невизначеністю та змінними умовами.

РОЗДІЛ 3. ПРАКТИЧНЕ ЗАСТОСУВАННЯ

3.1 Вступ

У цьому розділі спробуємо розглянути різні підходи до вирішення біматричних ігор з урахуванням впливу природи як третього гравця. Ми змоделюємо ігрові ситуації, дослідимо їх особливості та проаналізуємо результати.

Спочатку опишемо процедури розробки та налаштування програмного забезпечення, включаючи вибір відповідних структур даних та алгоритмів. Далі представимо результати тестування на різних прикладах біматричних ігор, що дозволить оцінити ефективність підходу.

Обговоримо отримані результати та зробимо висновки щодо перспектив подальшого використання розроблених методів у практичних завданнях.

3.2 Вибір платформи та мови програмування

Для реалізації практичної частини дипломної роботи було обрано використовувати мову програмування Python. Ця мова програмування має дуже простий і зрозумілий синтаксис, що дозволяє швидко писати код і легко його читати. Це особливо важливо для академічних досліджень, де потрібно фокусуватися на алгоритмах і концепціях, а не на деталях реалізації. Важливо також, що Python є кросплатформним і підтримує об'єктно-орієнтоване програмування. Для реалізації високопродуктивних алгоритмів також можна використовувати мову C++, але знов ж таки, перевагою Python є значно простіший інтерфейс. Завдяки бібліотекам NumPy та Pandas, Python дозволяє ефективно працювати з великими обсягами даних, що важливо для реалізації генетичних алгоритмів. Велика кількість готових бібліотек дозволяють швидко розробляти та тестувати алгоритми. Python підтримує інтеграцію з іншими мовами та

інструментами, що дозволяє використовувати його в різних середовищах та для різних завдань. Враховуючи всі ці переваги, Python є найкращим вибором для реалізації цієї роботи.

Вхідні дані для реалізації включають адаптацію генетичного алгоритму для вирішення біматричної гри з урахуванням "природи", як описано в пункті 2.3.

Математичні моделі наведені в пункті 2.1.

Комп'ютерна реалізація представлена в додатках А, Б, В, Г. Опишемо їх детальніше.

Додаток А - Основний модуль ('main.py')

Основний модуль виконує роль стартового скрипта для всієї програми. У ньому задаються вхідні матриці для біматричної гри між двома гравцями та для гри з природою. За допомогою класів 'GeneticAlgorithmBimatrixGame' та 'GeneticAlgorithmZeroSumGame' відбувається запуск генетичних алгоритмів, які вирішують відповідні ігри. Після завершення алгоритмів обчислюються комбіновані стратегії гравців, що включають результати як біматричної гри, так і гри з природою. Результати обчислень виводяться на екран, що дозволяє побачити найкращі знайдені стратегії для обох гравців та їх взаємодію з природою.

Додаток Б - Клас GeneticAlgorithmBimatrixGame

Клас GeneticAlgorithmBimatrixGame реалізує генетичний алгоритм для вирішення біматричної гри між двома гравцями. Вхідними даними є матриці виграшів для кожного гравця. Алгоритм починається зі створення початкової популяції хромосом, де кожна хромосома представляє можливу стратегію гравців. Протягом багатьох поколінь хромосоми піддаються відбору, схрещуванню та мутаціям для підвищення їхньої пристосованості, яка оцінюється на основі виграшів гравців. Кінцевим результатом є хромосоми з найвищою пристосованістю, які представляють оптимальні стратегії для гравців у

біматричній грі. Алгоритм також візуалізує процес еволюції пристосованості за допомогою графіків.

Додаток В - Клас GeneticAlgorithmZeroSumGame

Клас GeneticAlgorithmZeroSumGame реалізує генетичний алгоритм для вирішення гри з природою, де природа виступає як третій гравець у нульовій сумі. Вхідними даними є матриця виграшів природи. Алгоритм створює початкову популяцію хромосом, кожна з яких представляє можливу стратегію гравців. Протягом кількох поколінь хромосоми проходять через процеси відбору, схрещування та мутації. Пристосованість хромосом оцінюється на основі стратегій природи, і найкращі хромосоми відбираються для наступних поколінь. Результатом є хромосома з найвищою пристосованістю, яка представляє оптимальну стратегію для гри з природою. Алгоритм також надає візуалізацію еволюції функції пристосованості за допомогою графіків.

Додаток Г - Клас GeneticAlgorithmTwoPlayersAndNatureGame

Клас GeneticAlgorithmTwoPlayersAndNatureGame реалізує генетичний алгоритм для вирішення гри з трьома учасниками: двома гравцями та природою. Вхідними даними є матриці виграшів для двох гравців та природи. Алгоритм починається зі створення початкової популяції хромосом, де кожна хромосома представляє можливу стратегію для гравців і природи. Протягом багатьох поколінь хромосоми піддаються відбору, схрещуванню та мутаціям для підвищення їхньої пристосованості. Пристосованість оцінюється на основі комбінованих виграшів гравців та впливу природи. Кінцевим результатом є хромосоми з найвищою пристосованістю, які представляють оптимальні стратегії для всіх учасників гри. Алгоритм також візуалізує процес еволюції пристосованості за допомогою графіків, що дозволяє спостерігати динаміку оптимізації стратегій.

3.3 Розрахунки

Для виконання тестових розрахунків, змодельємо гру, представивши її як економічну ситуацію наступним чином.

Формулювання задачі.

Дві виробничі компанії, А і В, спеціалізуються на виготовленні трьох типів продуктів: Продукт1, Продукт2, і Продукт3 (П1, П2, П3).

Їхні взаємодії у конкурентному середовищі описуються моделлю біматричної гри, що представлена такими матрицями виграшів (табл. 3.1 - 3.2).

Таблиця 3.1 – Матриця виграшів гравця А та гравця В

Стратегії компанії В		П1	П2	П3
Стратегії компанії А	П1	3, 2	2, 2	0, 4
	П2	1, 1	3, 4	4, 0
	П3	2, 0	1, 3	1, 1

- Стратегії компанії А: П1, П2, П3.
- Стратегії компанії В: П1, П2, П3.

Кожен гравець має три можливі стратегії: П1, П2 і П3. Матриця виграшів показує прибуток обох гравців в залежності від вибору їх стратегій. Значення в клітинках матриці представляють виграш гравця А і гравця В відповідно.

Наприклад якщо гравець А обирає стратегію П1, а гравець В обирає стратегію П1, то компанія А отримає 2, а компанія В отримає 1 (перший рядок, перший стовпець: 2, 1). Якщо компанія А обирає стратегію П2, а компанія В обирає стратегію П2, то компанія А отримає 1, а гравець В отримає 2 (другий рядок, другий стовпець: 1, 2).

Таблиця 3.2 – Матриця виграшів гравців у грі з урахуванням впливу природи

Стратегії «природи»		M1	M2	M3
Стратегії компанії А	П1	8	2	1
	П2	4	8	2
	П3	2	4	8

- Стратегії «природи»: M1, M2, M3.
- Стратегії компанії А: П1, П2, П3.

Компанія А взаємодіє з природою, яка також має три можливі стратегії: M1, M2 і M3. Значення в клітинках матриці представляють виграш гравця А в залежності від вибору його стратегії та стратегії природи.

Завдання:

Потрібно дослідити який вплив природа чинить на біматричну гру між двома гравцями. Для цього виконати порівняння наступних сценаріїв:

1. Провести розрахунок біматричної гри між гравцями, не враховуючи вплив природи.

2. Виконати розрахунок моделі гри, де спочатку гравці змагаються один з одним у біматричній грі, отримуючи результат R_{bi} , а потім кожен з них грає з природою, отримуючи результат R_p . Отримані результати ігор усереднюються за формулою:

$$R = 0,6 \cdot R_{bi} + 0,4 \cdot R_p \quad (3.1)$$

3. Третя модель передбачає одночасну гру трьох гравців, де одна гра є біматричною, а інші дві антагоністичними. У цій грі функція пристосованості в генетичному алгоритмі визначається за формулою:

$$F = 0,6 \cdot F_{bi} + 0,4 \cdot F_p \quad (3.2)$$

4. Провести аналіз результатів розрахунків моделей і зробити висновки щодо впливу природи на біматричну гру.

Кроки для розв'язання

1. Визначити стратегії. Стратегії обох компаній K1 та K2 включають вибір одного з трьох продуктів для виробництва.
2. Створити початкову популяцію. Початкова популяція складатиметься з випадково обраних стратегій для обох компаній.
3. Оцінити пристосованість. Функція пристосованості для кожної хромосоми обчислюється на основі матриць виграшів для компаній і впливу Природи.
4. Селекція. Відбирати найкращі стратегії для наступних поколінь на основі їхньої пристосованості.
5. Кросовер. Виконати схрещування стратегій між парами компаній для створення нових стратегій.
6. Мутація. Вносити випадкові зміни в стратегії для збереження генетичної різноманітності.
7. Еволюція. Повторювати кроки 4-6 протягом декількох поколінь для пошуку оптимальних стратегій.
8. Аналіз результатів. Візуалізувати еволюцію функції пристосованості та проаналізувати кінцеві стратегії обох компаній та їх взаємодію з Природою.

Отримані результати під час дослідження:

Відповідно до алгоритму вирішення (див. 2.3) було розроблено програмне забезпечення (див. додатки Б, В, Г), за допомогою якого отримано результати (див. рис. 3.1-3.2, табл. 3.3-3.4).

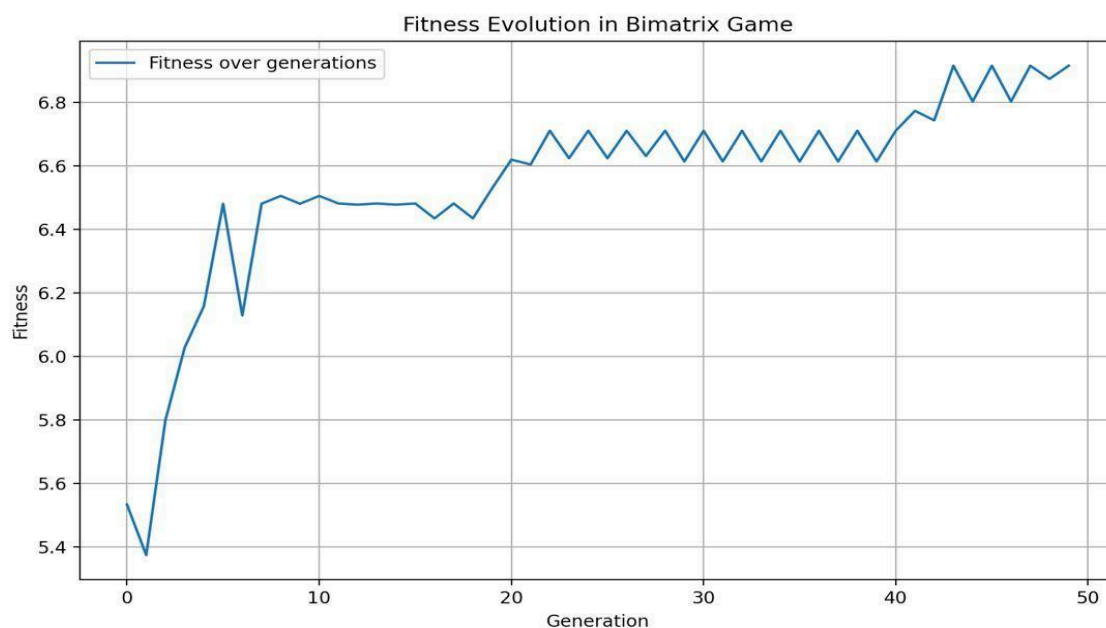


Рисунок 3.1 – Результати зміни функції пристосованості у біматричній грі в залежності від кількості поколінь, без урахування впливу «природи».

Таблиця 3.3 – Результати стратегічного вибору гравців А та В у рамках другої моделі.

Стратегія проти природи	0,47313266	0,38493724	0,22299255
Гравець А	0,006	0,993	0,001
Гравець В	0,083	0,905	0,012
Природа * 0,4	0,18925306	0,1539749	0,08919702
Гравець А * 0,6	0,0036	0,5958	0,0006
Гравець А з урахуванням природи	0,079301224	0,65738996	0,036278808
Природа * 0,04	0,18925306	0,1539749	0,08919702
Гравець В * 0,06	0,00498	0,0543	0,00072
Гравець В з урахуванням природи	0,06308	0,6878	0,007488

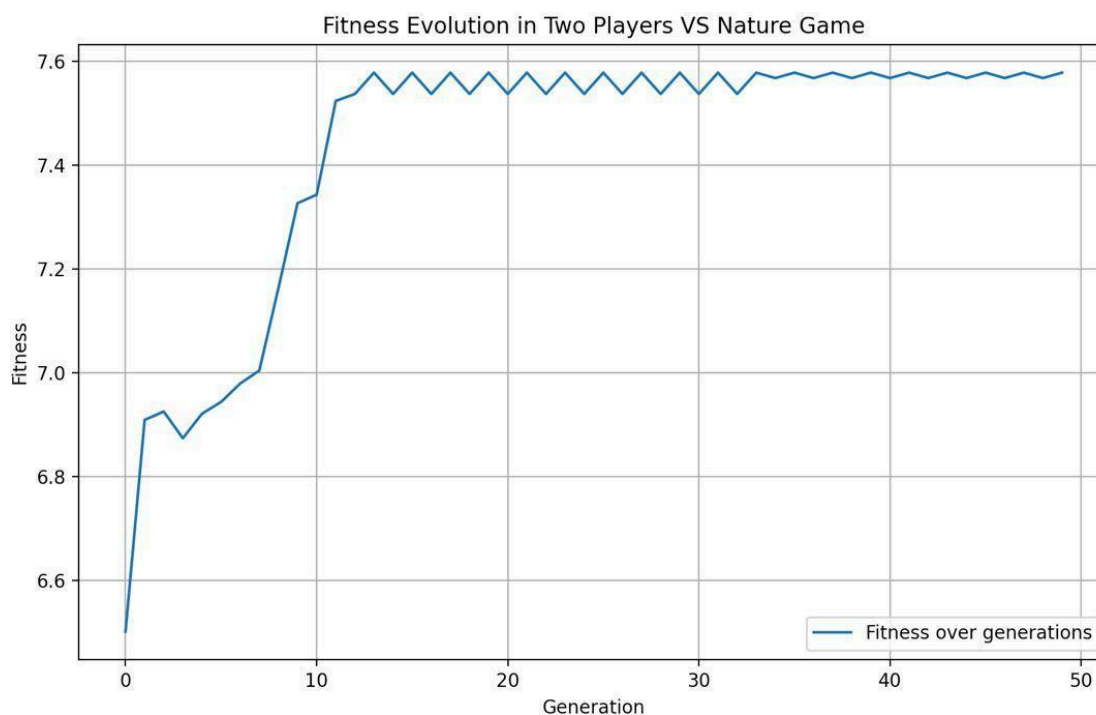


Рисунок 3.2 – Результати зміни функції пристосованості у біматричній грі в залежності від кількості поколінь, з урахуванням впливу «природи».

Таблиця 3.4 – Результати стратегічного вибору гравців А та В у рамках третьої моделі.

Стратегія проти природи	0,47313266	0,38493724	0,22299255
Гравець А	0,006	0,993	0,001
Гравець В	0,083	0,905	0,012

Гра трьох гравців

Природа	0,47313266	0,38493724	0,22299255
Гравець А	0,0762644	0,6285556	0,05518
Гравець В	0,6114644	0,0375556	0,11098

На рисунку 3.1 зображено графік змін функції пристосованості в біматричній грі через покоління. На графіку видно два періоди стабільності функції.

1. Перше плато: від 21 до 40 поколінь, де значення функції пристосованості коливається приблизно між 6.4 і 6.5.
2. Друге плато: від 40 до 50 поколінь, де значення функції пристосованості коливається приблизно між 6.6 і 6.8.

На рисунку 3.2 зображено графік еволюції функції пристосованості в грі між двома гравцями та природою через покоління. Графік демонструє три періоди стабільності функції.

1. Перше плато: досягнуто приблизно на 10-11 поколінні, де значення функції пристосованості стабілізується на рівні 7.2.
2. Друге плато: спостерігається коливання між 19 та 35 поколіннями, де значення функції пристосованості коливається близько 7.3.
3. Третє плато: починається приблизно з 40 покоління, де значення функції пристосованості стабілізується на рівні 7.5.

Обидва графіки демонструють швидке початкове зростання пристосованості, що вказує на ефективну початкову оптимізацію стратегій. Гра "Два гравці проти природи" починається з вищого рівня пристосованості і досягає більш високих максимальних значень порівняно з біматричною грою. Стабілізація значення пристосованості в обох іграх відбувається на різних рівнях, що може свідчити про різний вплив природи як третього гравця у порівнянні з чисто біматричною грою.

Дуже важливим є також те, що результати, отримані від застосованих стратегій, демонструють наближення до рівноважного стану, який зазвичай спостерігається у чистих стратегіях. Це вказує на те, що обрані стратегії працюють ефективно і взаємодіють оптимально. Такий результат непрямо підтверджує надійність та достовірність проведеного комп'ютерного

моделювання, що свідчить про правильність налаштувань алгоритму і коректність моделювання ігрових умов.

Природа впливає на біматричну гру як у другій моделі, так і у третій моделі. Зведемо розв'язки щодо вибору кращих стратегій гравця А та гравця В для всіх трьох моделей в одну таблицю (див. табл. 3.5).

Таблиця 3.5 Загальна таблиця результатів

Номер моделі	Стратегії гравця А	Стратегії гравця В
№1	0,006; 0,993; 0,001	0,083; 0,905; 0,012
№2	0,0793; 0,6574; 0,0363	0,1255; 0,6046; 0,0429
№3	0,0762644; 0,6285556; 0,05518	0,6114644; 0,0375556; 0,11098

3.4 Висновки до розділу 3

У цьому розділі було здійснено комп'ютерну реалізацію адаптованого генетичного алгоритму для вирішення біматричних ігор із впливом третього гравця (природи). Розроблене програмне забезпечення дозволило провести тестові випробування та отримати результати, які показали, що природа суттєво має вплив на те, як гравці обирають оптимальні стратегії у конфліктних ситуаціях. Результати тестових розрахунків підтвердили ефективність запропонованого підходу та його застосовність для аналізу складних ігрових моделей.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

В заданому розділі буде проведено оцінювання основних характеристик для майбутнього програмного продукту, що спеціалізується на дослідженні демографічного стану.

Дана реалізація буде сприяти проведенню усіх необхідних досліджень, що дасть змогу якісно дослідити питання не лише в Україні, проте у всьому світі.

Також в даному дослідженні показано різні варіанти реалізації для забезпечення найбільш коректної та оптимальної стратегії вибору, що має вплив на економічні фактори та сумісність з майбутнім програмним продуктом. Для цього застосовувався апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) передбачає собою технологію, що дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. ФВА проводиться з метою виявлення резервів зниження витрат за рахунок ефективніших варіантів виробництва, кращого співвідношення між споживчою вартістю виробу та витратами на його виготовлення. Для проведення аналізу використовується економічна, технічна та конструкторська інформація.

Алгоритм функціонально-вартісного аналізу включає в себе визначення послідовності етапів розробки продукту, визначення повних витрат (річних) та кількості робочих часів, визначення джерел витрат та кінцевий розрахунок вартості програмного продукту.

4.1 Постановка задачі проектування

У роботі застосовується метод ФВА для проведення техніко-економічного аналізу розробки системи прогнозу стійкості фінансових показників. Оскільки рішення стосовно проектування та реалізації компонентів, що розробляється,

впливають на всю систему, кожна окрема підсистема має її задовольняти. Тому фактичний аналіз представляє собою аналіз функцій програмного продукту, призначеного для збору, обробки та проведення аналізу даних по компанії.

Технічні вимоги до програмного продукту є наступні:

- функціонування на персональних комп'ютерах із стандартним набором компонентів;
- зручність та зрозумілість для користувача;
- швидкість обробки даних та доступ до інформації в реальному часі;
- можливість зручного масштабування та обслуговування;
- мінімальні витрати на впровадження програмного продукту.

4.2 Обґрунтування функцій програмного продукту

Головна функція F_0 – розробка можливого програмного продукту, яка дозволяє аналізувати різні характеристики, що безпосередньо впливають на стійкість підприємства. Беручи за основу цю функцію, можна виділити наступні:

F_1 – вибір самої програми.

F_2 – якісний аналіз даних.

F_3 – графічні показники.

Кожна з цих функцій має декілька варіантів реалізації:

Функція F_1 :

а) Python.

б) R.

Функція F_2 :

а) Застосування вбудованих функцій.

б) Створення своїх обчислень значень.

Функція F_3 :

а) Використання шаблонних графіків.

б) Створення своїх.

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1).

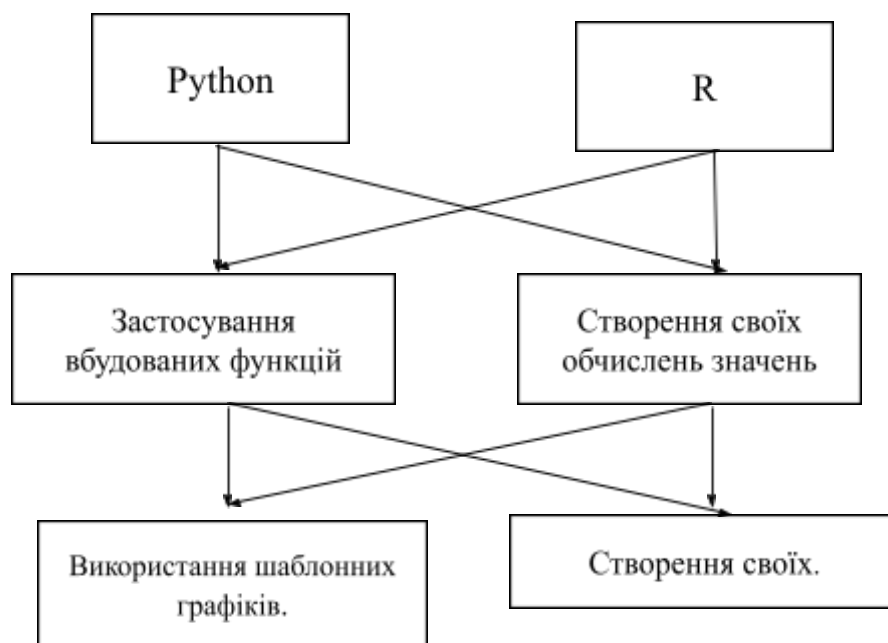


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображає множину всіх можливих варіантів основних функцій. Позитивно-негативна матриця показана в таблиці 4.1.

Таблиця 4.1 - Позитивно-негативна матриця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	<i>A</i>	Загальнодоступна програма, доступність багатьох бібліотек	Необхідність повної реалізації алгоритму
	<i>B</i>	Доступна в реалізації програма для різних обчислень	На написання коду необхідно мати базові навички та вміння

Продовження таблиці 4.1

F_2	A	Доступність та легкість при написанні	Іноді не відповідає задачі яку треба розв'язати
	B	Ідеально описують усі необхідні характеристики	Достатньо затратно реалізовувати свої алгоритми для подальшої реалізації
F_3	A	Загально прийнята реалізація	Іноді не відповідає очікуваним значенням
	B	При виконанні власних досліджень краще може передавати висновки	Необхідно достатньо багато часу для написання програми для побудови та знаходження всього необхідного в задачі.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція F_1 :

Перевагу даємо загальнодоступності. Для спрощення роботи по написанню коду варіант Б має бути відкинутий.

Функція F_2 :

Програма допускає обрання обох варіантів. Можливо використати варіанти А чи Б.

Функція F_3 :

Реалізація першого варіанту є сприйнятливою для програми. Це варіант А.

Таким чином, будемо розглядати такий варіанти реалізації ПП:

$$F_1 a - F_2 a - F_3 a$$

$$F_1 a - F_2 b - F_3 a$$

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.3 Обґрунтування системи параметрів програмного продукту

На основі даних, розглянутих вище, визначаються основні параметри вибору, які будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- $X1$ – швидкодія мови програмування;
- $X2$ – об'єм пам'яті для обчислень та збереження даних;
- $X3$ – час навчання даних;
- $X4$ – потенційний об'єм програмного коду.

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію програмного продукту, як показано у таблиці 4.2.

Таблиця 4.2 - Основні параметри програмного продукту

Назва Параметра	Умовні позначе ння	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	$X1$	оп/мс	60	80	110
Об'єм пам'яті	$X2$	Мб	60	50	30
Час попередньої обробки даних	$X3$	мс	80	70	60
Потенційний об'єм програмного коду	$X4$	кількість рядків коду	35	25	20

За даними таблиці 4.3 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

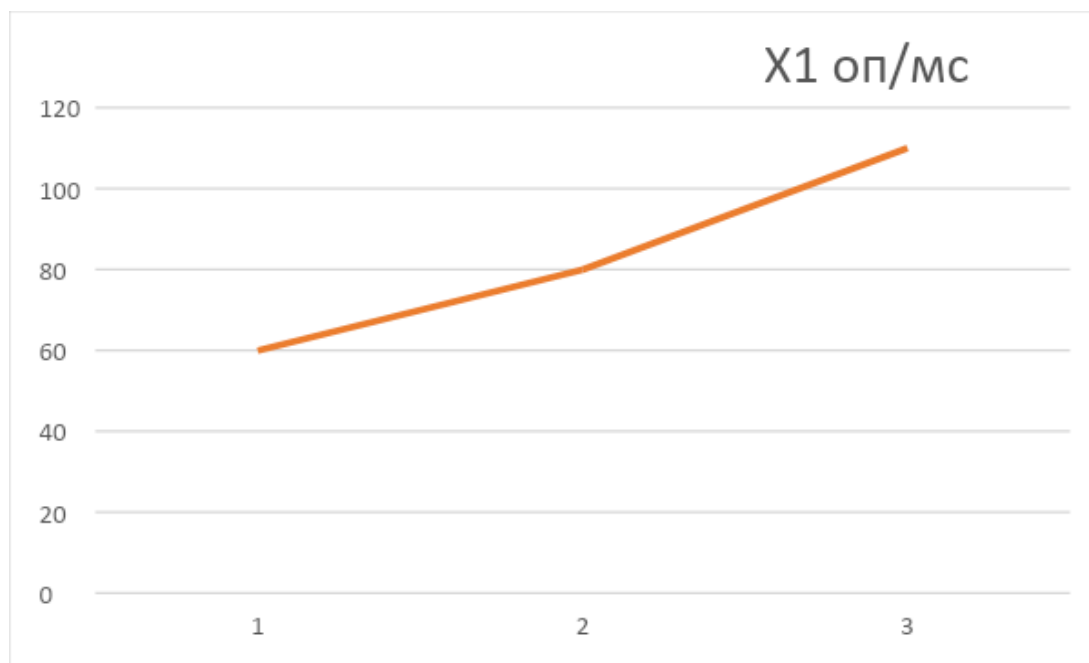


Рисунок 4.2 – X1, швидкодія мови програмування

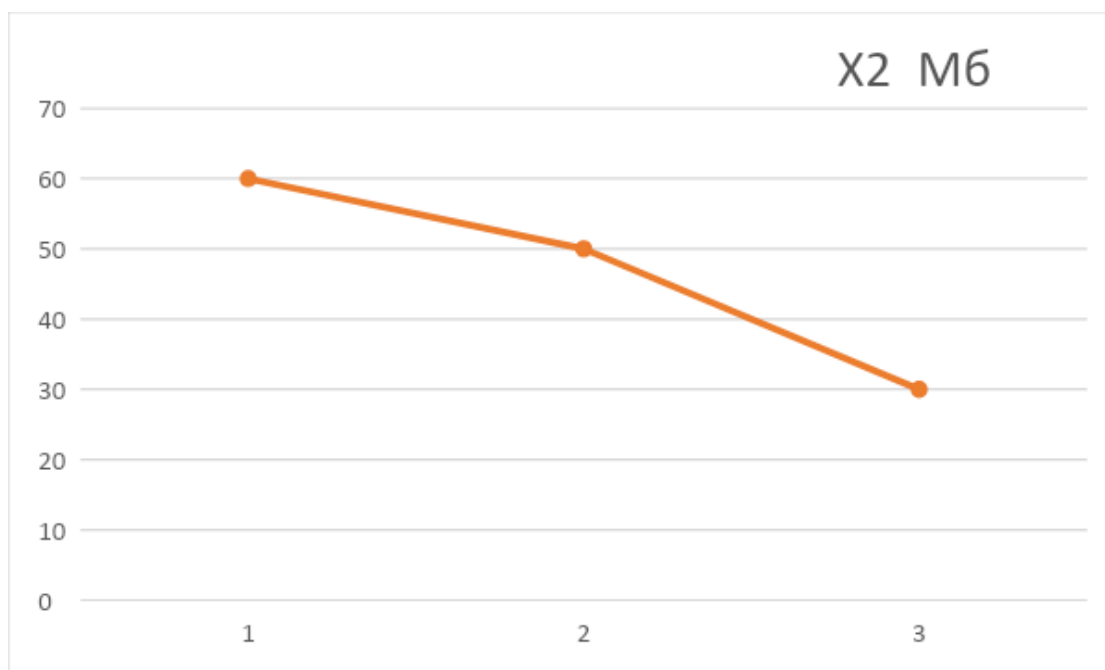


Рисунок 4.3 – X2, об'єм пам'яті

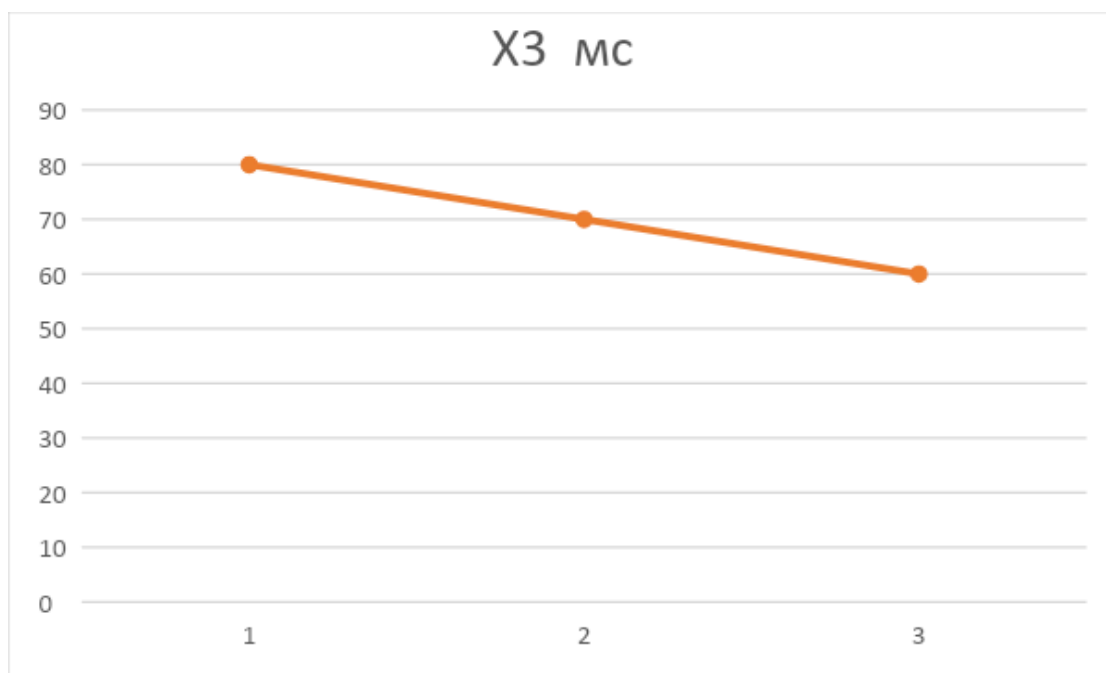


Рисунок 4.4 – X3, час попередньої обробки даних

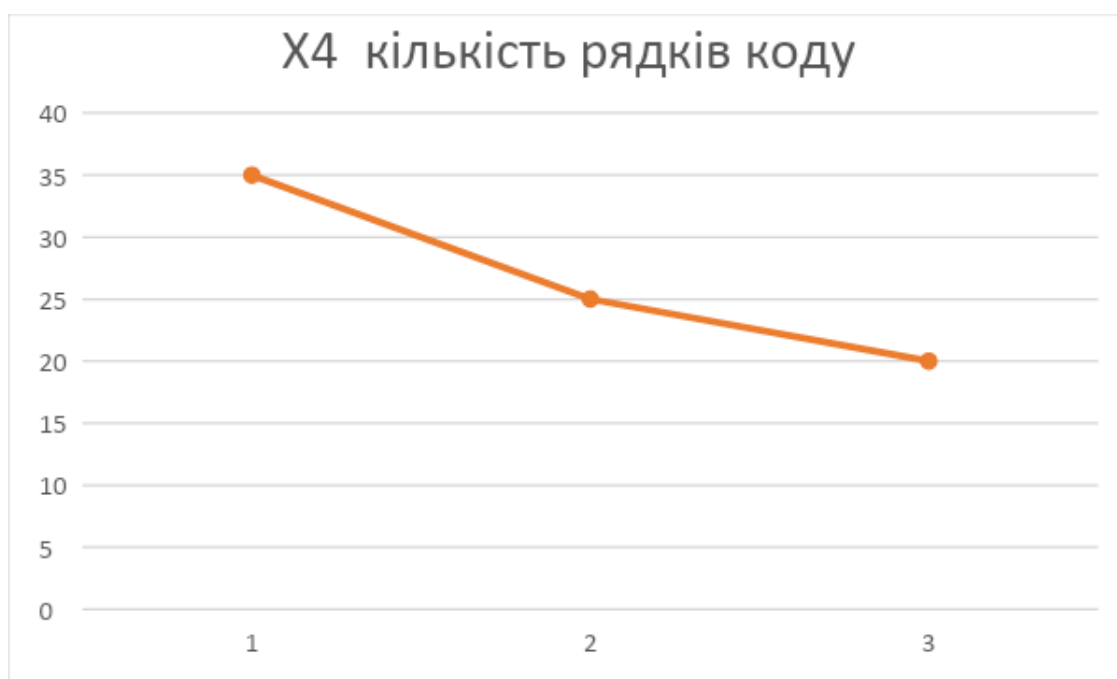


Рисунок 4.5 – X4, потенційний об'єм програмного коду

4.4 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка

програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 - Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
$X1$	Швидкодія мови програмування	Оп/мс	2	2	1	2	1	1	3	11	-6,5	42,25
$X2$	Об'єм пам'яті	Мб	4	4	5	4	3	5	3	28	10,5	110,25
$X3$	Час попередньої обробки даних	мс	1	2	2	1	2	1	1	10	-7,5	56,25

Продовження таблиці 4.3

<i>X4</i>	Потенційний об'єм програмного коду	Кількість рядків коду	3	3	2	3	4	3	3	21	3,5	12,25
	Разом		10	10	10	10	10	10	10	70	0	221

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70, \quad (4.1)$$

де N – число експертів,

n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17,5 \quad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (4.3)$$

Сума відхилень по всіх параметрах повинна дорівнювати 0;

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^N \Delta_i^2 = 221. \quad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{10S}{N^2(n^3 - n)} = \frac{10 \cdot 221}{7^2(4^3 - 4)} = 0,752 > W_k = 0,67. \quad (4.5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Таблиця 4.4 - Попарне порівняння параметрів.

Параметр и	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	<	<	<	<	<	<	<	<	0,5
X1 і X3	<	>	>	<	<	<	>	<	0,5
X1 і X4	<	<	<	<	<	<	<	<	0,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	<	>	<	<	>	<	<	<	0,5
X3 і X4	<	<	<	<	<	<	<	<	0,5

Числове значення, що визначає ступінь переваги i -го параметра над j -тим, a_{ij} визначається по формулі:

$$a_{ij} = \{1.5 \text{ при } X_i > X_j \quad 1.0 \text{ при } X_i = X_j \quad 0.5 \text{ при } X_i < X_j\}. \quad (4.6)$$

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$.

Для кожного параметра зробимо розрахунок вагомості K_{gi} за наступними формулами:

$$K_{bi} = \frac{b_i}{\sum_{i=1}^n b_i} \quad (4.7)$$

$$b_i = \sum_{i=1}^N a_{ij} \quad (4.8)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{bi} = \frac{b'_i}{\sum_{i=1}^n b'_i}, \quad (4.9)$$

$$b'_i = \sum_{j=1}^N a_{ij} b_j \quad (4.10)$$

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 - Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер.	
	X1	X2	X3	X4	b_i	K_{bi}	b_i^1	K_{bi}^1	b_i^2	K_{bi}^2
X1	1	0,5	1	0,5	3	0,19	11	0,19	40,75	0,185
X2	1,5	1	1,5	1,5	5,5	0,34	21,25	0,35	78,625	0,36
X3	1,5	0,5	1	0,5	3	0,19	11	0,19	40,75	0,185
X4	1,5	0,5	1,5	1	4,5	0,28	16,25	0,27	59,875	0,27
Всього:					16	1	59,5	1	220	1

4.5 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2 (Об'єм пам'яті), X3 (час попередньої обробки даних) та X4 (потенційний об'єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X1 (швидкість роботи мови програмування) обрано не найгіршим.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^n K_{bi,j} B_{i,j}, \quad (4.11)$$

де n – кількість параметрів;

K_{bi} – коефіцієнт вагомості i -го параметра;

B_i – оцінка i -го параметра в балах.

Таблиця 4.6 - Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіанти реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	A	X1	10000	21	0,185	3,885
F3	A	X2	64	17	0,36	6,12
	B	X3	128	15	0,185	2,775
F4	A	X4	400	16	0,27	4,32

За даними з таблиці 4.6 за формулою:

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \dots + K_{\text{ТУ}}[F_{zk}], \quad (4.12)$$

визначаємо рівень якості кожного з варіантів:

$$K_{K1} = 3,885 + 6,12 + 4,32 = 14,325 ;$$

$$K_{K2} = 3,885 + 2,775 + 4,32 = 10,98 .$$

Як видно з розрахунків, кращим є 1 варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.6 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;
2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Загальна трудомісткість обчислюється як:

$$T_0 = T_p \cdot K_{\Pi} \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М}, \quad (4.13)$$

де T_p – трудомісткість розробки ПП;

K_{Π} – поправочний коефіцієнт;

$K_{СК}$ – коефіцієнт на складність вхідної інформації;

K_M – коефіцієнт рівня мови програмування;

$K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм;

$K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру ступеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $T_p = 37$ людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_{\Pi} = 1.8$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх семи завдань рівний 1: $K_{СК} = 1$. Оскільки при розробці першого завдання

використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $K_{CT} = 0.9$. Тоді загальна трудомісткість програмування першого завдання дорівнює:

$$T_1 = 37 \cdot 1.8 \cdot 0.9 = 59,94 \text{ людино-днів.}$$

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $T_p = 29$ людино-днів, $K_{II} = 0.9$, $K_{CK} = 1$, $K_{CT} = 0.8$:

$$T_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.}$$

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$$T_I = (59,94 + 20.88 + 4.8 + 20.88) \cdot 8 = 852 \text{ людино-годин.}$$

$$T_{II} = (59,94 + 20.88 + 6.91 + 20.88) \cdot 8 = 868,88 \\ \text{людино-годин.}$$

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 27675 грн., один аналітик в області даних з окладом 31365 грн. Визначимо середню зарплату за годину за формулою:

$$СЧ = \frac{M}{T_m \cdot t} \text{ грн.,} \quad (4.14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів тиждень;

t – кількість робочих годин в день.

$$CЧ = \frac{27675+27675 + 31365}{3 \cdot 21 \cdot 8} = 172,05 \text{ грн.} \quad (4.15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$CЗП = C_{\text{ч}} \cdot T_i \cdot KД, \quad (4.16)$$

де $C_{\text{ч}}$ — величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$KД$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$\text{I. } C_{ЗП} = 172,05 \cdot 945,92 \cdot 1.2 = 195294,64 \text{ грн.}$$

$$\text{II. } C_{ЗП} = 172,05 \cdot 960,48 \cdot 1.2 = 198300,701 \text{ грн.}$$

Відрахування на єдиний соціальний внесок становить 22%:

$$\text{I. } C_{\text{ВІД}} = C_{ЗП} \cdot 0.22 = 195294,64 \cdot 0.22 = 42964,82 \text{ грн.}$$

$$\text{II. } C_{\text{ВІД}} = C_{ЗП} \cdot 0.22 = 198300,701 \cdot 0.22 = 43626,15 \text{ грн.}$$

Тепер визначимо витрати на оплату однієї машино-години. (C_M)

Так як одна ЕОМ обслуговує одного програміста з окладом 27675 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$C_{\Gamma} = 12 \cdot M \cdot K_3 = 12 \cdot 27675 \cdot 0,2 = 66420 \text{ грн.}$$

З урахуванням додаткової заробітної плати:

$$C_{ЗП} = C_{\Gamma} \cdot (1 + K_3) = 66420 \cdot (1 + 0.2) = 79704 \text{ грн.}$$

Відрахування на соціальний внесок:

$$C_{\text{ВІД}} = C_{\text{ЗП}} \cdot 0.22 = 79704 \cdot 0.22 = 17534,88 \text{ грн.}$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$$C_A = K_{\text{ТМ}} \cdot K_A \cdot \Pi_{\text{ПР}} = 1.2 \cdot 0.25 \cdot 28500 = 8550 \text{ грн.},$$

де $K_{\text{ТМ}}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

K_A – річна норма амортизації;

$\Pi_{\text{ПР}}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$C_P = K_{\text{ТМ}} \cdot \Pi_{\text{ПР}} \cdot K_P = 1.4 \cdot 10000 \cdot 0.08 = 1120 \text{ грн.},$$

де K_P – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$\begin{aligned} T_{\text{ЕФ}} &= (D_K - D_B - D_C - D_P) \cdot t_z \cdot K_B = (365 - 104 - 12 - 16) \cdot 8 \cdot 0.35 = \\ &= 627,2 \text{ години,} \end{aligned}$$

де D_K – календарна кількість днів у році;

D_B, D_C – відповідно кількість вихідних та святкових днів;

D_P – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

K_B – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$C_{\text{ЕЛ}} = T_{\text{ЕФ}} \cdot N_{\text{С}} \cdot K_3 \cdot \Pi_{\text{ЕН}} = 627,2 \cdot 0,2 \cdot 0,3 \cdot 5,21 = 196,06 \text{ грн.},$$

де $N_{\text{С}}$ – середньо-споживча потужність приладу;

K_3 – коефіцієнтом зайнятості приладу;

$\Pi_{\text{ЕН}}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$C_{\text{Н}} = \Pi_{\text{ПР}} \cdot 0,67 = 10000 \cdot 0,67 = 6700 \text{ грн.}$$

Тоді, річні експлуатаційні витрати будуть:

$$C_{\text{ЕКС}} = C_{\text{ЗП}} + C_{\text{ВІД}} + C_{\text{А}} + C_{\text{Р}} + C_{\text{ЕЛ}} + C_{\text{Н}}, \quad (4.17)$$

$$C_{\text{ЕКС}} = 79704 + 17534,88 + 8550 + 1120 + 196,06 + 6700 = 113804,94 \text{ грн.}$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{\text{М-Г}} = C_{\text{ЕКС}} / T_{\text{ЕФ}} = 113804,94 / 627,2 = 181,45 \text{ грн/год.}$$

Оскільки в даному випадку всі роботи, які пов'язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$C_M = C_{M-\Gamma} \cdot T, \quad (4.18)$$

$$\text{I. } C_M = 181,45 \cdot 852 = 154595,4 \text{ грн.}$$

$$\text{II. } C_M = 181,45 \cdot 868,88 = 157658,3 \text{ грн.}$$

Накладні витрати складають 67% від заробітної плати:

$$C_H = C_{3\Pi} \cdot 0,67, \quad (4.19)$$

$$\text{I. } C_H = 195294,6 \cdot 0,67 = 130846,98 \text{ грн.}$$

$$\text{II. } C_H = 198300,7 \cdot 0,67 = 132861,47 \text{ грн.}$$

Отже, вартість розробки ПП за варіантами становить:

$$C_{\Pi\Pi} = C_{3\Pi} + C_{\text{ВІД}} + C_M + C_H, \quad (4.20)$$

$$\text{I. } C_{\Pi\Pi} = 195294,6 + 42964,82 + 154595,4 + 130846,98 = 523701,8 \text{ грн.}$$

$$\text{II. } C_{\Pi\Pi} = 198300,7 + 43626,15 + 157658,3 + 132861,47 = 532446,62 \text{ грн.}$$

4.7 Вибір кращого варіанту ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{\text{ТЕР}j} = K_{Kj} / C_{\Phi j}, \quad (4.21)$$

$$K_{\text{TEP1}} = 14,325 / 523701,8 = 2,735 \cdot 10^{-5},$$

$$K_{\text{TEP2}} = 10,98 / 532446,62 = 2,062 \cdot 10^{-5}.$$

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{\text{TEP1}} = 2,735 \cdot 10^{-5}$.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишилися після першого відбору двох варіантів виконання програмного комплексу оптимальним є перший варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості

$$K_{\text{TEP}} = 2,735 \cdot 10^{-5}.$$

Цей варіант реалізації програмного продукту має такі параметри:

- Вибір програмного продукту – Python;
- Реалізація важливої постановки з допомогою вбудованих функцій;
- Використання стандартного інтерфейсу для побудови значень.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, швидко реалізацію програми та доступний функціонал для роботи.

4.8 Висновки до розділу 4

В даному розділі було проведено повний функціонально-вартісний аналіз програмного продукту. Також було знайдено оцінку основних функцій програмного продукту.

В результаті виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, було визначено та проведено оцінку основних

функцій програмного продукту, а також знайдено параметри, які його характеризують.

На основі аналізу вибрано варіант реалізації програмного продукту.

ВИСНОВКИ

У даній дипломній роботі було розглянуто та досліджено проблему моделювання стратегічної взаємодії гравців у біматричних іграх з урахуванням впливу природи як третього гравця. У процесі роботи було досягнуто наступні результати.

Було проведено адаптацію генетичного алгоритму для вирішення задач біматричних ігор із урахуванням впливу природи. Генетичний алгоритм було обрано через його здатність ефективно вирішувати задачі оптимізації в умовах невизначеності та мінливості середовища.

Створено математичну модель, яка описує взаємодію двох гравців у біматричній грі з врахуванням впливу природи. Ця модель включає розрахунок вигравів кожного гравця залежно від їх стратегій та стану середовища, який впливає на гру. Модель враховує можливість використання змішаних стратегій гравцями, що підвищує гнучкість і точність моделювання.

Було розроблено програмне забезпечення (Python), яке реалізує адаптований генетичний алгоритм для вирішення задач біматричних ігор. Програмний код структуровано у вигляді окремих класів, кожен з яких виконує певні функції, що дозволяє легко модифікувати і розширювати програму для подальших досліджень.

Проведено тестові випробування розробленого програмного забезпечення на прикладі біматричних ігор.

Було досліджено кілька сценаріїв розвитку подій, що дозволило оцінити вплив природи на оптимальні стратегії гравців.

Результати показали, що врахування природи як третього гравця суттєво впливає на результати гри, що підтверджує необхідність врахування зовнішніх факторів у моделюванні стратегічних взаємодій, що свідчить про ефективність використання генетичних алгоритмів для вирішення задач біматричних ігор з врахуванням впливу зовнішніх факторів.

Розроблений підхід дозволяє більш точно моделювати реальні ситуації, де гра не обмежується лише діями гравців, а включає вплив середовища.

Перспективами подальших досліджень можуть бути розширення моделі для більш складних ігор, включення додаткових факторів впливу, а також дослідження можливостей використання інших еволюційних алгоритмів для вирішення подібних задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Von Neumann J., Morgenstern O. Theory of Games and Economic Behavior. Princeton: Princeton University Press, 1944.
2. Nash J. Equilibrium in Non-Cooperative Games. Journal of Economic Theory, 1950.
3. Axelrod R. The Evolution of Cooperation. New York: Basic Books, 1984.
4. Smith J. M. Evolution and the Theory of Games. Cambridge: Cambridge University Press, 1982.
5. Shoham Y., Leyton-Brown K. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge: Cambridge University Press, 2009.
6. Nasar S. A Beautiful Mind: The Life of Mathematical Genius and Nobel Laureate John Nash. New York: Simon & Schuster, 1998.
7. Myerson R. Game Theory: Analysis of Conflict. Cambridge: Harvard University Press, 1991.
8. Holland J. Adaptation in Natural and Artificial Systems. Michigan: University of Michigan Press, 1975.
9. Goldberg D. Genetic Algorithms in Search, Optimization, and Machine Learning. Redwood City: Addison-Wesley, 1989.
10. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Berlin: Springer, 1996.
11. Baker T. A Taxonomy of Genetic Algorithms. Journal of Heuristics, 1996.
12. Smith K. Modern Heuristic Techniques for Combinatorial Problems. Oxford: Oxford University Press, 1995.
13. Тверський А., Канеман Д. Судження під невизначеністю: евристики та упередження. Науковий журнал, 1974.

- 14.Майерсон Р. Вступ до теорії ігор. Кембридж: Cambridge University Press, 1991.
- 15.Мюллер А. Прийняття рішень в умовах невизначеності: від класичної теорії до поведінкової економіки. Берлін: Springer, 2010.
- 16.Ігнатенко О. Теорія ігор: суть і напрями застосування. Київ: Інститут програмних систем НАН України, 2020.
- 17.Капустян В. О., Мажара Г. А. КПП ім. Ігоря Сікорського. Оптимальне керування та теорія ігор в економіці: методичні вказівки щодо виконання курсової роботи для студентів галузі знань 0305 – «Економіка та підприємництво» напряму. Київ: КПП, 2022. URL: https://ela.kpi.ua/bitstream/123456789/47796/1/Optymalne_2022.pdf
- 18.Макаренко В. П., Лапшина Н. О. Теоретичні основи біматричних ігор та їхнє застосування у моделюванні соціально-економічних процесів. Київ : НТУУ "КПІ", 2018. 128 с.
- 19.Козленко А. В. Застосування генетичних алгоритмів для оптимізації стратегій у біматричних іграх з урахуванням випадкових факторів. Вісник Київського національного університету імені Тараса Шевченка. Серія: Математика. 2020. № 2 (45). С. 123-130.
- 20.Learn Python Tutorial | Python Programming Language - javatpoint. www.javatpoint.com. URL: <https://www.javatpoint.com/python-tutorial>
- 21.Introduction to Game Theory - InnovationManagement. InnovationManagement. URL: <https://innovationmanagement.se/2020/11/06/introduction-to-game-theory/>.
- 22.Учасники проєктів Вікімедіа. Оскар Моргенштерн – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Оскар_Моргенштерн.
- 23.File:Cournot Duopoly Nash equilibrium.png - Wikimedia Commons. Wikimedia Commons. URL: https://commons.wikimedia.org/wiki/File:Cournot_Duopoly_Nash_equilibrium.png.
- 24.Nash J. F. Non-Cooperative Games. The Annals of Mathematics, Second Series, Vol. 54, No. 2 (Sep., 1951), pp. 286-295.

25. Gómez L., Vercher J., Camacho E. F. Evolutionary Observer Ensemble for Leak Diagnosis in Water Pipelines. *Water*, 2019. URL: https://www.researchgate.net/publication/337730518_Evolutionary_Observer_Ensemble_for_Leak_Diagnosis_in_Water_Pipelines
26. Mitchell M. An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1996.
27. Nordstrom J. A. F. Introduction to Game Theory: A Discovery Approach. Linfield College, 2021. URL: [https://math.libretexts.org/Bookshelves/Applied_Mathematics/Introduction_to_Game_Theory%3A_A_Discovery_Approach_\(Nordstrom\)](https://math.libretexts.org/Bookshelves/Applied_Mathematics/Introduction_to_Game_Theory%3A_A_Discovery_Approach_(Nordstrom))

ДОДАТОК А

```
```python
```

```
import numpy as np
```

```
from genetic_algorithm_bimatrix_game import GeneticAlgorithmBimatrixGame
```

```
from genetic_algorithm_zero_sum_game import GeneticAlgorithmZeroSumGame
```

```
if __name__ == "__main__":
```

```
 values_a = np.array([[3, 1, 2], [2, 3, 1], [0, 4, 1]])
```

```
 values_b = np.array([[2, 1, 0], [2, 4, 3], [4, 2, 1]])
```

```
 values_nature = np.array([[8, 4, 2], [2, 8, 4], [1, 2, 8]])
```

```
 genetic_algorithm_bimatrix_game = GeneticAlgorithmBimatrixGame(values_a,
values_b)
```

```
 bimatrix_strategy = genetic_algorithm_bimatrix_game.launch()
```

```
 genetic_algorithm_zero_sum_game =
GeneticAlgorithmZeroSumGame(values_nature)
```

```
strategy_against_nature = genetic_algorithm_zero_sum_game.launch()
```

```
strategy_a = 0.6 * bimatrix_strategy[0] + 0.4 * strategy_against_nature
```

```
strategy_b = 0.6 * bimatrix_strategy[1] + 0.4 * strategy_against_nature
```

```
print("Strategy against nature:", strategy_against_nature)
```

```
print("Player A:", bimatrix_strategy[0])
```

```
print("Player B:", bimatrix_strategy[1])
```

```
print("Strategy of player A:", strategy_a)
```

```
print("Strategy of player B:", strategy_b)
```

```
'''
```

**ДОДАТОК Б**

```
```python

import numpy as np

import random

import matplotlib.pyplot as plt


class Chromosome:

    def __init__(self, genes_p, genes_k, fitness):

        self.genes_p = genes_p

        self.genes_k = genes_k

        self.fitness = fitness


    def __lt__(self, other):

        return self.fitness < other.fitness


class GeneticAlgorithmBimatrixGame:
```

```
    def __init__(self, values_a, values_b):
```

```

self.values_a = values_a

self.values_b = values_b


def launch(self):

    chromosomes = self.create_chromosomes(100)

    print("Bimatrix game:")

    pi = np.zeros(3)

    ki = np.zeros(3)

    fitness_values = []


    for generation in range(50):

        chromosomes.sort(reverse=True)

        best_fitness = chromosomes[0].fitness

        fitness_values.append(best_fitness)

        print(f"Generation {generation}: Fitness value: {best_fitness}, P values:
{chromosomes[0].genes_p}, K values: {chromosomes[0].genes_k}")

        pi = chromosomes[0].genes_p

        ki = chromosomes[0].genes_k

        self.crossover_population(chromosomes)

        self.mutate_population(chromosomes)

```

```
self.plot_fitness(fitness_values)

return [pi, ki]


def create_chromosomes(self, number):

    return [self.create_chromosome() for _ in range(number)]


def create_chromosome(self):

    genes_p = self.create_genes()

    genes_k = self.create_genes()

    fitness = self.calculate_fitness(genes_p, genes_k)

    return Chromosome(genes_p, genes_k, fitness)


def create_genes(self):

    random_numbers = np.random.random(3)

    return random_numbers / random_numbers.sum()


def calculate_fitness(self, genes_p, genes_k):

    fitness = 0
```

```

for i in range(len(genes_p)):

    for j in range(len(genes_k)):

        fitness += self.values_a[i][j] * genes_p[i] * genes_k[j]

        fitness += self.values_b[i][j] * genes_p[i] * genes_k[j]

    return fitness

def crossover_population(self, chromosomes):

    for i in range(0, 60, 2):

        self.crossover_chromosomes(chromosomes[i], chromosomes[i + 1])

def crossover_chromosomes(self, first_chromosome, second_chromosome):

    crossover_genes_p = self.crossover_genes(first_chromosome.genes_p,
second_chromosome.genes_p)

    if crossover_genes_p is not None:

        first_chromosome.genes_p, second_chromosome.genes_p = crossover_genes_p

    crossover_genes_k = self.crossover_genes(first_chromosome.genes_k,
second_chromosome.genes_k)

    if crossover_genes_k is not None:

        first_chromosome.genes_k, second_chromosome.genes_k = crossover_genes_k

```

if crossover_genes_p is not None or crossover_genes_k is not None:

```
    first_chromosome.fitness = self.calculate_fitness(first_chromosome.genes_p,
first_chromosome.genes_k)
```

```
    second_chromosome.fitness =
self.calculate_fitness(second_chromosome.genes_p, second_chromosome.genes_k)
```

```
def crossover_genes(self, first_genes, second_genes):
```

```
    for i in range(len(first_genes)):
```

```
        for j in range(len(second_genes)):
```

```
            if i != j:
```

```
                if first_genes[i] + second_genes[j] < 1 and first_genes[j] + second_genes[i]
< 1:
```

```
                    temp = first_genes[i]
```

```
                    first_genes[i] = second_genes[i]
```

```
                    second_genes[i] = temp
```

```
                    third_index = 3 - i - j
```

```
                    first_genes[third_index] = round(1 - first_genes[i] - first_genes[j], 3)
```

```
                    second_genes[third_index] = round(1 - second_genes[i] -
second_genes[j], 3)
```

```
            return first_genes, second_genes
```

```
return None
```

```
def mutate_population(self, chromosomes):
```

```
    for i in range(60, len(chromosomes)):
```

```
        self.mutate_chromosome(chromosomes[i])
```

```
def mutate_chromosome(self, chromosome):
```

```
    chromosome.genes_p = self.mutate_genes(chromosome.genes_p)
```

```
    chromosome.genes_k = self.mutate_genes(chromosome.genes_k)
```

```
        chromosome.fitness = self.calculate_fitness(chromosome.genes_p,
chromosome.genes_k)
```

```
def mutate_genes(self, genes):
```

```
    gene_to_mutate = random.randint(0, 2)
```

```
    gene_to_leave_as_is = random.choice([i for i in range(3) if i != gene_to_mutate])
```

```
    while True:
```

```
        new_gene_value = round(random.random(), 3)
```

```
        if new_gene_value + genes[gene_to_leave_as_is] < 1:
```

```
            break
```

```
    third_index = 3 - gene_to_mutate - gene_to_leave_as_is
```

```
genes[gene_to_mutate] = new_gene_value

genes[third_index] = round(1 - new_gene_value - genes[gene_to_leave_as_is], 3)

return genes


def plot_fitness(self, fitness_values):

    plt.figure(figsize=(10, 6))

    plt.plot(fitness_values, label='Fitness over generations')

    plt.xlabel('Generation')

    plt.ylabel('Fitness')

    plt.title('Fitness Evolution in Bimatrix Game')

    plt.legend()

    plt.grid(True)

    plt.show()

'''
```

ДОДАТОК В

```
```python

import numpy as np

import random

import matplotlib.pyplot as plt

class Chromosome:

 def __init__(self, genes_p, fitness):

 self.genes_p = genes_p

 self.fitness = fitness

 def __lt__(self, other):

 return self.fitness < other.fitness

class GeneticAlgorithmZeroSumGame:

 def __init__(self, nature_values):

 self.nature_values = nature_values
```

```

def launch(self):

 chromosomes = self.create_chromosomes(100)

 print("Zero-sum game:")

 xi = np.zeros(3)

 v = 0

 fitness_values = []

 for generation in range(30):

 chromosomes.sort()

 best_fitness = chromosomes[0].fitness

 fitness_values.append(best_fitness)

 print(f"Generation {generation}: Fitness value: {best_fitness}, P values:
{chromosomes[0].genes_p}")

 xi = chromosomes[0].genes_p

 v = 1 / chromosomes[0].fitness

 self.crossover_population(chromosomes)

 self.mutate_population(chromosomes)

 xi *= v

```

```
self.plot_fitness(fitness_values)

return xi

def create_chromosomes(self, number):

 return [self.create_chromosome() for _ in range(number)]

def create_chromosome(self):

 genes_p = self.create_genes()

 fitness = self.calculate_fitness(genes_p)

 return Chromosome(genes_p, fitness)

def create_genes(self):

 while True:

 genes = np.random.random(3)

 if self.validate_generated_genes(genes):

 return genes

def validate_generated_genes(self, genes):

 for column in range(self.nature_values.shape[1]):
```

```
if np.dot(self.nature_values[:, column], genes) < 1:
```

```
 return False
```

```
return True
```

```
def calculate_fitness(self, genes_p):
```

```
 return genes_p.sum()
```

```
def crossover_population(self, chromosomes):
```

```
 for i in range(0, 60, 2):
```

```
 self.crossover_chromosomes(chromosomes[i], chromosomes[i + 1])
```

```
def crossover_chromosomes(self, first_chromosome, second_chromosome):
```

```
 crossover_genes_p = self.crossover_genes(first_chromosome.genes_p,
second_chromosome.genes_p)
```

```
 if crossover_genes_p is not None:
```

```
 first_chromosome.genes_p, second_chromosome.genes_p = crossover_genes_p
```

```
 first_chromosome.fitness = self.calculate_fitness(first_chromosome.genes_p)
```

```

second_chromosome.fitness =
self.calculate_fitness(second_chromosome.genes_p)

```

```

def crossover_genes(self, first_genes, second_genes):

 for i in range(len(first_genes)):

 temp = first_genes[i]

 first_genes[i] = second_genes[i]

 second_genes[i] = temp

 if self.validate_generated_genes(first_genes) and
self.validate_generated_genes(second_genes):

 return first_genes, second_genes

 return None

```

```

def mutate_population(self, chromosomes):

```

```

 for i in range(60, len(chromosomes)):

 self.mutate_chromosome(chromosomes[i])

```

```

def mutate_chromosome(self, chromosome):

```

```

 chromosome.genes_p = self.mutate_genes(chromosome.genes_p)

 chromosome.fitness = self.calculate_fitness(chromosome.genes_p)

```

```
def mutate_genes(self, genes):

 while True:

 genes_res = genes.copy()

 gene_to_mutate = random.randint(0, 2)

 new_gene_value = round(random.random(), 3)

 genes_res[gene_to_mutate] = new_gene_value

 if self.validate_generated_genes(genes_res):

 return genes_res

def plot_fitness(self, fitness_values):

 plt.figure(figsize=(10, 6))

 plt.plot(fitness_values, label='Fitness over generations')

 plt.xlabel('Generation')

 plt.ylabel('Fitness')

 plt.title('Fitness Evolution in Zero-Sum Game')

 plt.legend()

 plt.grid(True)

 plt.show()
```

**ДОДАТОК Г**

```
import numpy as np
```

```
import random
```

```
import matplotlib.pyplot as plt
```

```
class Chromosome:
```

```
 def __init__(self, genes_p, genes_k, fitness):
```

```
 self.genes_p = genes_p
```

```
 self.genes_k = genes_k
```

```
 self.fitness = fitness
```

```
 def __lt__(self, other):
```

```
 return self.fitness < other.fitness
```

```
class GeneticAlgorithmTwoPlayersAndNatureGame:
```

```
 def __init__(self, values_a, values_b, values_nature):
```

```
 self.values_a = values_a
```

```
 self.values_b = values_b
```

```

self.values_nature = values_nature

def launch(self):

 chromosomes = self.create_chromosomes(100)

 print("Two players VS Nature:")

 pi = np.zeros(3)

 ki = np.zeros(3)

 fitness_values = []

 for generation in range(50):

 chromosomes.sort(reverse=True)

 best_fitness = chromosomes[0].fitness

 fitness_values.append(best_fitness)

 print(f"Generation {generation}: Fitness value: {best_fitness}, P values:
{chromosomes[0].genes_p}, K values: {chromosomes[0].genes_k}")

 pi = chromosomes[0].genes_p

 ki = chromosomes[0].genes_k

 self.crossover_population(chromosomes)

 self.mutate_population(chromosomes)

```

```
self.plot_fitness(fitness_values)

return [pi, ki]

def create_chromosomes(self, number):

 return [self.create_chromosome() for _ in range(number)]

def create_chromosome(self):

 genes_p = self.create_genes()

 genes_k = self.create_genes()

 fitness = self.calculate_fitness(genes_p, genes_k)

 return Chromosome(genes_p, genes_k, fitness)

def create_genes(self):

 random_numbers = np.random.random(3)

 return random_numbers / random_numbers.sum()

def calculate_fitness(self, genes_p, genes_k):

 players_sum = 0

 nature_sum = 0
```

```

for i in range(len(genes_p)):

 for j in range(len(genes_k)):

 players_sum += self.values_a[i][j] * genes_p[i] * genes_k[j]

 players_sum += self.values_b[i][j] * genes_p[i] * genes_k[j]

 nature_sum += self.values_nature[i][j] * genes_p[i] * 0.33

 nature_sum += self.values_nature[i][j] * genes_k[i] * 0.33

 return 0.6 * players_sum + 0.4 * nature_sum

def crossover_population(self, chromosomes):

 for i in range(0, 60, 2):

 self.crossover_chromosomes(chromosomes[i], chromosomes[i + 1])

def crossover_chromosomes(self, first_chromosome, second_chromosome):

 crossover_genes_p = self.crossover_genes(first_chromosome.genes_p,
second_chromosome.genes_p)

 if crossover_genes_p is not None:

 first_chromosome.genes_p, second_chromosome.genes_p = crossover_genes_p

 crossover_genes_k = self.crossover_genes(first_chromosome.genes_k,
second_chromosome.genes_k)

```

if crossover\_genes\_k is not None:

first\_chromosome.genes\_k, second\_chromosome.genes\_k = crossover\_genes\_k

if crossover\_genes\_p is not None or crossover\_genes\_k is not None:

first\_chromosome.fitness = self.calculate\_fitness(first\_chromosome.genes\_p,  
first\_chromosome.genes\_k)

second\_chromosome.fitness =  
self.calculate\_fitness(second\_chromosome.genes\_p, second\_chromosome.genes\_k)

def crossover\_genes(self, first\_genes, second\_genes):

for i in range(len(first\_genes)):

for j in range(len(second\_genes)):

if i != j:

if first\_genes[i] + second\_genes[j] < 1 and first\_genes[j] + second\_genes[i]  
< 1:

temp = first\_genes[i]

first\_genes[i] = second\_genes[i]

second\_genes[i] = temp

third\_index = 3 - i - j

first\_genes[third\_index] = round(1 - first\_genes[i] - first\_genes[j], 3)

```

second_genes[third_index] = round(1 - second_genes[i] -
second_genes[j], 3)

```

```

 return first_genes, second_genes

```

```

 return None

```

```

def mutate_population(self, chromosomes):

```

```

 for i in range(60, len(chromosomes)):

```

```

 self.mutate_chromosome(chromosomes[i])

```

```

def mutate_chromosome(self, chromosome):

```

```

 chromosome.genes_p = self.mutate_genes(chromosome.genes_p)

```

```

 chromosome.genes_k = self.mutate_genes(chromosome.genes_k)

```

```

 chromosome.fitness = self.calculate_fitness(chromosome.genes_p,
chromosome.genes_k)

```

```

def mutate_genes(self, genes):

```

```

 gene_to_mutate = random.randint(0, 2)

```

```

 gene_to_leave_as_is = random.choice([i for i in range(3) if i != gene_to_mutate])

```

```

 while True:

```

```

 new_gene_value = round(random.random(), 3)

```

```
 if new_gene_value + genes[gene_to_leave_as_is] < 1:

 break

 third_index = 3 - gene_to_mutate - gene_to_leave_as_is

 genes[gene_to_mutate] = new_gene_value

 genes[third_index] = round(1 - new_gene_value - genes[gene_to_leave_as_is], 3)

 return genes

def plot_fitness(self, fitness_values):

 plt.figure(figsize=(10, 6))

 plt.plot(fitness_values, label='Fitness over generations')

 plt.xlabel('Generation')

 plt.ylabel('Fitness')

 plt.title('Fitness Evolution in Two Players VS Nature Game')

 plt.legend()

 plt.grid(True)

 plt.show()
```

## ДОДАТОК Д

**ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, МОДЕЛІ І МЕТОДИ  
БІМАТРИЧНОЇ ГРИ З УРАХУВАННЯМ ВПЛИВУ  
«ПРИРОДИ» ЯК ТРЕТЬОГО ГРАВЦЯ**

Виконала: студентка групи КА-05 Ровенко О.М.

Керівник: доцент, к.ф.-м.н. Барановська Л.В.

I

**АКТУАЛЬНІСТЬ**

У сучасному світі, де інформаційні технології відіграють все більш важливу роль у прийнятті стратегічних рішень, дослідження впливу "природи" як третього гравця в біматричних іграх набуває особливого значення. Це дозволяє краще розуміти динаміку взаємодії між різними учасниками та розробляти більш ефективні стратегії в умовах невизначеності та конкуренції.

2

## ОБ'ЄКТ

## ПРЕДМЕТ

## МЕТА

## ДОСЛІДЖЕННЯ

Об'єктом дослідження є інформаційні технології та моделі біматричних ігор, що є ключовими компонентами в процесі прийняття стратегічних рішень

Предметом дослідження є вплив "природи" як третього гравця в біматричних іграх

Мета дослідження полягає в розробці та аналізі моделей взаємодії гравців у біматричних іграх з урахуванням третього гравця з метою виявлення оптимальних стратегій

3

## ПОСТАНОВКА ЗАДАЧІ

***Потрібно дослідити вплив «природи» у ролі третього гравця у результаті біматричної гри двох гравців.***

Для цього необхідно:

1. Провести аналіз теоретичних основ теорії ігор, біматричних ігор та генетичного алгоритму.
2. Розробити математичну модель біматричної гри з урахуванням впливу природи, як третього гравця.
3. Реалізувати генетичний алгоритм для розрахунку оптимальних стратегій гравців без урахування та з урахуванням природи.
4. Реалізувати комп'ютерну програму для моделювання і аналізу біматричних ігор із впливом природи.
5. Провести аналіз отриманих результатів і зробити висновки.

4

## БІМАТРИЧНІ ІГРИ ТА ЇХ ЗАСТОСУВАННЯ

### Сутність біматричних ігор

Біматричні ігри є важливим класом ігор, які моделюють стратегічну взаємодію двох гравців з урахуванням їх індивідуальних переваг та виграшів. Вони знаходять широке застосування в моделюванні економічної конкуренції, міжнародних відносин, військових конфліктів та інших сферах.

### Застосування біматричних ігор

Реальні ігрові ситуації часто відбуваються в динамічному та невизначеному середовищі, яке може суттєво впливати на вибір стратегій гравців та їх виграші. Врахування впливу навколишнього середовища на процес прийняття рішень в біматричних іграх є актуальною науковою проблемою.

5

Складові елементи біматричної моделі :  $\langle N, SN, UN \rangle$  ,  
що представляють собою:

$N$  – гравці або учасники гри (у подальшому розглядатимемо, що  $N=2$ ).

$SN$  – набір можливих дій або виборів, доступних кожному гравцю під час гри.

$UN$  – функції виграшів гравців:  $UN : X \times Y \rightarrow R$ , де  $X, Y \in SN$

6

## РІВНОВАГА НЕСА

У грі  $G_{bi} = \{S_i, U_i, iN\}$ , стратегії  $(s^*_1, s^*_2, \dots, s^*_N)$  створюють рівновагу Неша, якщо для кожного гравця  $i$  стратегія  $s^*_i$  є його найкращою відповіддю на набір стратегій  $s^*_i$ -і інших гравців. Тобто,  $s^*_i$  вирішує задачу.

$$\max_{s_i \in S_i} U_i(s_1^*, \dots, s_{i-1}^*, s_i, s_{i+1}^*, \dots, s_n^*)$$

7

$$G_{bi} = \begin{pmatrix} a_{11}, b_{11} & a_{12}, b_{12} & \dots & a_{1n}, b_{1n} \\ a_{21}, b_{21} & a_{22}, b_{22} & \dots & a_{2n}, b_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1}, b_{m1} & a_{m2}, b_{m2} & \dots & a_{mn}, b_{mn} \end{pmatrix}$$

ПЛАТІЖНА МАТРИЦЯ БІМАТРИЧНОЇ ГРИ

8

ВИГРАШ ГРАВЦЯ  $A$  У БІМАТРИЧНІЙ ГРІ

$$U_A(A, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n a_{ij} p_i q_j$$

ВИГРАШ ГРАВЦЯ  $B$  У БІМАТРИЧНІЙ ГРІ

$$U_B(B, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n b_{ij} p_i q_j$$

$$\sum_{i=1}^m p_i = 1, \quad \sum_{j=1}^n q_j = 1$$

ВИГРАШ ГРАВЦЯ  $A$  ЧИ  $B$  У ГРІ З ПРИРОДОЮ

$$U(Gp, S_A, S_B) = \sum_{i=1}^m \sum_{j=1}^n a_{ij}^{\wedge} p_i^{\wedge} q_j^{\wedge}$$

$$\sum_{i=1}^m p_i^{\wedge} = 1, \quad \sum_{j=1}^n q_j^{\wedge} = 1$$

9

## ГЕНЕТИЧНІ АЛГОРИТМИ ДЛЯ ВИРІШЕННЯ БІМАТРИЧНИХ ІГОР

### Моделювання еволюції

Генетичні алгоритми моделюють процеси природної еволюції, такі як спадковість, мутації та природний відбір, для пошуку оптимальних рішень в складних просторах.

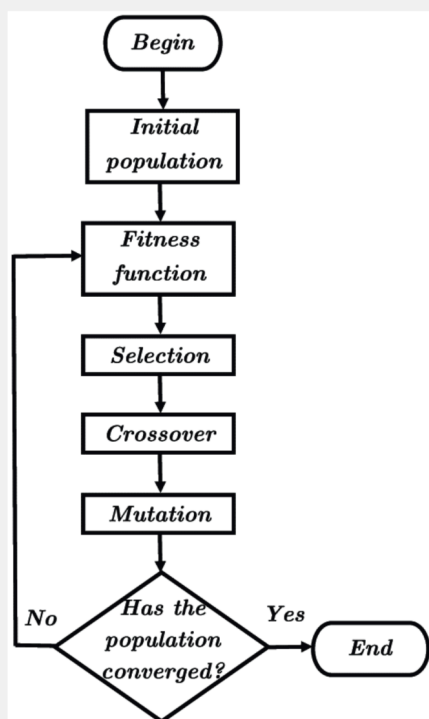
### Адаптивні рішення

Застосування генетичних алгоритмів для вирішення біматричних ігор з урахуванням впливу середовища дозволяє знаходити адаптивні та стійкі рішення в умовах невизначеності.

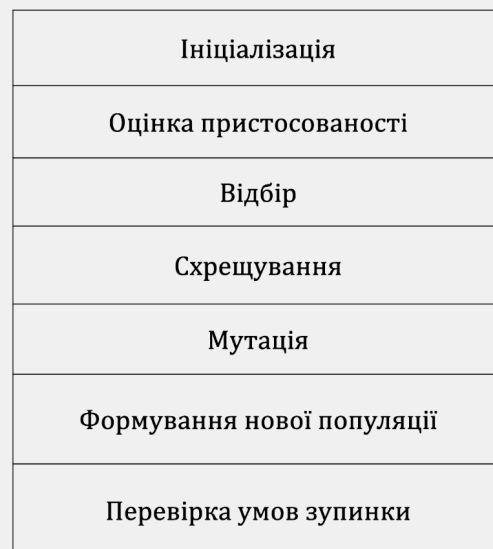
### Ефективні реалізації

Розробка ефективних програмних реалізацій генетичних алгоритмів для вирішення біматричних ігор є важливим прикладним завданням, що вимагає вибору відповідних структур даних, операторів та параметрів алгоритму.

10



## ГЕНЕТИЧНИЙ АЛГОРИТМ



11

## КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ

Основний модуль

Клас GeneticAlgorithmBimatrixGame

Клас GeneticAlgorithmZeroSumGame

Клас GeneticAlgorithmTwoPlayersAndNatureGame

12

### МАТРИЦЯ ВИГРАШІВ ГРАВЦЯ А ТА ГРАВЦЯ В

Стратегії компанії В		П1	П2	П3
Стратегії компанії А	П1	3, 2	2, 2	0, 4
	П2	1, 1	3, 4	4, 0
	П3	2, 0	1, 3	1, 1

- Стратегії компанії А: П1, П2, П3.
- Стратегії компанії В: П1, П2, П3.

13

### МАТРИЦЯ ВИГРАШІВ ГРАВЦІВ У ГРІ З УРАХУВАННЯМ ВПЛИВУ ПРИРОДИ

Стратегії «природи»		М1	М2	М3
Стратегії компанії А	П1	8	2	1
	П2	4	8	2
	П3	2	4	8

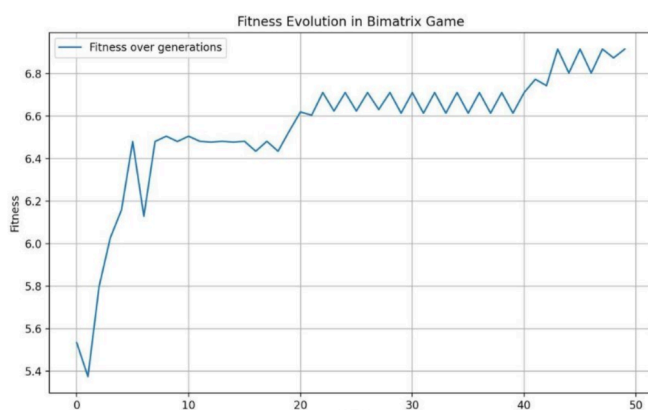
- Стратегії «природи»: М1, М2, М3.
- Стратегії компанії А: П1, П2, П3.

14

## ЗАВДАННЯ ПРАКТИЧНОЇ ЧАСТИНИ

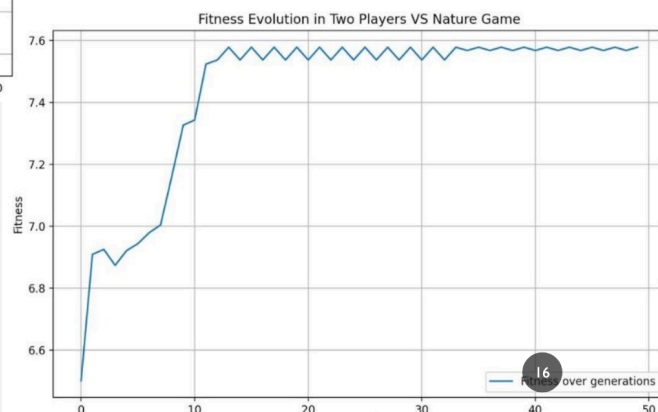
1. Провести розрахунок біматричної гри між гравцями, не враховуючи вплив природи.
2. Виконати розрахунок моделі гри, де спочатку гравці змагаються один з одним у біматричній грі, отримуючи результат  $R_{bi}$ , а потім кожен з них грає з природою, отримуючи результат  $R_p$ . Отримані результати ігор усереднюються за формулою:  $R = 0,6 \cdot R_{bi} + 0,4 \cdot R_p$
3. Третя модель передбачає одночасну гру трьох гравців, де одна гра є біматричною, а інші дві антагоністичними. У цій грі функція пристосованості в генетичному алгоритмі визначається за формулою:  $F = 0,6 \cdot F_{bi} + 0,4 \cdot F_p$
4. Провести аналіз результатів розрахунків моделей і зробити висновки щодо впливу природи на біматричну гру.

15



Результати зміни функції пристосованості у біматричній грі в залежності від кількості поколінь, без урахування впливу «природи»

Результати зміни функції пристосованості у біматричній грі в залежності від кількості поколінь, з урахуванням впливу «природи»



16

### РЕЗУЛЬТАТИ СТРАТЕГІЧНОГО ВИБОРУ ГРАВЦІВ А ТА В У РАМКАХ ДРУГОЇ МОДЕЛІ

Стратегія проти природи	0,47313266	0,38493724	0,22299255
Гравець А	0,006	0,993	0,001
Гравець В	0,083	0,905	0,012
Природа * 0,4	0,18925306	0,1539749	0,08919702
Гравець А * 0,6	0,0036	0,5958	0,0006
Гравець А з урахуванням природи	0,079301224	0,65738996	0,036278808
Природа * 0,04	0,18925306	0,1539749	0,08919702
Гравець В * 0,06	0,00498	0,0543	0,00072
Гравець В з урахуванням природи	0,06308	0,6878	0,007488

17

### РЕЗУЛЬТАТИ СТРАТЕГІЧНОГО ВИБОРУ ГРАВЦІВ А ТА В У РАМКАХ ТРЕТЬОЇ МОДЕЛІ

Стратегія проти природи	0,47313266	0,38493724	0,22299255
Гравець А	0,006	0,993	0,001
Гравець В	0,083	0,905	0,012

#### Гра трьох гравців

Природа	0,47313266	0,38493724	0,22299255
Гравець А	0,0762644	0,6285556	0,05518
Гравець В	0,6114644	0,0375556	0,11098

18

### ЗАГАЛЬНА ТАБЛИЦЯ РЕЗУЛЬТАТІВ

Номер моделі	Стратегії гравця А	Стратегії гравця В
№1	0,006; 0,993; 0,001	0,083; 0,905; 0,012
№2	0,0793; 0,6574; 0,0363	0,1255; 0,6046; 0,0429
№3	0,0762644; 0,6285556; 0,05518	0,6114644; 0,0375556; 0,11098

19

### ОСНОВНІ РЕЗУЛЬТАТИ

Аналіз результатів показав, що врахування природи може змінювати оптимальні стратегії гравців, роблячи їх менш передбачуваними. Це відкриває нові можливості для дослідження та моделювання стратегічних взаємодій у різних системах, де зовнішні фактори відіграють важливу роль. Отримані результати також вказують на те, що включення природи як третього гравця може бути корисним у різних галузях, таких як економіка, екологія та соціальні науки, де взаємодія між учасниками часто залежить від зовнішніх умов. Таким чином, дослідження підтверджує важливість врахування впливу зовнішніх факторів на стратегічні взаємодії та підкреслює необхідність подальшого вивчення цієї проблеми.

20

## НАУКОВА НОВИЗНА

Традиційно для розв'язання задач біматричних ігор розглядаються лише два гравці. Вплив природи на стратегії та результати гри є новим підходом, який дозволяє значно розширити розуміння динаміки таких ігор. У цій роботі було адаптовано генетичний алгоритм для розв'язання задачі з урахуванням трьох гравців, що включає природу як активного учасника.

21

## ВИСНОВКИ

- Здійснено аналіз впливу "природи" на стратегічне прийняття рішень у біматричних іграх та встановлено, що цей вплив може суттєво змінити стратегії гравців.
- Досліджено генетичні алгоритми, які є потужним інструментом для вирішення біматричних ігор в умовах невизначеності та динамічного середовища.
- Розроблено математичні моделі та методи, що враховують вплив навколишнього середовища.
- Отримано результати, які можуть бути застосовані в різних сферах, таких як економіка, політика, соціологія та військова справа.
- Зроблено висновки щодо ефективності різних моделей та методів аналізу біматричних ігор з урахуванням впливу "природи".

22

## ПОДАЛЬШІ ДОСЛІДЖЕННЯ



### Інтеграція з AI

Поєднання генетичних алгоритмів з іншими методами штучного інтелекту може підвищити ефективність вирішення біматричних ігор.



### Динамічні середовища

Дослідження поведінки генетичних алгоритмів в умовах швидкозмінного та невизначеного середовища є перспективним напрямком.



### Реальні застосування

Впровадження розроблених методів в реальні системи прийняття рішень в різних галузях є важливим практичним завданням.

23

## ДЯКУЮ ЗА УВАГУ!

24