

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська кваліфікаційна робота на тему:

**«ДОСЛІДЖЕННЯ РОЙОВОГО КОМПОЗИЦІЙНОГО АЛГОРИТМУ
ДЛЯ ЗАДАЧ ДИСКРЕТНОЇ ОПТИМІЗАЦІЇ»**

Виконав: студент IV курсу, групи ІСТ-216
Спеціальність 126 – Інформаційні системи та
технології

Лаврінчук О. О.

« 12 » 06 2025 р.

Керівник: к.т.н., доцент каф. АІТ

Іванов Ю. Ю.

Рецензент: д.т.н., професор каф КСУ

Ковтун В. В.

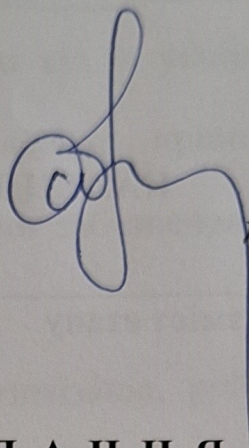
« 12 » 06 2025 р.

Допущено до захисту
зав. кафедри АІТ
д.т.н., проф. Бісікало О. В.

« 16 » 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – «Інформаційні технології»
Спеціальність 126 – «Інформаційні системи та технології»
Освітньо-професійна програма Інтелектуальні інформаційні системи



ЗАТВЕРДЖУЮ

завідувач кафедри АІТ
д.т.н., проф. Бісікало О. В.

« 16 » 06 2025 р.

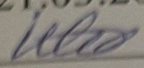
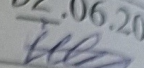
З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Лаврінчуку Олегу Олександровичу

1. Тема роботи: Дослідження ройового композиційного алгоритму для задач дискретної оптимізації.
Керівник роботи: к.т.н., доцент каф. АІТ Іванов Ю. Ю.
Затверджені наказом ВНТУ від «20» 03 2025 року № 97.
2. Строк подання роботи студентом: «до 23» 06 2025 р.
3. Вихідні дані до роботи: довжина перестановки; координати точок на площині; параметри алгоритму (кількість пошукових агентів, агентів-спостерігачів та розвідників; кількість ітерацій); кількість запусків для збору статистики; набір тестових задач.
4. Зміст текстової частини: вступ; аналіз предметної області роботи; розробка математичного апарату ройового алгоритму; розробка програмного забезпечення та експериментальні дослідження; висновки; список використаних джерел.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема роботи рою; приклад кривої збіжності алгоритму; схема програми.

6. Консультанти розділів роботи

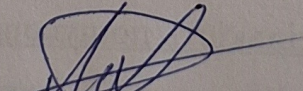
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Іванов Ю. Ю. к.т.н., доцент каф. АІТ	21.03.2025 	02.06.2025 

7. Дата видачі завдання: «21» 03 2025 р.

КАЛЕНДАРНИЙ ПЛАН

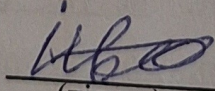
№ з/п	Назва та зміст етапу	Термін виконання		Примітки
		початок	закінчення	
1	Аналіз предметної області роботи	21.03.2025	04.04.2025	Виконано
2	Розробка математичного апарату ройового алгоритму	05.04.2025	26.04.2025	Виконано
3	Розробка програмного забезпечення та експериментальні дослідження	27.04.2025	16.05.2025	Виконано
4	Оформлення пояснювальної записки і графічного матеріалу	17.05.2025	27.05.2025	Виконано
5	Попередній захист роботи	28.05.2025	02.06.2025	Виконано
6	Остаточний захист роботи	16.06.2025	23.06.2025	Виконано

Студент


(підпис)

Лаврінчук О. О.
(прізвище та ініціали)

Керівник роботи


(підпис)

Іванов Ю. Ю.
(прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 71 сторінки формату А4, на яких є 20 рисунків, 9 таблиць, список використаних джерел містить 25 найменувань.

У роботі проаналізовано алгоритм ройового інтелекту, який використовує низку евристик залежно від їх успішності. Представлено його математичний апарат, а також наведено принципи роботи. Розроблено відповідне алгоритмічне та програмне забезпечення, яке дозволяє виконати експериментальні дослідження.

Ключові слова: дискретна оптимізація, ройовий алгоритм, евристика, перестановка, маршрут.

ABSTRACT

The bachelor's thesis consists of 71 A4 pages with 20 figures, 9 tables, the list of sources used contains 25 items.

In this work was analyzed a swarm intelligence algorithm that uses a several of heuristics depending on their success. Its mathematical apparatus was presented, as well as the working principles. The algorithm and software, allows us to perform some experimental researches was developed.

Keywords: discrete optimization, swarm algorithm, heuristics, permutation, route.

ЗМІСТ

ВСТУП.....	3
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ.....	6
1.1 Постановка тестових задач.....	6
1.2 Огляд класичних методів розв’язання.....	17
1.3 Огляд ройових методів розв’язання.....	24
1.4 Висновки.....	28
2 РОЗРОБКА МАТЕМАТИЧНОГО АПАРАТУ	29
РОЙОВОГО АЛГОРИТМУ	29
2.1 Загальні принципи використання ройових алгоритмів	29
2.2 Узагальнена модель композиційного алгоритму.....	33
2.3 Висновки.....	43
3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	
ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ.....	44
3.1 Обґрунтування вибору мови програмування.....	44
3.2 Опис роботи програми	46
3.3 Результати експериментів.....	47
3.4 Висновки.....	53
ВИСНОВКИ	54
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	55
ДОДАТКИ.....	59
Додаток А (обов’язковий). Технічне завдання	60
Додаток Б (обов’язковий). Ілюстративна частина.....	63
Додаток В (обов’язковий). Лістинг програмного забезпечення	66
Додаток Д (обов’язковий). Протокол перевірки БКР	70

ВСТУП

Актуальність. Нині розвиток людства відзначається широким впровадженням автоматизації, яка охоплює всі сфери життя: медицину, побут, культуру, фінанси, природокористування тощо. Розробляються нові підходи до автоматизації, які включають інтелектуальні методи управління, такі як експертні системи, нейронні мережі, еволюційні алгоритми, нечітка логіка та інші. В умовах зростання обсягу даних особливо важливими стають задачі комбінаторної оптимізації, які набувають все більшого значення у подібних інтелектуальних системах та у ході аналізу даних. Методи оптимізації застосовуються для розв'язання різноманітних задач, таких як оптимальне управління, максимізація прибутку підприємств, мінімізація похибок у вимірюваннях, пошук найкоротших шляхів, формування геномних послідовностей тощо [1].

У сучасному світі багато теорій аналізуються через призму штучного інтелекту. У одній із його гілок – обчислювальному інтелекті, виділяється роевий інтелект, який базується на принципах, запозичених із поведінки соціальних організмів. Такий підхід дозволяє використовувати моделювання колективної поведінки для розв'язання різноманітних оптимізаційних задач, коли їх неможливо вирішити точними математичними методами через недостатність або невизначеність вхідних і вихідних даних, обмеження, високу розмірність, динамічні зовнішні умови тощо [2, 3].

Останнім часом спостерігається збільшення кількості наукових робіт різних авторів, які застосовують ройовий інтелект для розв'язання задач дискретної оптимізації. Наприклад, можна виділити наукові роботи

Д. Мартінез-Муньос [4], М. Гундуз [5], Дж. Ксю [6], С.Д. Погорілий [7] та інших. Вони представляють композиційний алгоритм оптимізації, заснований на поєднанні низки евристик, як ефективний та зручний алгоритм, який можна використовувати для розв'язання заданих комбінаторних задач. Тому актуальною є науково-практична задача застосування даного алгоритму із різним набором евристик та дослідження ефективності його роботи на низці тестових задач.

Мета і задачі дослідження. Метою роботи є підвищення ефективності роботи композиційного ройового алгоритму для розв'язання задач дискретної оптимізації за рахунок застосування стратегії вибору складових евристик.

Для досягнення мети необхідно розв'язати наступні *задачі*:

- розглянути постановку тестових задач оптимізації;
- проаналізувати алгоритми розв'язання задач дискретної оптимізації;
- розробити математичну модель композиційного ройового алгоритму оптимізації;
- розробити програмні засоби та оцінити ефективність їх роботи.

Об'єктом дослідження є оптимізація функцій для дискретних задач.

Предметом дослідження є моделі, методи та інструментальні засоби для пошуку оптимального значення цільової функції для дискретних задач у заданих умовах.

Методи дослідження. У роботі використано поняття теорії оптимізації та ройового інтелекту під час дослідження роботи композиційного ройового інтелекту у ході розв'язання тестових задач. Для аналізу та перевірки достовірності отриманих теоретичних положень застосовано експериментальне дослідження. Обробка даних виконувалася за допомогою мови програмування Python.

Науково-технічний результат роботи полягає у тому, що на основі теоретичних положень розроблено математичне, алгоритмічне та програмне забезпечення, яке дозволяє застосовувати композиційний ройовий алгоритм оптимізації для розв'язання низки дискретних задач.

У роботі отримані результати, які мають *практичну цінність*, а саме: наведено експериментальні дослідження ефективності застосування представленого алгоритму на різних тестових задачах.

Апробація результатів та публікації. За результатами дослідження опубліковано тези доповіді [8] на Всеукраїнській науково-технічній конференції підрозділів ВНТУ (м. Вінниця, 2025).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОБОТИ

1.1 Постановка тестових задач

Комбінаторна оптимізація — це розділ прикладної математики, що займається пошуком оптимального розв'язку серед скінченного (але часто великого) набору можливих варіантів. Такі задачі формулюються як екстремальні задачі — мінімізація або максимізація певної функції при заданих обмеженнях. Інтерес до них збільшився в XIX та XX століттях у зв'язку з розвитком теорії графів, програмування, криптографії та теорії ймовірності. Зараз комбінаторні методи застосовуються як і у самій математиці, так і поза нею — теорія кодування, планування експерименту, топологія, кінцева алгебра, математична логіка, теорія ігор, кристалографія, біологія, статистична фізика, економіка тощо [9].

Для деяких таких задач виявили дивовижну властивість. Деякі з них універсальні в тому сенсі, що побудова поліноміального алгоритму для будь-якої такої задачі спричиняє можливість побудови такого ж алгоритму для решти задач. У таких задачах зі зростанням розмірності кількість варіантів перебору зростає екстремально швидко. Тому вони цікавлять дослідників і фахівців в області комбінаторної оптимізації, оскільки допускається застосування різноманітних алгоритмів [1, 2].

Можна виділити такі основні задачі:

- про комівояжера;
- про рюкзак;

- квадратична задача про призначення;
- про розфарбування графа;
- про пошук оптимальної перестановки;
- про покриття або про повне (точне) покриття.

Задачу комівозера можна сформулювати таким чином: знайти гамільтоновий цикл найменшої довжини на графі $G(V, E)$, вершини V якого відповідають вузловим точкам, а дуги E відображають комунікації між ними. Але, оскільки на графі існує багато гамільтонових циклів, то проявляється комбінаторна природа задачі. Математично представимо задачу, використавши цілочисельне лінійне програмування [10, 11]:

$$f_{\min}(X) = \sum_i^N \sum_j^N c(v_i, v_j) \cdot a_{ij} \rightarrow \min, \quad (1.1)$$

$$\left\{ \begin{array}{l} \sum_{i=1}^N a_{ij} = 1, j = \overline{1, N}; i \neq j - \text{обмеження одноразового в'їзду до міста;} \\ \sum_{j=1}^N a_{ij} = 1, i = \overline{1, N}; i \neq j - \text{обмеження одноразового виїзду з міста;} \\ u_i - u_j + N \cdot a_{ij} \leq N - 1, (i, j) = \overline{2, N}; i \neq j - \text{усунення існування підциклів;} \\ u_i, u_j \in N^*; \\ a_{ij} \in \{0; 1\}, (i, j) = \overline{1, N}, \end{array} \right. \quad (1.2)$$

де $X = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), \dots, (v_N, v_1)\}$ – перестановка цілих чисел (міст);

N – кількість вершин графа;

$a_{ij} \in \{0; 1\}$, $(i, j) = \overline{1, N}$ – булева змінна, яка означає включення (1) або виключення (0) ребра (v_i, v_j) із загального маршруту;

u – значення порядкових номерів міст (додатні цілі числа із множини N^*);

$q \sim (N - 1)!$ – кількість гамільтонових циклів на графі;

$c(v_i, v_j)$ – елементи матриці суміжності C графа $G(V, E)$, які визначають довжину відповідного ребра:

$$C = \begin{pmatrix} \infty & c(v_1, v_2) & \dots & c(v_1, v_N) \\ c(v_2, v_1) & \infty & \dots & c(v_2, v_N) \\ \dots & \dots & \dots & \dots \\ c(v_N, v_1) & c(v_N, v_2) & \dots & \infty \end{pmatrix}; \quad (1.3)$$

Тобто задано певну скінченну множину перестановок n елементів (пунктів) у формі:

$$S_N = \{s_i\}, \quad (1.4)$$

де s_i – одна перестановка.

Цільова функція $f(s_i)$ представляє собою довжину замкнутої ламаної, такої, що проходить через n заданих точок в порядку, який задає перестановка $s_i \in S_n$. Необхідно знайти найменшу або оптимальну довжину цієї ламаної [2].

Враховуючи те, що задача замкнута, можна перевизначити цільову функцію, використовуючи умову створення циклу на графі в евклідовому просторі, тобто [1]:

$$f(X) = \sum_{i=1}^{N-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}, \quad (1.5)$$

де x та y – координати міст на площині.

Можна виділити такі види задачі комівояжера:

1. Симетрична задача – відстані задані між будь-якими двома містами однакові (матриця відстаней симетрична). Симетричну задачу називають метричною, якщо відносно довжин шляхів виконується нерівність трикутника, тобто обхідні шляхи довші за прямі (ребро від вершини i до вершини j ніколи не довше за шлях через проміжну вершину k), що можна представити правилом:

$$c_{ij} \leq c_{ik} + c_{kj}. \quad (1.6)$$

2. Асиметрична задача допускає несиметричність матриці відстаней. У ще більш загальному випадку шляхи між певними містами відсутні.

3. Задача з частковим упорядкуванням вимагає, щоб вузловий пункт i був відвіданий до пункту j (таких умов може бути декілька).

Отже, задача комівояжера пов'язана з комбінаторним аналізом, теорією графів та лінійним програмуванням.

Гамільтоновий цикл означає, що необхідно пройти всі вершини графа та повернутися у початкову точку. На графі може бути дуже багато гамільтонових циклів [11]:

$$q = \begin{cases} \frac{(n-1)!}{2} & \text{для симетричної задачі комівояжера;} \\ (n-1)! & \text{для асиметричної задачі комівояжера,} \end{cases} \quad (1.7)$$

а потрібно вибрати мінімальний з них. Дана задача є NP -складною, тобто дуже складною в обчислювальному плані задачею, для якої не існує ефективного методу розв'язання, з ростом розмірності її неможливо розв'язати точно за

прийнятний час. Але, якщо є певне рішення, то його правильність для цієї задачі можна легко перевірити.

У реальних задачах цілочисельних значень набувають не всі, а одна або декілька змінних. Такі задачі називають частково цілочисельними. Сюди можна віднести задачу про призначення, про найкоротший маршрут у графі, про комівояжера тощо [9].

Наприклад, маємо 12 доріг та 6 вершин, в яких має побувати робот. З першої точки можна потрапити до кожної з п'яти інших

$$a_{12}, a_{13}, a_{14}, a_{15}, a_{16}.$$

Друга точка пов'язана з трьома, а отже, маємо такі три змінні:

$$a_{21}, a_{24}, a_{25}.$$

Аналогічно позначаємо змінні, що відповідають можливим маршрутам руху з третьої, четвертої, п'ятої та шостої точок:

$$a_{31}, a_{34}, a_{36},$$

$$a_{41}, a_{42}, a_{43}, a_{45}, a_{46},$$

$$a_{51}, a_{52}, a_{54}, a_{56},$$

$$a_{61}, a_{63}, a_{64}, a_{65}.$$

Отримали 24 булеві змінні ($a_{ij} = \{0; 1\}$). Однак деякі змінні, наприклад, та a_{12} та a_{21} описують один маршрут, довжина якого за умовою задачі не

змінюється залежно від напрямку пересування. Матрицю найкоротших відстаней C між всіма вершинами можна представити таким чином:

$$C = \begin{pmatrix} \infty & 50 & 75 & 40 & 70 & 100 \\ 50 & \infty & 80 & 30 & 40 & 90 \\ 75 & 80 & \infty & 50 & 110 & 70 \\ 40 & 30 & 50 & \infty & 60 & 80 \\ 70 & 40 & 110 & 60 & \infty & 50 \\ 100 & 90 & 70 & 80 & 50 & \infty \end{pmatrix}.$$

Тоді запишемо критерій оптимальності, враховуючи дані умови:

$$\begin{aligned} \min f(\{a_{ij}\}) = & 50a_{12} + 75a_{13} + 40a_{14} + 70a_{15} + 100a_{16} + 30a_{24} + \\ & + 40a_{25} + 50a_{34} + 70a_{36} + 60a_{45} + 80a_{46} + 50a_{56}. \end{aligned}$$

Обмеження щодо одноразового в'їзду в кожний пункт:

$$\begin{cases} a_{21} + a_{31} + a_{41} + a_{51} + a_{61} = 1; \\ a_{12} + a_{42} + a_{52} = 1; \\ a_{13} + a_{43} + a_{63} = 1; \\ a_{14} + a_{24} + a_{34} + a_{54} + a_{64} = 1; \\ a_{15} + a_{25} + a_{45} + a_{65} = 1; \\ a_{16} + a_{36} + a_{46} + a_{56} = 1. \end{cases}$$

Обмеження щодо одноразового виїзду з кожного пункту:

$$\begin{cases} a_{12} + a_{13} + a_{14} + a_{15} + a_{16} = 1; \\ a_{21} + a_{24} + a_{25} = 1; \\ a_{31} + a_{34} + a_{36} = 1; \\ a_{41} + a_{42} + a_{43} + a_{45} + a_{46} = 1; \\ a_{51} + a_{52} + a_{54} + a_{56} = 1; \\ a_{61} + a_{63} + a_{64} + a_{65} = 1. \end{cases}$$

У результаті, розв'язавши задачу, можна отримати оптимальний варіант маршруту. Тобто за оптимальним планом необхідно пройти маршрут $[1-2-5-6-3-4-1]$, довжина якого мінімальна і дорівнює 300 одиниць.

Слід зазначити, що невідомо жодного простого критерію або ефективного алгебраїчного методу, що дозволяє відповісти на запитання, чи існує в довільному графі G гамільтоновий цикл [12].

Критерії існування гамільтонових графів, які подано в роботах Пошана, Неша-Вільямса, Оре і Дірака, є занадто загальними і не придатні для довільних графів, які зустрічаються на практиці. Можна привести відомі достатні умови (сильну – із теореми Оре, та більш слабку – із теореми Дірака) [12].

Теорема Оре: нехай G – кінцевий та простий (з кінцевою кількістю вершин та ребер та без кратних ребер) граф, який має $N > 2$ вершин. Позначимо через $\deg$ степінь певної вершини в G , тобто число інцидентних цій вершині ребер. Тоді, якщо:

$$\deg(v) + \deg(w) \geq N \quad (1.8)$$

для будь-якої пари несуміжних (з'єднаних одним і тим же ребром) вершин v і w графа G , то граф є гамільтоновим.

Теорема Дірака. Нехай G – кінцевий та простий граф, який має $N > 2$ вершин. Тоді, якщо:

$$\deg(v) \geq N/2 \quad (1.9)$$

для будь-якої вершини v графа G , то граф є гамільтоновим.

Узагальненням цих двох теорем є теорема Бонді-Хватала: граф G є гамільтоновим тоді і лише тоді, коли його замикання – гамільтоновий граф.

Для графа G з N вершинами замикання будується додаванням до G ребра (v, w) для кожної пари несуміжних вершин v і w , сума ступенів яких не менша N .

Взагалі існує ряд методів для знаходження гамільтонового графа [12]:

- алгебраїчний метод заснований на роботах Йоу-Даніельсона-Дхавана включає в себе побудову шляхів за допомогою простого множення матриць;

- мультимаршрутний метод Селбі: крім побудови шляхів, що допомагають швидше знайти гамільтоновий цикл або вказують на його відсутність, значна увага приділяється частині графа, яка залишилася поза цими шляхами. За допомогою певної процедури деякі вершини можуть бути видалені, що дозволяє суттєво зменшити кількість кроків пошуку циклу;

- метод перебору Робертса-Флореса: на противагу алгебраїчним методам, за допомогою яких намагаються відразу знайти всі гамільтонові цикли, і при реалізації яких доводиться зберігати всі шляхи, які можуть виявитися частинами таких циклів, метод перебору має справу з одним шляхом, що безперервно подовжується аж до моменту, коли отримано гамільтоновий цикл, або стає ясно, що цей шлях не може привести до

гамільтонових циклів. Тоді маршрут модифікується деяким систематичним способом (який гарантує, що врешті-решт будуть вичерпані всі можливості), після чого продовжується пошук гамільтонового циклу.

Алгебраїчні методи знаходження гамільтонових циклів не можуть бути застосовані до задач з більш ніж кількома десятками вершин, оскільки вони потребують занадто великого часу роботи. Більш прийнятним є підхід Робертса-Флореса, який не має надмірних вимог до пам'яті комп'ютера, але час роботи якого експоненціально залежить від кількості вершин графа.

Ще однією важливою задачею теорії алгоритмів є квадратична задача про призначення. Вона представляє собою задачу комбінаторної оптимізації, у якій необхідно знайти оптимальний розподіл n об'єктів (наприклад, машини, роботи) на n локаціях (фабрика, склад) так, щоб мінімізувати / максимізувати певну цільову функцію, яка зазвичай пов'язана з витратами на взаємодію об'єктів, розміщених у різних локаціях. Дана задача широко застосовується у різних галузях науки та техніки: розміщення машин на фабриці для мінімізації витрат на переміщення матеріалів між ними; мінімізація витрат на зв'язки між вузлами мережі; призначення робітників на завдання у проєктах з урахуванням витрат на навчання та час виконання роботи тощо.

Сформульована задача у 1957 році, а у 1976 році доведено те, що вона належить до класу NP-складних задач. Для тестування алгоритмів розроблено підрозділ бібліотеки тестових задач із дослідження операцій OR_library, максимальна розмірність яких складає 150 [13].

Необхідно знайти таку перестановку об'єктів π , що мінімізує загальну вартість розміщення, яка визначається як сума добутків інтенсивності потоку (взаємодії) між об'єктами та відстані між відповідними локаціями.

Цільова функція, яка описує вартість розміщення об'єктів у локаціях, обчислюється як [14]:

$$f_{\min} = \sum_{i=1}^N \sum_{j=1}^N f_{ij} \cdot c_{\pi(i)\pi(j)} \rightarrow \min, \quad (1.10)$$

де $\pi(i)$, $\pi(j)$ – позиції об'єктів i та j у перестановці;

f_{ij} – інтенсивність потоку між об'єктами i та j ;

$c_{\pi(i)\pi(j)}$ – відстань між локаціями $\pi(i)$ та $\pi(j)$.

Наприклад, якщо є три робітників: O_1 – бухгалтер, O_2 – менеджер, O_3 – програміст та три кабінети L_1 – поряд із ліфтом, L_2 – біля вікна, L_3 – у глибині офісу. Вони взаємодіють за таблицею 1.1 (матриця потоків F), тобто бухгалтер і менеджер часто ходять один до одного (5 разів на день), бухгалтер і програміст – 2 рази, а менеджер і програміст – 3 рази. Відстані між кабінетами наведені у таблиці 1.2. Розрахунки для перестановки $\pi = [c_2, c_1, c_3]$ можна провести наступним чином ($O_1 \rightarrow L_2, O_2 \rightarrow L_1, O_3 \rightarrow L_3$):

$$f_{12} = 5, c_{L_2L_1} = 2 \Rightarrow 5 \times 2 = 10,$$

$$f_{13} = 2, c_{L_2L_3} = 1 \Rightarrow 2 \times 1 = 2,$$

$$f_{21} = 5, c_{L_1L_2} = 2 \Rightarrow 5 \times 2 = 10,$$

$$f_{23} = 3, c_{L_1L_3} = 3 \Rightarrow 3 \times 3 = 9,$$

$$f_{31} = 2, c_{L_3L_2} = 1 \Rightarrow 2 \times 1 = 2,$$

$$f_{32} = 3, c_{L_3L_1} = 3 \Rightarrow 3 \times 3 = 9.$$

Таблиця 1.1 – Таблиця потоків F

#	O_1	O_2	O_3
O_1	–	5	2
O_2	5	–	3
O_3	2	3	–

Таблиця 1.2 – Таблиця відстаней C

#	L_1	L_2	L_3
L_1	–	2	3
L_2	2	–	1
L_3	3	1	–

Тоді отримаємо значення функції:

$$f = 10 + 2 + 10 + 9 + 2 + 9 = 42.$$

Відомі методи розв’язання даних задач поділяють на точні та евристичні (наближені). Перші з них гарантовано знаходять оптимальний шлях, потребуючи значних часових витрат, тому на практиці їх застосовують до задач відносно малих розмірностей.

Евристичні методи можуть знайти субоптимальний розв’язок, причому за короткий час. Їх можна поділити на три групи: конструктивні (алгоритм найближчого сусіда; жадібний алгоритм; алгоритм вставки) та стохастичні алгоритми (комбінують методи пошуку локальних та глобальних розв’язків у абстрактні стратегії оптимізації), а також алгоритми локальної оптимізації

(декомпозиція та зшивання карти, геометричні евристики), що і розглянемо далі.

1.2 Огляд класичних методів розв'язання

Точні методи забезпечують знаходження оптимального рішення, але їх застосування потребує значних часових витрат, тому їх зазвичай використовують для задач малої розмірності. Евристичні методи можуть знайти субоптимальний розв'язок за коротший час. Їх можна поділити на три основні групи: конструктивні та стохастичні алгоритми (які поєднують методи локального та глобального пошуку рішень), а також алгоритми локальної оптимізації [8, 12].

Метод повного перебору дає гарантію знаходження розв'язку в будь-якому випадку, проте розмірність задачі часто буває настільки великою, що вирішення її простим перебором неможливе. Наприклад, кількість гамільтонових циклів у графі може бути величезною, і потрібно знайти найкоротший із них. Це призводить до проблеми “прокляття розмірності”, або правильніше сказати — “комбінаторного вибуху”, коли часова складність алгоритму різко зростає зі збільшенням кількості вхідних даних задачі [3].

Загальна схема розв'язання задачі (рисунок 1.10):

1. Згенерувати всі можливі перестановки.
2. Для кожної перестановки розрахувати значення цільової функції.
3. Вибрати найкраще із них.

Приклад повного перебору перестановок від 1 до 4 наведено на рисунку 1.1.

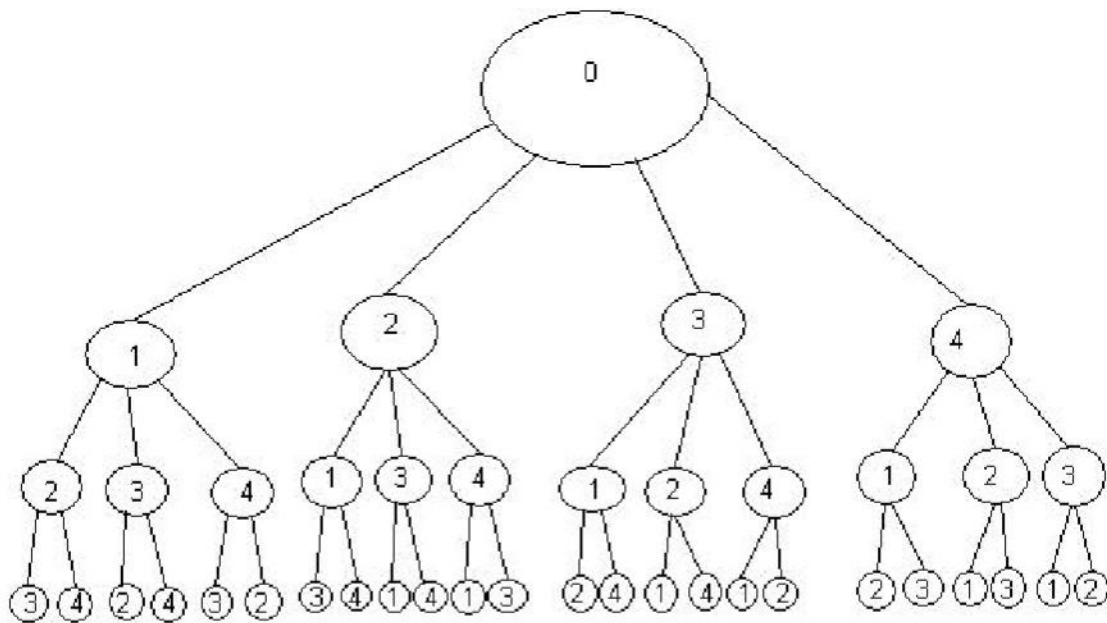


Рисунок 1.1 – Прилад повного перебору

Евристика — це метод, який знаходить деякий допустимий розв’язок $\tilde{X} \in R^n$, який може не бути оптимальним (а тільки квазіоптимальним). Звісно, бажано, щоб розв’язок $\tilde{X}$ співпадав з оптимальним результатом X^* . Однак для більшості евристичних методів можна лише сподіватися, а для деяких довести асимптотичну збіжність, що вони наближаються до оптимуму [3].

Залежно від того, чи евристика обчислює новий маршрут або намагається покращити вже існуючий, евристичні методи поділяються на конструктивні та постоптимізаційні. Найбільш поширеними є any-time алгоритми, які поступово покращують поточне наближене рішення.

Метод гілок і меж — це універсальний алгоритмічний підхід, який активно застосовується для вирішення задач дискретної оптимізації. Він є вдосконаленим методом повного перебору, при якому поетапно відсіюються рішення, які виглядають не вигідними. Суть методу полягає в розбитті множини можливих рішень (стратегія “розділяй і володарюй”) через розгалуження та оцінку (межі) кожної множини рішень. Оскільки деякі

рішення не аналізуються, метод гілок і меж не завжди гарантує точне розв'язання задачі. Відкинуте на початковому етапі “невигідне” рішення в кінцевому підсумку може виявитися найкращим. Як правило, алгоритм потрібно повторити для $n-2$ рівнів дерева, де n — це кількість стартових вершин графа [12]. Приклад наведено на рисунку 1.2.

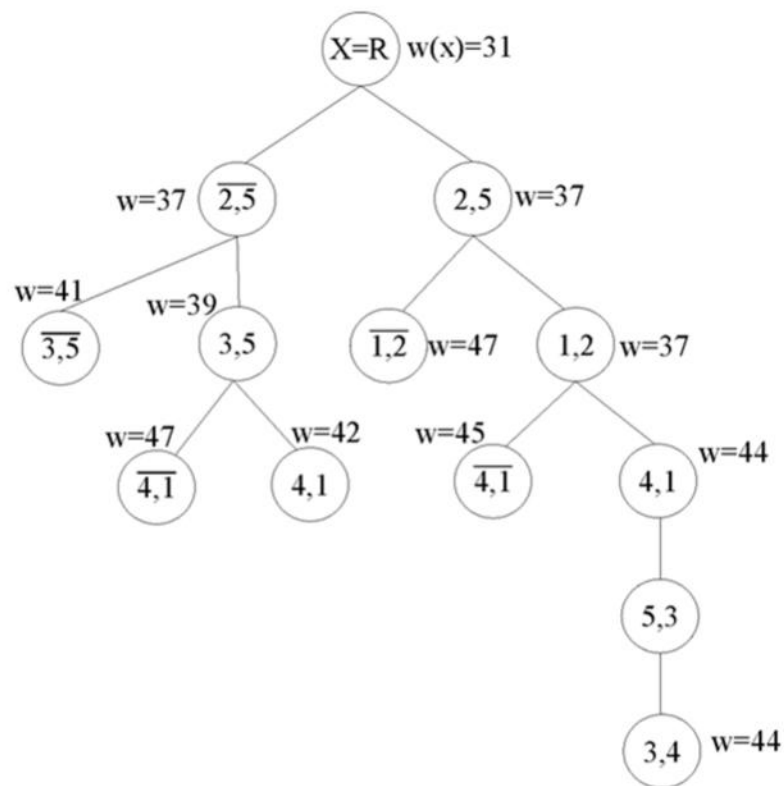


Рисунок 1.2 – Приклад роботи методу гілок та меж

Задовільних теоретичних оцінок швидкодії алгоритму немає. Практика показує, що на сучасних ЕОМ він дозволяє знайти оптимальний розв'язок, наприклад, для графів з кількістю вершин $n \leq 100$. Слід зазначити, що маючи достатньо часу, можна знайти оптимальний результат, оскільки зі збільшенням витрат часу алгоритм вироджується в повний перебір.

Ще одним із таких алгоритмів, який дозволяє теоретично отримати всі можливі покриття, є метод Петрика. Він представляє варіацію методу гілок та

меж, використовується в алгебрі логіки для кон'юнктивно-диз'юнктивного представлення імплікантної таблиці. Його складно застосувати для задач великої розмірності, але легко реалізувати програмно.

Жадібний алгоритм – це простий і швидкий алгоритм, який на кожному кроці вибирає найкращий локальний варіант (той, що здається найвигіднішим у даний момент), сподіваючись, що це приведе до глобально оптимального рішення (рисунок 1.3). Його можна описати такою послідовністю кроків [15]:

Крок 1. Вибрати найкращий елемент відповідно до жадібної стратегії вибору, якщо дана дія не порушує заданих обмежень.

Крок 2. Додати його до поточного розв'язку.

Крок 3. Повторити крок 1, оновивши задачу.

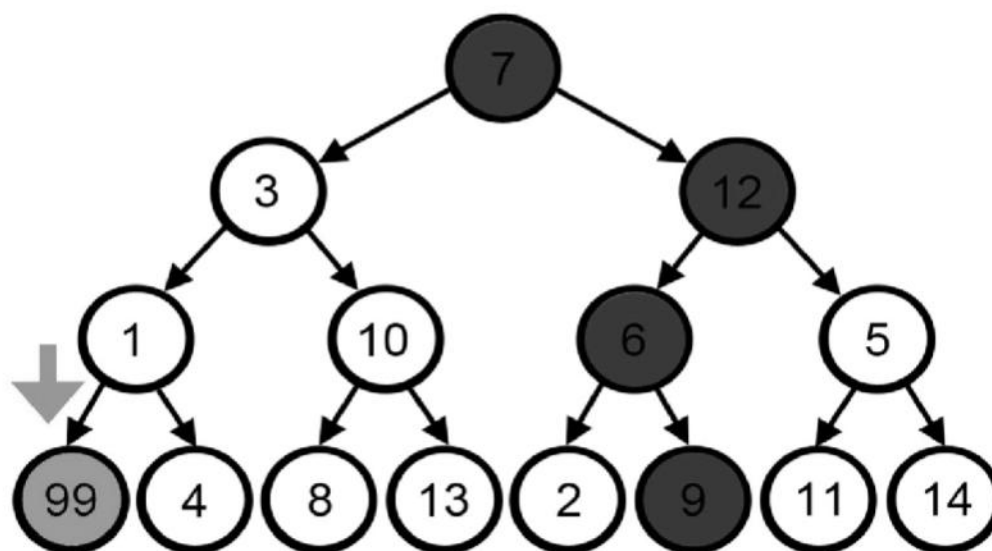


Рисунок 1.3 – Приклад роботи жадібного алгоритму

Для задачі комівояжера можна застосувати, наприклад варіацію жадібного алгоритму – алгоритм найближчого сусіда, який формулюється таким чином [15]:

Крок 1. Вибрати випадкову точку маршруту.

Крок 2. Додати до маршруту наступну точку, яка буде найближчою до попередньої.

Алгоритм простий у реалізації і працює швидко з складністю $O(n^2)$, однак, як і всі “жадібні” алгоритми, він може не завжди приводити до оптимального рішення. Для будь-якої кількості пунктів $n > 3$ можна знайти таке їх розташування, що цей алгоритм дасть найгірше можливе рішення, враховуючи відстані між точками та вибір початкової точки.

Методи лінійного програмування, які базуються на симплексному методі та методі Куна-Манкреса (угорський алгоритм) для задачі про призначення, використовують математичний апарат лінійного програмування. Спочатку математик Г. Кун дав йому ім'я “угорський метод” у зв'язку з тим, що алгоритм значною мірою заснований на ранніх роботах двох угорських математиків (Кеніга та Егерварі). Англієць Д. Манкрес у 1957 році зауважив, що алгоритм є поліноміальним. Часова складність оригінального алгоритму була $O(n^4)$, проте Дж. Едмондс і Р. Карп показали, що його можна модифікувати так, щоб досягти часу виконання $O(n^3)$. Л.Р. Форд та Д.Р. Фалкерсон поширили метод на загальні транспортні задачі [16].

Ефективними є методи локальної оптимізації. Наприклад, для задач, зведених до задачі комівояжера слід у побудованому маршруті вибрати декілька близьких один до одного вузлів і змінити конфігурацію ребер між ними, поки не отримано найменшої довжини. Узагальнення цього простого принципу є основою для алгоритму Ліна-Кернігана, реалізованого в 1965 році. Кількість з'єднань між вузлами, які приймають участь в перестановці, визначають складність методу локальної оптимізації (коефіцієнт складності k). Правила перетворення наведені на рисунках 1.4 та 1.5 [17, 18].

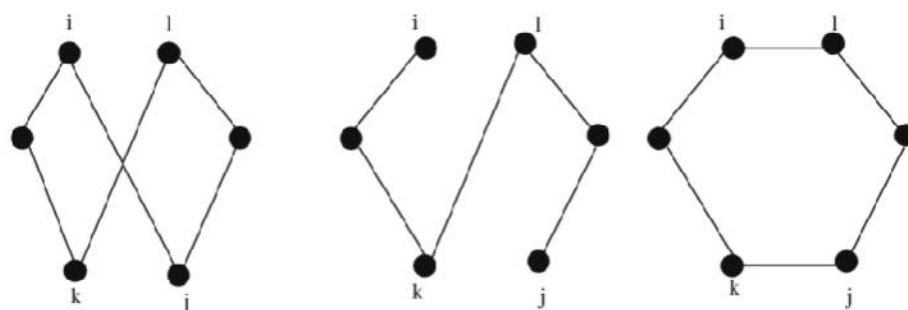


Рисунок 1.4 – Заміна ребер 2-орт евристикою

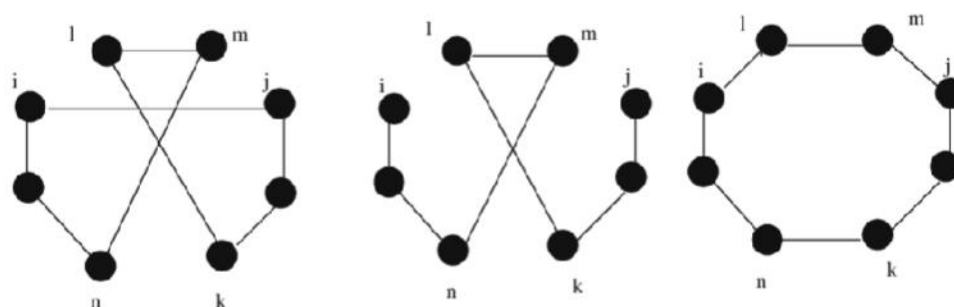


Рисунок 1.5 – Заміна ребер 3-орт евристикою

Метаевристичні алгоритми включають сучасні стратегії пошуку, натхнені живою та неживою природою. Серед них зазвичай виділяють алгоритм імітації відпалу, пошук із заборонами, імовірнісні жадібні алгоритми, генетичні алгоритми, алгоритми рою часток, імітацію поведінки мурашиних колоній, вовків, котів, кажанів, бактеріальних роїв, рою бджіл, імунних систем, нейромережі тощо [3, 19].

Основна ідея цих алгоритмів полягає в припущенні, що цільова функція $f(X)$ має багато локальних екстремумів, і що аналіз усіх можливих рішень є неможливим, незважаючи на їх кінцеву кількість. У таких випадках необхідно зосередити пошук на найбільш перспективних ділянках області допустимих рішень. Таким чином, задача зводиться до виявлення таких частин і швидкого їх перегляду.

Наприклад, алгоритм імітації відпалу, заснований на процесі кристалізації металів. Ідея в тому, щоб привести метал (цільова функція), нагрітий до високої температури (критерій зупинки), в стан з найменшою енергією (мінімальний гамільтоновий цикл), використовуючи контрольоване охолодження. Алгоритм використовує функції стану (всі можливі маршрути), енергії (найбільш короткий маршрут) і температури [3].

Механізм роботи генетичного алгоритму полягає в пошуку рішення шляхом комбінування параметрів у ході комп'ютерної біологічної еволюції. Відбувається своєрідний природний відбір рішень. Відмінною здатністю алгоритму є використання ряду операторів схрещування і мутації, які проводять рекомбінацію рішень-кандидатів (рисунок 1.6) [20].

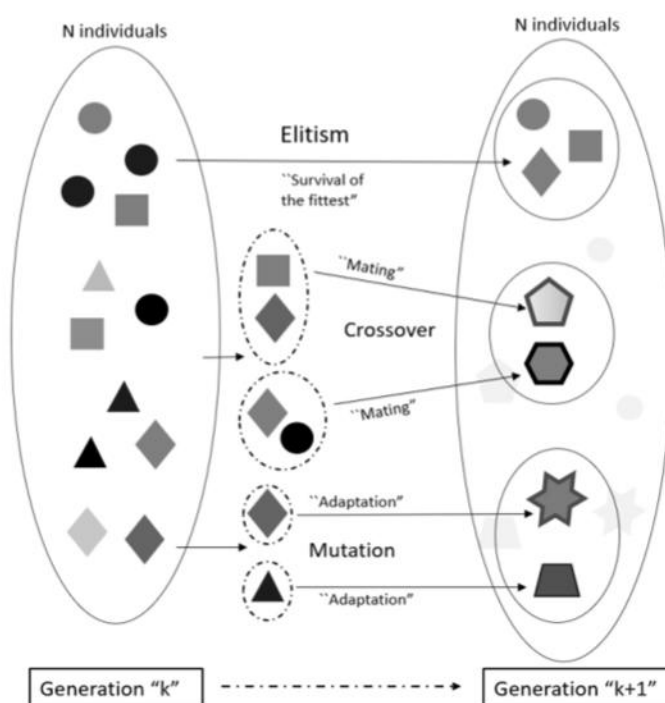


Рисунок 1.6 – Приклад роботи генетичного алгоритму

1.3 Огляд ройових методів розв'язання

Системи з використанням колективу інтелектуальних агентів, як правило, складаються з множини агентів, які локально взаємодіють між собою і з навколишнім середовищем. Самі агенти поодиночі, зазвичай, є достатньо простими, але всі разом, локально взаємодіючи, вони створюють так званий ройовий інтелект, який самостійно впродовж певного часу може самонавчатися та самоорганізовуватися (рисунок 1.7) [3, 20].

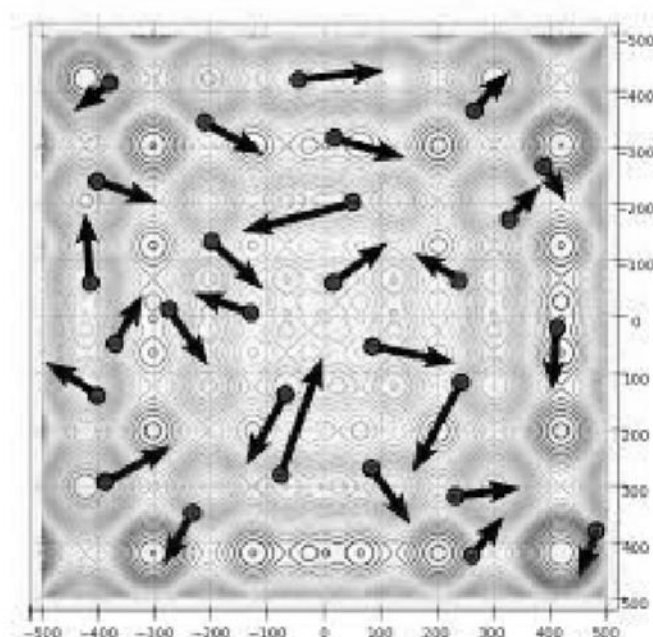


Рисунок 1.7 – Приклад роботи ройового алгоритму

Суть роботи алгоритму наслідування поведінки мурашиної колонії полягає у використанні моделі поведінки реальних мурашок, які шукають дорогу від колонії до джерела живлення, відкладаючи феромон. Мурашка, який виявив помічений феромоном шлях, з певною ймовірністю піде ним, зміцнивши своїм власним феромоном. Таким чином, ймовірність того, що в майбутньому інші мурашки будуть рухатися даним шляхом, росте з кількістю

мурашок, які раніше використали цей шлях (рисунок 1.8). Це приводить до виникнення найкоротших шляхів, оскільки феромон прагне акумулюватися швидше [3, 16].

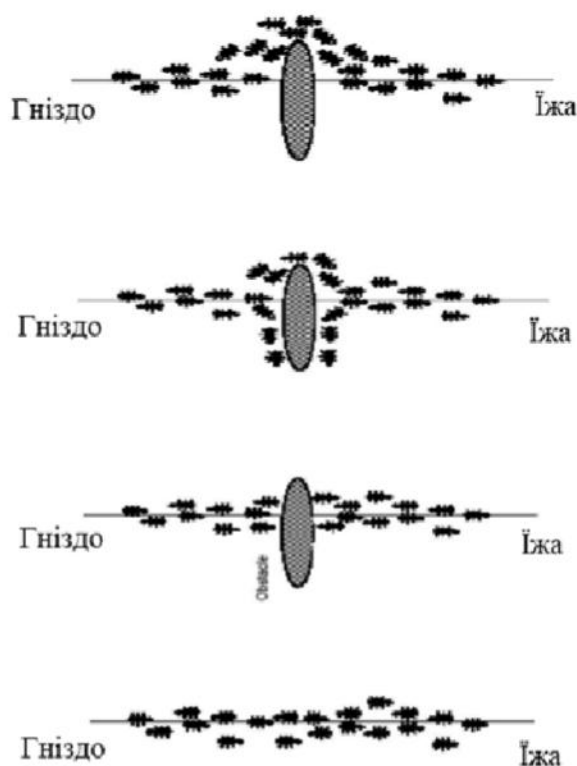


Рисунок 1.8 – Принцип роботи мурашиного алгоритму

Ще одним подібним алгоритмом є алгоритм зграї сірих вовків, розроблений у 2013 році С. Мірджалілі. Алгоритм імітує полювання та ієрархічний розподіл зграї вовків. Для того, щоб математично змодельовати соціальну ієрархію вовків використовують найбільш пристосоване рішення – альфа, друге та третє найкращі рішення, які називають бета та дельта відповідно, а решта кандидатів-рішень вважаються омегою [19].

Перший рівень – альфа, який є лідером, не обов’язково повинен бути найсильнішим вовком, головне – перевершення інших вовків у керуванні зграєю. Таким чином, він відповідає за прийняття рішень. Другий рівень –

бета, допомагає альфі приймати рішення. Отже, бета представляє собою радника альфи та вихователя зграї. Третій рівень – дельта, його задача – контролювати омегу; у цю категорію можуть входити скаути, старійшини, мисливці та доглядачі. Нарешті, четвертий рівень – омега, особини звичаної зграї, які поступаються всім домінуючим вовкам (рисунок 1.9).

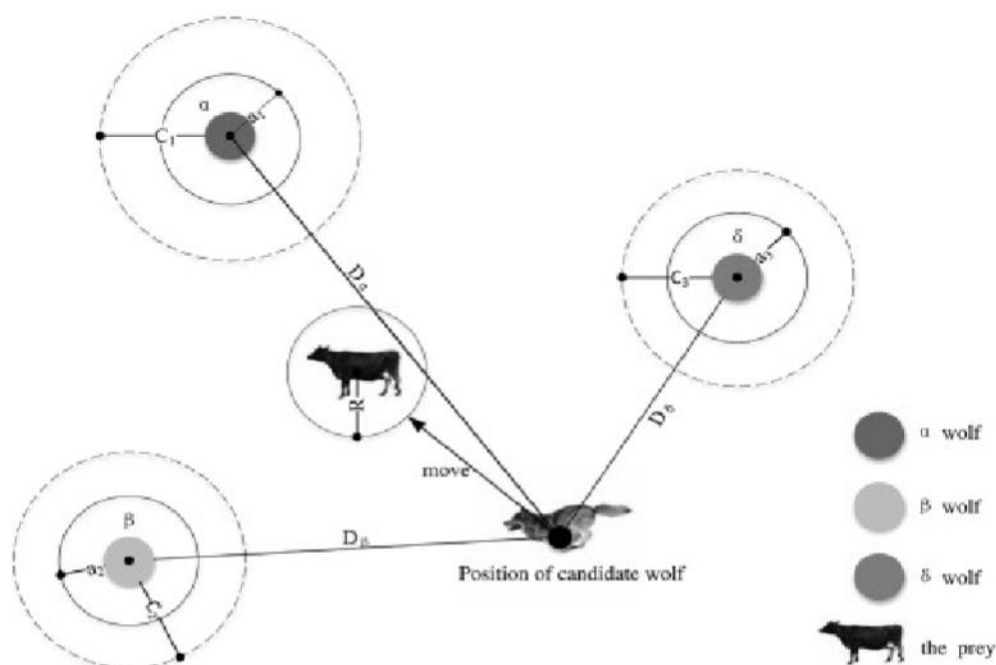


Рисунок 1.9 – Принцип роботи алгоритму зграї вовків

Алгоритм штучної бджолоїної колонії – це алгоритм ройового інтелекту, заснований на імітації поведінки колонії бджіл. Він розроблений турецьким вченим Д. Карабогою у 2005 році для розв’язання задач нелінійної оптимізації, а у 2019 році – задачі комівояжера [19].

Штучна колонія використовує алгоритм подібний до видобутку нектару медоносними бджолами. Для збору нектару в бджолиній колонії застосовуються бджоли-розвідники та бджоли-робітники. Перші проводять дослідження території, що оточує вулик, щодо наявності нектару. Після

повернення у вулик вони повідомляють інформацію про кількість нектару, напрям його розташування та відстань до нього. Далі, у найбільш відповідні області вилітають робітники, причому, чим більше нектару в цій галузі, тим більше бджіл вилітає в неї (рисунок 1.10). Алгоритм показав себе дуже ефективним у ході розв'язання низки практичних задач.

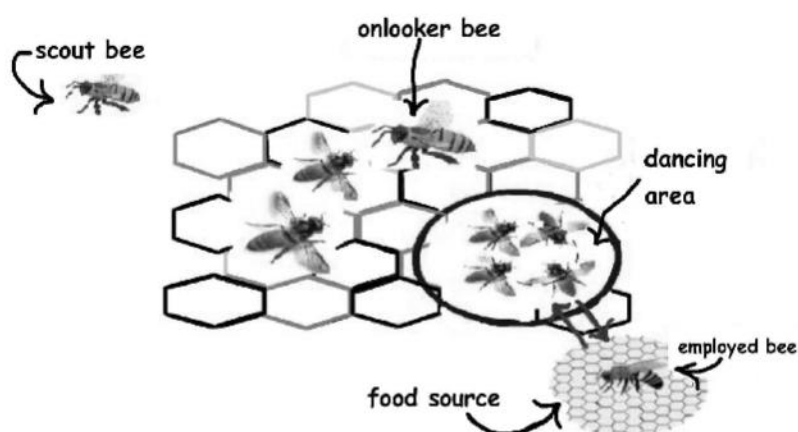


Рисунок 1.10 – Принцип роботи алгоритму штучної бджолоїної колонії

Поєднання низки евристик із різних алгоритмів дозволяє задати композиційний ройовий алгоритм, який комбінує різні підходи ройових алгоритмів для досягнення більш ефективних результатів у складних задачах оптимізації. Даний алгоритм використовує переваги низки розглянутих алгоритмів, об'єднуючи їх сильні сторони для розв'язання складних задач [6].

Отже, алгоритми ройового інтелекту намагаються об'єднати основні евристичні методи в рамках алгоритмічних схем більш високого рівня, направлених на ефективне вивчення простору пошуку. Дані методи використовують випадковість, елементи самонавчання, інтенсифікацію і диверсифікацію пошуку, адаптивні механізми управління, конструктивні

евристики і методи локального пошуку. Хороша реалізація метаевристики може забезпечити знаходження оптимального рішення за розумний час.

1.4 Висновки

У цьому розділі розглянуто комбінаторну природу задач дискретної оптимізації, наведено математичну постановку задач комівояжера та про покриття. Проведений аналіз вказує на те, що для розв'язання даних задач при великій розмірності графів найбільш ефективними є методи ройового інтелекту. Математична модель одного із таких алгоритмів буде детально розглянута в наступному розділі.

2 РОЗРОБКА МАТЕМАТИЧНОГО АПАРАТУ РОЙОВОГО АЛГОРИТМУ

2.1 Загальні принципи використання ройових алгоритмів

Загальна схема ройового алгоритму оптимізації наведена у таблиці 2.1. Спочатку в області пошуку формуємо певну кількість початкових наближень до шуканого рішення задачі, тобто ініціалізуємо популяцію агентів. У ході цього процесу можна застосувати детерміновані та стохастичні методи. Зазвичай агентів розподіляють випадковим чином рівномірно по всій області пошуку екстремуму [8].

Таблиця 2.1 – Схема ройового алгоритму

Крок	Опис
1	Ініціалізація популяції
2	Міграція агентів популяції
3	Завершення пошуку

За допомогою міграційних операторів переміщуємо агентів (випадкове блукання) таким чином, щоб наблизитися до глобального екстремуму цільової функції – функції пристосованості, придатності, корисності, яка оцінює "якість" популяції. Прикладами випадкових блукань можуть послужити траєкторія руху молекули в рідині або газі (броунівський рух), в природі у тварин для виживання (стратегія пошуку їжі), коливання цін акцій на фондовому ринку, фінансовий стан гравця. Всі описані випадки можуть бути

апроксимовані моделями випадкового блукання, навіть незважаючи на те, що вони можуть не бути повністю випадковими в реальному житті. Випадкове блукання пояснює поведінку багатьох процесів і представляє собою фундаментальну модель для зареєстрованої стохастичної активності.

У стохастичних алгоритмах оптимізації на етапі міграції випадкові блукання представляють велику цінність, оскільки дозволяють досліджувати простір рішень. Для генерації кроку дослідження зазвичай використовують кілька законів розподілу випадкових чисел. Приклад випадкового блукання показано на рисунку 2.1 [3].

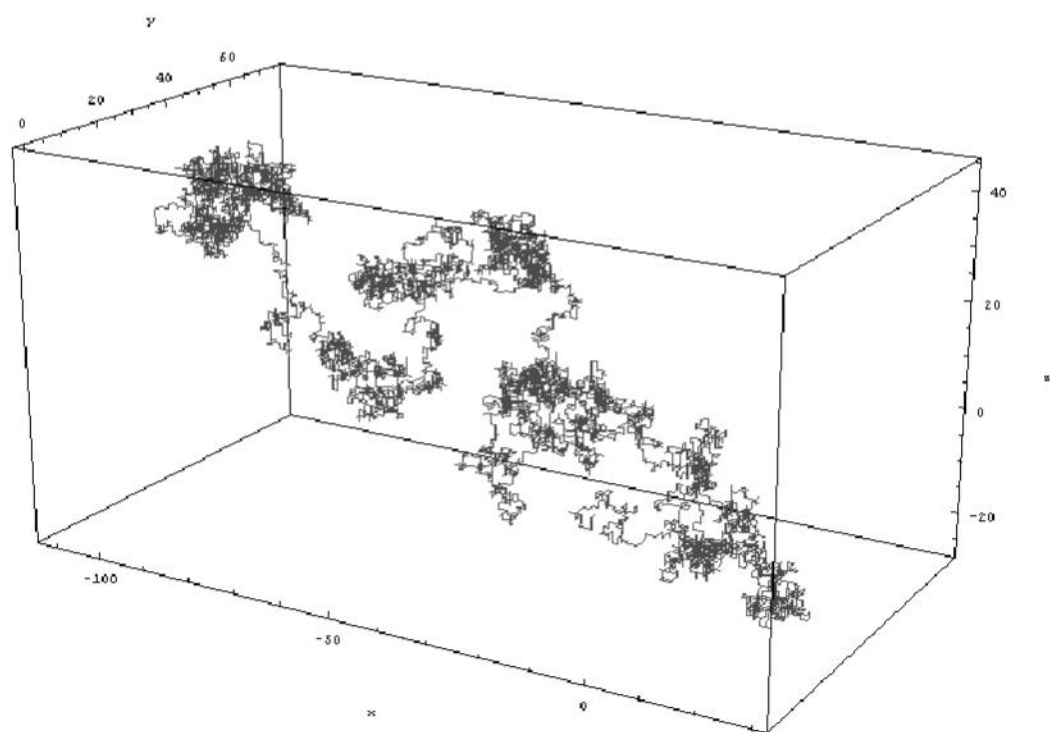


Рисунок 2.1 – Приклад випадкового блукання

В якості базової послідовності випадкових чисел для імітації випадкових факторів різної природи необхідно вибирати таку послідовність, яка може бути отримана з найменшими витратами машинного часу і, крім того, забезпечує простоту і зручність подальших перетворень. Практика показує, що

найкраще даним вимогам задовольняє послідовність випадкових чисел з рівномірним розподілом на інтервалі:

$$f(x|a,b) = \frac{1}{b-a}, \quad a \leq x \leq b. \quad (2.1)$$

У Python для генерації випадкового числа, розподіленого рівномірно на відрізок $[a, b] = [0, 1]$, використовується функція `random.uniform(a, b)`. Функція щільності ймовірності рівномірного розподілу наведена на рисунку 2.2 [3].

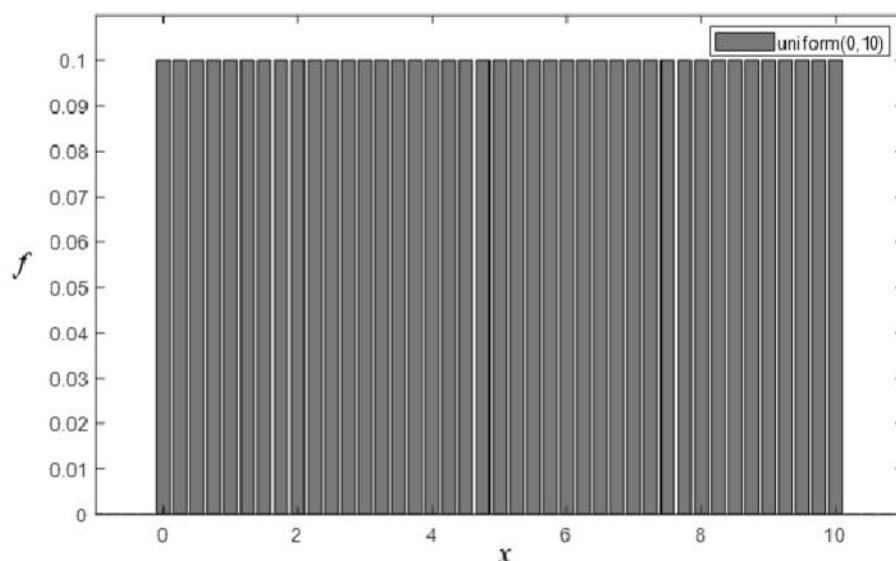


Рисунок 2.2 – Приклад функції щільності рівномірного розподілу

Останнім кроком ройового алгоритму є перевірка виконання умови закінчення ітераційного процесу (час роботи; кількість ітерацій або поколінь; стагнація алгоритму – найкраще отримане значення не змінюється впродовж декількох поколінь тощо) і, якщо вони виконані, припиняємо обчислення, вважаючи найкраще із знайдених положень агентів популяції наближеним розв’язком задачі.

Оскільки ітераційний процес не може тривати нескінченно, потрібно вибрати відповідну умову його закінчення – критерій збіжності ітерацій, при виконанні якого, останнє знайдене ітераційне наближення буде прийняте за шукане рішення [3, 10].

Розглянемо кілька критеріїв закінчення ітераційного процесу. Контролювати збіжність можна одночасно всіма критеріями або вибірково деякими з них. Зазвичай використовують такі критерії зупинки пошуку:

а) знайдено відомий глобальний оптимум $f(X^{best})$, де $X^{best} = (x_1, x_2, \dots, x_n)^T$;

б) досягнуто граничних значень:

1) ітерацій:

$$i > limit, \quad (2.2)$$

2) часу:

$$t > t_limit, \quad (2.3)$$

3) звернень до цільової функції:

$$f_call > f_limit. \quad (2.4)$$

в) стагнація пошуку:

1) найкраще рішення не змінюється досить довго;

2) всі агенти знаходяться на дуже близькій відстані eps від найкращого рішення;

3) середня якість популяції не змінюється досить довго.

2.2 Узагальнена модель композиційного алгоритму

Розглядаємо задачу мінімізації. Узагальнений композиційний алгоритм оптимізації включає в себе такі кроки [4–6, 8, 14]:

Крок 1. Ініціалізація параметрів алгоритму:

- $X_i = (x_{i1}, \dots, x_{in})^T$ – позиції пошуку, до яких рухаються E пошукових агентів;
- $O \geq E, S \ll E$ – кількість агентів-спостерігачів та агентів-розвідників;
- *limit* – рішення вважається забутим, якщо воно не змінювалося впродовж такої кількості ітерацій;
- K – максимальна кількість ітерацій алгоритму.

Для задачі неперервної оптимізації, якщо при генерації нового рішення його j -та компонента виходить за межі області D , то проводиться декілька спроб згенерувати її заново. Якщо після закінчення k спроб нове рішення постійно знаходиться за межами множини D , то дана координата замінюється на відповідне граничне значення:

$$x_{ij}^{new} = \begin{cases} LB_j, & \text{якщо } x_{ij} < LB_j; \\ UB_j, & \text{якщо } x_{ij} > UB_j, \end{cases} \quad (2.5)$$

де LB_j, UB_j – нижня та верхня межі множини допустимих рішень по j -ій координаті, $UB_j > LB_j$.

Крок 2. Для кожного агента E знаходяться координати його положення:

$$X_i = LB + rand \otimes (UB - LB), i = \overline{1, E}, \quad (2.6)$$

де $rand$ – $(n \times 1)$ -вектор рівномірно розподілених випадкових чисел;

$\otimes$ – поелементне множення.

Для задачі дискретної оптимізації необхідно згенерувати для кожного пошукового агента перестановку цілих чисел. Для цього запропоновано застосувати алгоритм найближчого сусіда для низки вершин графа G .

Крок 3. Для всіх отриманих рішень оцінюються позиції – значення цільової функції $f(X_i)$, $i = \overline{1, E}$.

Крок 4. Отримати нові рішення для пошукових агентів E :

$$X_{new_i} = X_i + U(a, b) \otimes (X_i - X_r), i = \overline{1, E}, \quad (2.7)$$

де $U(a, b)$ – $(n \times 1)$ -вектор незалежних дійсних випадкових чисел, рівномірно розподілених на інтервалі $[a, b]$; $a = -1$; $b = 1$; $b > a$;

індекс r вибирається випадково з множини $\{1, \dots, E\} / \{i\}$, тобто $r \neq i$.

Для задачі дискретної оптимізації аналогічно необхідно згенерувати для кожного пошукового агента перестановку цілих чисел, застосувавши певні перетворення уже відомих вхідних рішень. Для цього використано операцію реорганізації, яка полягає в наступному [15]:

1. Випадково відібрати декілька позицій у перестановці $P_1(P_2)$.
2. Запам'ятати їх та видалити з перестановки $P_2(P_1)$.
3. Скопіювати збережені пункти в маршрут $P_2(P_1)$ в такому ж порядку, в якому вони розташовані у перестановці $P_1(P_2)$.

Наприклад, розглянемо перестановки, які наведено в таблиці 2.2.

Таблиця 2.2 – Дві перестановки

P_1	2	3	4	5	6	1
P_2	4	5	2	1	6	3

Наприклад, візьмемо позиції 1, 3 та 4 для маршруту P_2 , в яких знаходяться пункти 4, 2 та 1 відповідно. Нова перестановка ініціалізується всіма пунктами маршруту P_1 за виключенням вузлів 4, 2, 1. Замість них на відкриті позиції копіюємо ці пункти в такому ж порядку, в якому вони розташовані у маршруті P_2 (таблиця 2.3). Аналогічно можна згенерувати іншу перестановку пунктів, змінивши ролі P_1 та P_2 .

Таблиця 2.3 – Приклад реорганізації на основі першої перестановки

A_1	–	3	–	5	6	–
A_1	4	3	2	5	6	1

Крок 5. Розрахувати нове значення цільової функції $f(X_{new_i})$.

Крок 6. Провести селекцію серед нових X_{new_i} та існуючих X_i рішень ($i = \overline{1, E}$) жадібним способом:

$$X_i = \begin{cases} X_{new_i}, & \text{якщо } f(X_{new_i}) \leq f(X_i); \\ X_i, & \text{інакше.} \end{cases} \quad (2.8)$$

Крок 7. Розрахувати ймовірність вибору p_i позиції X_i агентами-спостерігачами. Зазвичай використовують колесо рулетки.

Пропорційний відбір. Імовірність вибору p_i агентом-спостерігачем певної позиції виявляється пропорційною значенню фітнес-функції, тобто кожній позиції відповідає сектор колеса рулетки:

$$p_i = \frac{F(X_i)}{\sum_{i=1}^E F(X_i)}, \quad (2.9)$$

де $F(.)$ – певна функція (залежно від задачі, на максимум або мінімум).

Наприклад, для задачі на мінімум та максимум відповідно можна використати такі функції [24]:

$$p_i = \frac{\exp\left(\frac{-f(X_i)}{f_{mean}}\right)}{\sum_{i=1}^E \exp\left(\frac{-f(X_i)}{f_{mean}}\right)}, \quad (2.10)$$

$$p_i = \frac{\exp\left(\frac{f(X_i)}{f_{mean}}\right)}{\sum_{i=1}^E \exp\left(\frac{f(X_i)}{f_{mean}}\right)}, \quad (2.11)$$

де $f_{mean} = \frac{\sum_{i=1}^E f(X_i)}{E}$ – середнє значення якості позицій на даній ітерації.

Слабка сторона цього підходу у тому, що особини з дуже малим значенням функції пристосованості $f(X_i)$ мало досліджуються, що може призвести до передчасної збіжності алгоритму.

Лінійне ранжування, в якому використовується тільки номер позиції в упорядкованому списку, тоді ймовірність вибору можна описати формулою

$$p_i = \frac{1}{E} \cdot \left(a - (a-b) \cdot \frac{i-1}{E-1} \right), \quad (2.12)$$

$$b = 2 - a, \quad (2.13)$$

де $1 \leq a \leq 2$ – вибирається випадково;

i – номер позиції в упорядкованому списку.

Також можна застосовувати рівномірне ранжування, яке задається наступною формулою:

$$p_i = \begin{cases} \frac{1}{\mu}, & \text{якщо } 1 \leq i \leq \mu, \\ 0, & \text{якщо } \mu < i \leq E, \end{cases} \quad (2.14)$$

де $\mu \leq E$ — параметр (фактично кількість найкращих агентів).

Слід зазначити, що попередньо необхідно відсортувати агентів за значенням цільової функції (за зростанням для задачі на мінімум або за спаданням для задачі на максимум), зберігши порядкові номери агентів до сортування. При цьому у відсортованому списку найбільшу ймовірність відбору отримає агент із номером 1.

Сігма-відсікання виконує масштабування цільової функції за формулою:

$$p_i = \frac{F_{\sigma}(X_i)}{\sum_{i=1}^E F_{\sigma}(X_i)}, \quad (2.15)$$

$$F_{\sigma}(X_i) = 1 + \frac{f(X_i) - f_{mean}}{2 \cdot RMSE}, \quad (2.16)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^E (f(X_i) - f_{mean})^2}{E}}, \quad (2.17)$$

де $f_{mean} = \frac{\sum_{i=1}^E f(X_i)}{E}$ – середнє значення якості позицій на даній ітерації.

Вибір позиції можна ототожнити з вибором числа з інтервалу $[A; B]$, де A та B позначають відповідно початок і кінець фрагмента кола. Отже, у цьому випадку вибір зводиться до вибору числа з інтервалу $[0, 1]$ (відповідного сектору колеса рулетки), яке відповідає конкретній точці на колі (рисунок 2.3). Слід зазначити, що:

$$\sum_{i=1}^E p_i = 1. \quad (2.18)$$

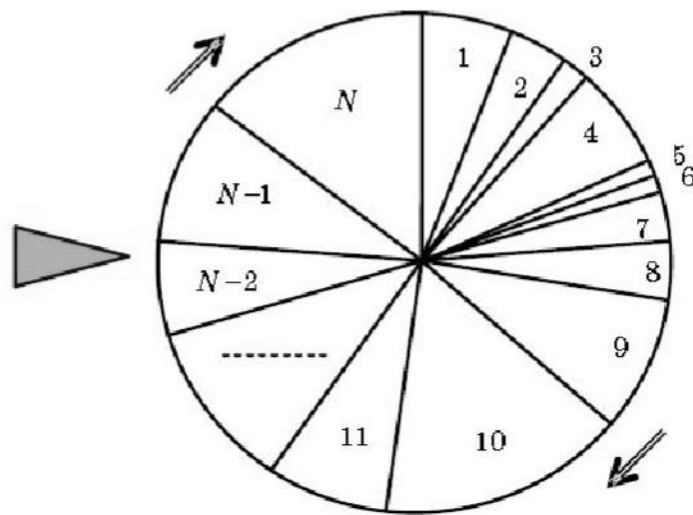


Рисунок 2.3 – Приклад роботи колеса рулетки

Крок 8. Отримати нові рішення для агентів-спостерігачів O аналогічно рішенням пошукових агентів на кроці 4. Але індекс r вибирається на основі ймовірностей p з кроку 7. Для цього генерується випадкове число $rand$ і перевіряється, в який сектор рулетки r воно потрапляє.

Для задач дискретної оптимізації тут необхідно виконати низку перетворень, які нагадують процедуру схрещування в генетичному алгоритмі. Щоб згенерувати кандидати, для агентів-спостерігачів запропоновано використати операції перестановки, зсуву та подвійної інверсії. У алгоритмі ці операції спочатку виконуються з однаковою ймовірністю вибору, а в наступних ітераціях вона змінюється відповідно до успіху кожної із них, що підвищує вибір успішної операції з використанням колеса рулетки, також задається лічильник успіху кожної операції [6].

Операція перестановки: два різних цілих числа x і y генеруються випадковим чином (від 1 до розмірності задачі n) та міняються місцями для нового маршруту (рисунок 2.4).

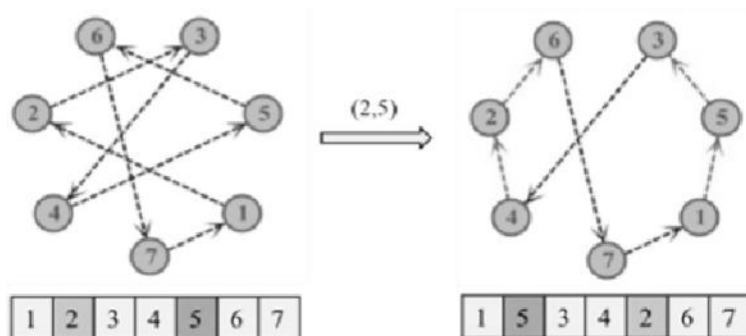


Рисунок 2.4 – Приклад операції перестановки

Операція зсуву: два різних цілих числа x і y генеруються випадковим чином (від 1 до розмірності задачі n). Потім відбувається циклічний зсув на одиницю вліво для їх позицій (рисунок 2.5).

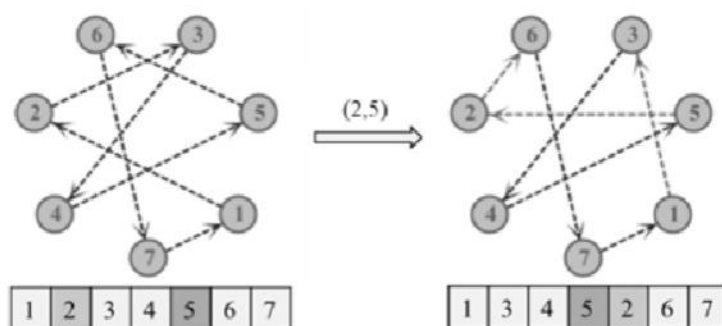


Рисунок 2.5 – Приклад операції зсуву

Операція подвійної інверсії формує значні зміни у заданому маршруті: генеруються дві підпослідовності:

$$A = [x, y] \text{ та } B = [y + 1, z],$$

які не пересікаються, з послідовними позиціями $[A, B]$. Кожна з підпослідовностей інвертується:

$$A^* = [y, x], B^* = [z, y + 1],$$

а потім послідовності міняються місцями – $[B^*, A^*]$ (рисунок 2.6).

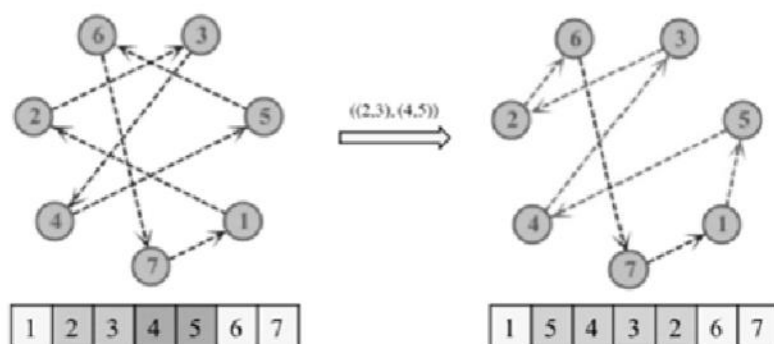


Рисунок 2.6 – Приклад операції подвійної інверсії

Крок 9. Повторити кроки 5 (оцінка позицій) та 6 (селекція рішень).

Крок 10. Генерація нових позицій замість “забутих”. Для таких позицій випадково генерують нові рішення на множині допустимих рішень D аналогічно до кроку 2.

Крок 11. Видалити S найгірших рішень і згенерувати нові аналогічно до кроку 2. Генеруємо для трьох найгірших рішень.

Крок 12. Перевіряємо критерій зупинки пошуку (час, кількість ітерацій, стагнація пошуку тощо).

Крок 13. Якщо умова на кроці 12 не виконана, то визначити найкращого агента s^{best} та повернутися до кроку 4; інакше – перейти на крок 14.

Крок 14. Для найкращого маршруту застосовується геометрична 2-орт евристика, ідея якої в тому, що з маршруту систематично видаляються групи з двох ребер $(v_k, v_t), (v_b, v_m)$, які замінюються іншими двома ребрами $(v_k, v_b), (v_t, v_m)$. Слід зазначити, що умова циклічності маршруту не повинна порушуватися. Якщо така заміна дозволяє зменшити довжину циклу [18]:

$$\Delta = c(v_k, v_t) + c(v_b, v_m) - (c(v_k, v_b) + c(v_t, v_m)) \geq 0, \quad (2.19)$$

то ребра $(v_k, v_b), (v_t, v_m)$ зберігають, а даний алгоритм продовжує свою роботу.

Крок 15. Визначити та вивести глобально найкраще рішення $f^{best}(X^{best})$ для агента s^{best} :

$$f_{\min} = f^{best}(X^{best}). \quad (2.20)$$

Перевагами алгоритму є:

- простота обчислень;
- збіжність до оптимуму;
- можливість модифікації;
- можливість розпаралелювання.

а з недоліків можна виділити:

- складність аналізу;
- невідомий час збіжності.

Однією з основних проблем розробки метаевристик є забезпечення балансу між інтенсивністю і широтою пошуку. Інтенсифікація пошуку вимагає швидкої збіжності алгоритму, що означає швидке зменшення різноманітності популяції. А диверсифікація пошуку навпаки дозволяє забезпечити більш широкий огляд простору пошуку і більш високу ймовірність локалізувати глобальний екстремум задачі, тобто вона вимагає збереження різноманітності популяції якомога більшу кількість поколінь.

Найбільш розвиненими механізмами вирішення даної проблеми є механізми самоадаптації популяційних алгоритмів. У даний час нові високоефективні популяційні алгоритми розробляють за допомогою гібридизації різних метаевристик. Також важливою задачею, у зв'язку з доступністю потужних обчислювальних засобів, є розробка метаевристик з використанням паралелізації обчислень.

Таким чином, досліднику необхідно адаптувати кожний метод до конкретної задачі. Варто пам'ятати, що пошук рішення завжди є мистецтвом, яке можна освоїти тільки через практику, експериментуючи з різними підходами для вирішення специфічних задач.

2.3 Висновки

У розділі сформульовано основні принципи узагальненого композиційного алгоритму оптимізації. Представлено модифіковану математичну модель для розв'язання цілочисельних комбінаторних оптимізаційних задач. У наступному розділі необхідно виконати експериментальні дослідження ефективності роботи алгоритму.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Обґрунтування вибору мови програмування

Однією з найпопулярніших мов програмування для вирішення інженерних задач є Python (рисунок 3.1). Завдяки швидкому розвитку обчислювальної техніки постійно змінюються вимоги до мов програмування. Зокрема, дедалі більше значення набувають інтерпретовані мови, оскільки вони є простими та гнучкими інструментами, а потужність сучасних персональних комп'ютерів дозволяє забезпечити достатню швидкість виконання програм, написаних на таких мовах. У той же час, основною перевагою компільованих мов залишається створення високошвидкісного коду.

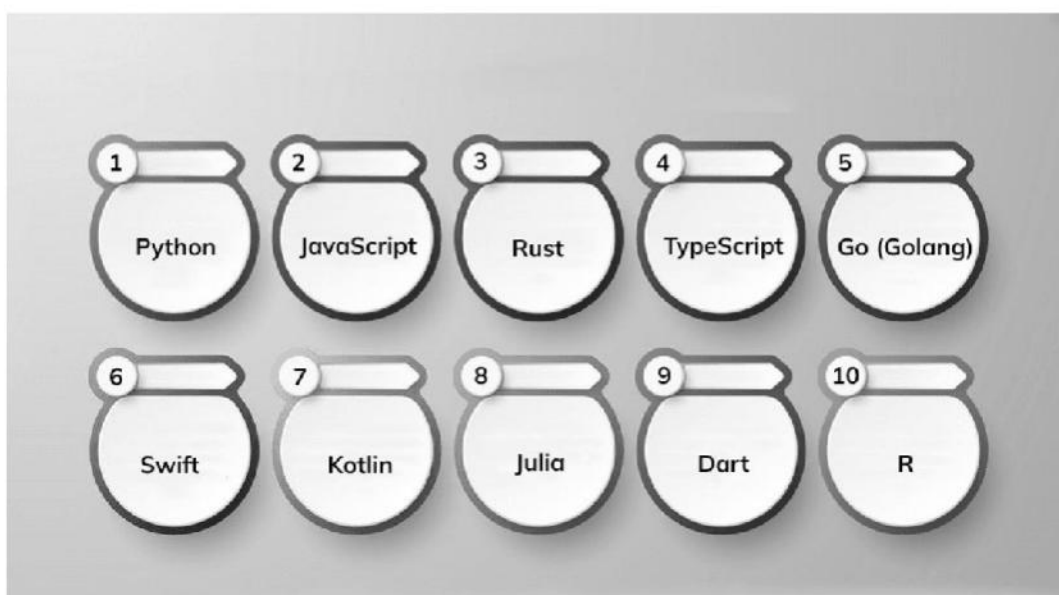


Рисунок 3.1 – Рейтинг мов програмування “майбутнього” [21]

Однією із таких простих, функціональних і гнучких мов є Python. Його очевидною перевагою є те, що інтерпретатор Python реалізований практично на всіх платформах і операційних системах. Важлива риса – розширюваність мови, тобто можливість її вдосконалення зацікавленими програмістами.

Наявність великої кількості різноманітних програмних модулів забезпечує додаткові функції. Можна навести такі модулі: Numerical Python (розширені математичні можливості), Tkinter (розробка додатків з використанням графічного інтерфейсу користувача), OpenGL (використання бібліотеки графічного моделювання об'єктів). Недоліком даної мови є порівняно невисока швидкість виконання програм, але її можна нівелювати перевагами.

Дана мова програмування поєднує потужність і гнучкість з високою ефективністю виконавчого коду й можливістю доступу до апаратних ресурсів комп'ютера. Тому програмне забезпечення для розв'язання поставленої задачі буде розроблятися саме на цій мові [22].

Основні технічні та мінімальні системні вимоги до розробленої програми: операційна система *Windows 10*; наявність клавіатури та миші; процесор *Core i5 (Ryzen 5)* або кращий; оперативна пам'ять – 8 Гб; відеопам'ять – 1 Гб; жорсткий диск – 500 Гб і більше.

Таким чином, оскільки дуже важливо для розробки нашої програми наявність як потужної математичної компоненти, так і бібліотек для роботи з дискретними об'єктами, то мовою програмування додатку вибрана Python.

3.2 Опис роботи програми

Концептуально система складається з модуля керування та функціональних модулів. Модуль керування відповідає за базові функції для маніпуляції з даними та їх візуалізацією. Центральний модуль керування є ключовою компонентою системи. Він відповідає за координацію та керування роботою інших складових, виконуючи такі важливі функції, як прийняття рішень, розподіл ресурсів, збирання та аналіз даних, керування взаємодією з користувачем та іншими компонентами, введення журналу подій.

Відповідно функціональні модулі реалізують обробники даних та візуалізатори. Модуль вводу вхідних даних відповідає за збирання та підготовку інформації, яка буде використовуватись в програмі. Його основна функція – отримання даних від користувача або з зовнішніх джерел і їх передавання до центрального модуля керування, який переправляє дані до модуля опрацювання вхідних даних для подальшої обробки. Відповідно далі можна переглянути вибрані зображення в інтерфейсі користувача.

Для роботи програми потрібно використовувати файли з розширенням **.json*, внутрішній вміст яких можна побачити, наприклад в текстовому редакторі NotePad++. Виконання самого алгоритму може бути розбито на три основні блоки:

- I. Надання початкового значення необхідним параметрам.
- II. Обчислення результату ітеративно.
- III. Перевірка критерію зупинки процесу.

Якщо блок 3 повертає значення true, то алгоритм припиняє роботу і виконує такі операції:

1. Виводить дані на екран у вигляді окремого вікна, створює два додаткових вікна із зображенням результатів.

2. Записує результат у файл, якщо було вибрано відповідні пункти меню.

За допомогою наочної демонстрації роботи алгоритму можна забезпечити вищий рівень розуміння його роботи у ході розв'язання оптимізаційної задачі. Тому для забезпечення демонстрації поведінки алгоритму в різних умовах користувач програми має можливість регулювати параметри.

3.3 Результати експериментів

Оскільки алгоритм ефективно використовує експлуатаційний і дослідницький процеси для забезпечення достатньої різноманітності в популяції колонії, то не потрібно задавати їй надвеликий розмір.

Значення *limit* не повинно бути занадто великим, щоб гарантувати оновлення популяції. Але при малій кількості джерел нектару значення *limit* все ж таки доцільно вибирати більшим, щоб не "загубилися" значення, які могли б бути покращені надалі.

Через те, що кількість параметрів у алгоритмі досить мала, то до вибору значення максимальної кількості ітерацій K потрібно ставитися досить акуратно.

Задача 1. Порівняння ефективності роботи алгоритмів з випадковим вибором евристик (ABB) та з вибором за успішністю їх роботи (ABY) наведено для 4-х задач з бібліотеки TSPLib [23]. Використано найкращі отримані параметри для агентів: $E = 50, O = 50, S = 3, \text{limit} = 300, K = 10000$ або

стагнація обчислень, 15 запусків програми. Довжини маршрутів у таблиці 3.1 показано без стандартних одиниць вимірювання. Для даних задач наведено середню відносну похибку та середній час пошуку розв'язку (с), які розраховано за формулами:

$$\varepsilon = \frac{1}{4} \cdot \sum_{j=1}^4 \frac{|L_j - L_j^{\text{opt}}|}{L_j^{\text{opt}}} \cdot 100\%, \quad (3.1)$$

$$t = \frac{\sum_{j=1}^4 t_j}{4}, \quad (3.2)$$

де L_j – середня довжина маршруту для j -ої задачі;

L_j^{opt} – найкращий відомий розв'язок j -ої задачі;

t_j – середній час розв'язання j -ої задачі.

Таблиця 3.1 – Результати експериментів

Назва задачі			<i>kroa100</i>	<i>tsp225</i>	<i>pcb442</i>	<i>u1060</i>
Найкращий відомий результат			21282	3916	50778	224094
Алгоритм	АВВ	Середня довжина	25841,23	4457,52	56567,45	242335,24
		Середній час, с	55,56	85,47	285,56	944,53
	АВУ	Середня довжина	21639,54	4084,68	52213,34	229675,45
		Середній час, с	217,45	927,57	1664,46	3324,67

У таблиці 3.2 показано узагальнення результатів експериментів. Можна помітити, що алгоритм без налаштувань з випадковим вибором евристики менш точний, але швидше працює, що пов'язано з зациклюванням програми на поганих розв'язках.

Таблиця 3.2 – Порівняння ефективності роботи
алгоритмів на тестових задачах

#	АВВ	АВУ	Результат порівняння
Середня відносна похибка, %	13,698	2,826	-10,872 (АВУ)
Середній час пошуку, с	342,780	1533,537	-1190,757 (АВВ)

На рисунках 3.2 та 3.3 наведено наглядне графічне відображення результатів роботи.

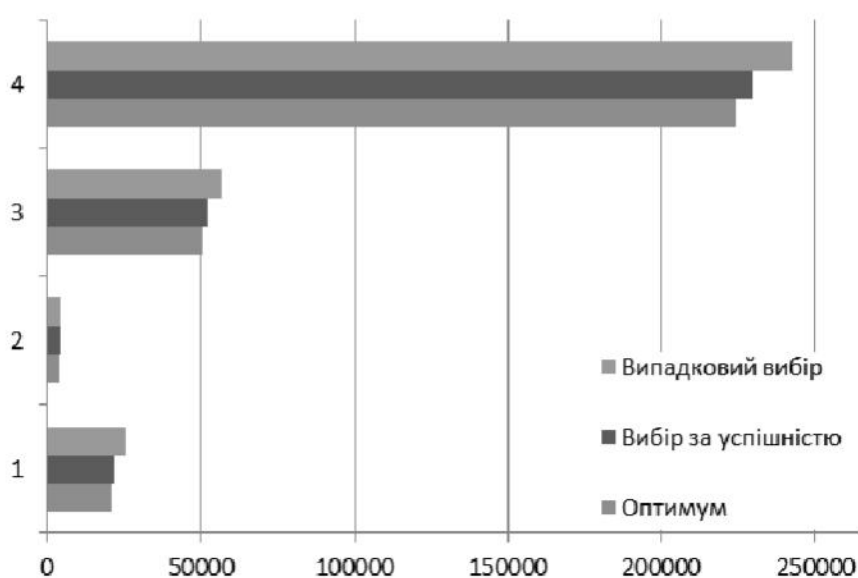


Рисунок 3.2 – Порівняння маршрутів

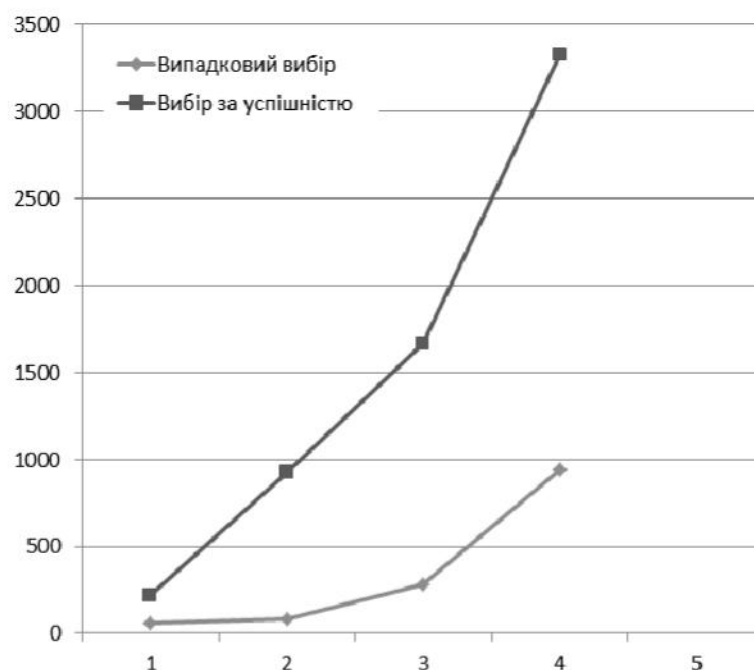


Рисунок 3.3 – Порівняння часу розв'язання

Задача 2. Тестування ефективності роботи алгоритму з вибором за успішністю на 2-х практичних квадратичних задачах про призначення [24, 25]. Використано аналогічні до задачі 1 параметри алгоритму АВУ. Застосована зміна цільової функції відповідно до заданої задачі, а також не застосовуються геометричні евристики, які використано для покращення розв'язку задачі комівояжера. Обмеження по часу розв'язання складає 30 хвилин на запуск або стагнація розрахунків. Результати пошуку показано у таблиці 3.3.

Перша задача els19 представляє собою опис відстаней для 19 різних відділень у госпіталі та потоки пацієнтів між ними. Метою є пошук такого оптимального розміщення відділень, яке мінімізує загальну вартість переміщення пацієнтів між ними, зважаючи на відстані.

Друга задача полягає у оптимізації розташування 26 літер на клавіатурі таким чином, щоб пари літер (біграми), які часто зустрічаються, розташовувалися поруч один із одною, що може підвищити швидкість і

зручність набору тексту. У даному випадку потоки представляють частоту розташування літер одна за одною, а відстань – трудомісткість переходу між клавішами (час переходу). Наприклад, QWERTY-клавіатура не найоптимальніша – вона розроблена, щоб уповільнити набір та уникнути заїдань на механічних друкарських машинках. Сучасні клавіатури типу Dvorak або Colemak намагаються мінімізувати цільову функцію для задачі QAP. Дані першої матриці відповідають часу друку середньостатистичного стенографіста, а друга матриця описує частоту виникнення біграм у різних мовах (понад 100000 пар).

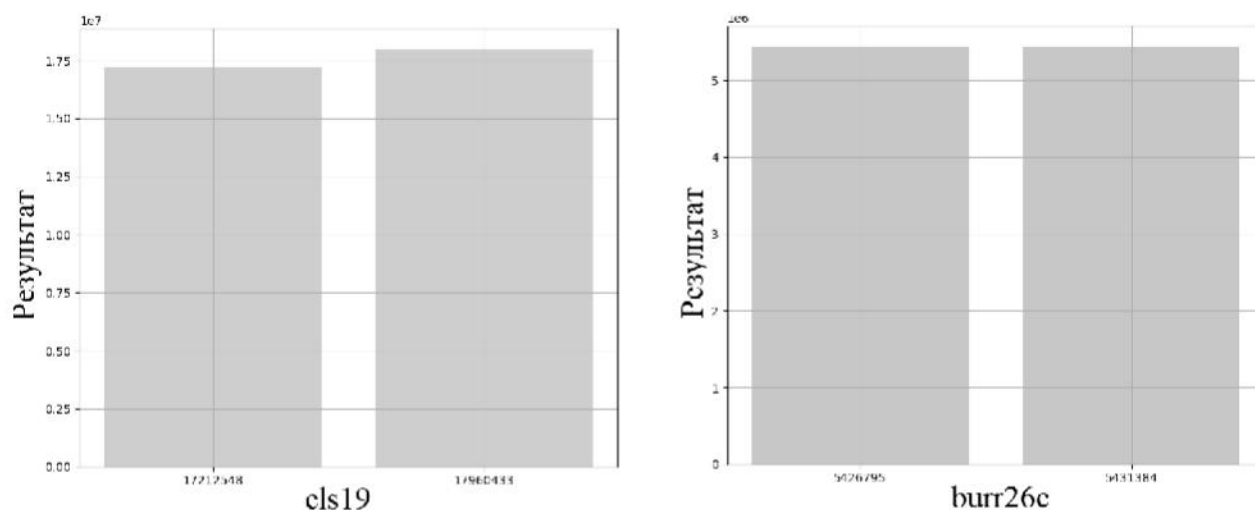
Таблиця 3.3 – Результати розв’язання задачі

Назва задачі	els19	eng_bur26c
Найкращий відомий результат	17212548	5426795
Середнє значення функції	17960433	5431384
Середній час, с	896,010	1264,011

У таблиці 3.4 наведено оцінки ефективності роботи алгоритму на заданих тестових задачах порівняно з оптимальним розв’язком. Оцінка ефективності є важливим етапом у визначенні практичної застосовності алгоритму для реальних задач. На рисунку 3.4 наведено відповідні бар-графіки.

Таблиця 3.4 – Порівняння ефективності роботи

Алгоритм		АВУ
Середня відносна похибка, %	1	4,345
	2	8,456

Рисунок 3.4 – Порівняння результатів розв’язання
квадратичної задачі про призначення

Визначено, що операція перестановки має низьку успішність та загалом нечасто дозволяє покращити розв’язок, на другому місці розташувалась операція зсуву, а найбільш часто використовувалась операція подвійної інверсії.

Загалом операція перестановки покращує розв’язок, але дуже повільно; операції зсуву та подвійної інверсії поодиночі доходять до плато насичення; однакова ймовірність вибору операції покращення маршруту агентами (випадковий вибір) програє подібній процедурі, але з вибором складових операцій (з вибором), за їх успішністю у ході розв’язання задачі. Крім того, останні дві стратегії поступово продовжують покращувати розв’язок залежно

від кількості ітерацій алгоритму. Найкраще себе проявила стратегія з вибором успішних операцій.

3.4 Висновки

У даному розділі представлено розроблене програмне забезпечення, наведено опис його параметрів. Програма дозволяє керувати параметрами композиційного ройового алгоритму й аналізувати ефективність його роботи. Отримано, що алгоритм досить точний, а використання додаткових евристик дозволить покращити результат роботи, при цьому може збільшитись час обчислень.

ВИСНОВКИ

У даній роботі на двох задачах комбінаторної оптимізації, пов'язаних із пошуком найкоротшого циклічного маршруту та квадратичною задачею про призначення, протестовано композиційний ройовий алгоритм. Розглянуто його математичну модель та особливості вибору складових евристик. Описано особливості розробленого програмного модуля. Тестування ефективності алгоритму з випадковим вибором евристик та за стратегією успішності проведено на низці задач комівояжера. Загалом середні похибки коливаються від 2,826% для першої задачі до 4,345...8,456% – для другої.

Визначено, що алгоритм досить точний, але повільний, що відбувається для задач дискретної оптимізації, враховуючи їх природу, та формування перестановок високої якості. Крім того, стохастичність алгоритму задає умову необхідності вдалого підбору параметрів алгоритму під кожну конкретну задачу. Таким чином, застосування композиційного ройового алгоритму дозволяє отримати хороше квазіоптимальне рішення, не потребуючи громіздких обчислень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Триус Ю.В. Сучасні тенденції у розвитку методів і засобів розв'язування оптимізаційних задач. *Новітні комп'ютерні технології*. К., 2018. С. 157–164.
2. Субботін С.О., Олійник А.О., Олійник О.О. Ітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей. Запоріжжя: ЗНТУ, 2009. 375 с.
3. Simon D. *Evolutionary Optimization Algorithms: Biologically Inspired and Population-Based Approaches to Computer Intelligence*. John Wiley & Sons, 2013. 776 p.
4. Discrete Swarm Intelligence Optimization Algorithms Applied to Steel-Concrete Composite Bridges / D. Martínez-Muñoz, C.J. Garcia, J. Martí-Albiñana, V. Yepes. *Engineering Structures*. 2022. Vol. 266. 12 p. URL: <https://victoryepes.blogs.upv.es/files/2022/07/1-s2.0-S0141029622007088-main.pdf>. (дата звернення 23.04.2025)
5. Gunduz M. Aslan M. DJAYA: A Discrete Jaya Algorithm for Solving Traveling Salesman Problem. *Applied Soft Computing*. 2021. Vol. 105. 15 p. URL: <https://www.sciencedirect.com/science/article/abs/pii/S1568494621001988?via%3Dihub>. (дата звернення 15.04.2025)
6. A Discrete JAYA Algorithm Based on Reinforcement Learning and Simulated Annealing for the Traveling Salesman Problem / J. Xu, W. Hu, W. Gu, Y. Yu. *Mathematics*. 2023. Vol. 11. 23 p. URL: <https://ideas.repec.org/a/gam/jmathe/v11y2023i14p3221-d1199888.html>. (режим доступу 29.04.2025)

7. Погорілий С.Д., Потебня А.В. Новітній метод розв'язання задач комбінаторної оптимізації великої розмірності. *Наукові праці ДонНТУ. Серія «Інформатика, кібернетика та обчислювальна техніка»*. 2014. №1. С. 114–125.

8. Евристичні алгоритми дискретної оптимізації / О.О. Лаврінчук, Р.В. Сіраченко, Ю.Ю. Іванов та інші. *LIV Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету*. Вінниця, 24-27.03.2025. URL: <https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23530/19764>. (дата звернення 24.04.2025)

9. Тимофієва Н.К. Про деякі властивості множини розв'язків задачі комівояжера. *Control Systems and Computers. Фундаментальные и прикладные проблемы Computer Science*. К., 2018. № 5. С. 3–12.

10. Xie J., Carrillo L.R.G., Jin L. An Integrated Traveling Salesman and Coverage Path Planning Problem for Unmanned Aircraft Systems. *IEEE Control Systems Letters*. 2019. № 3. pp. 67–72.

11. Mollabakhshi N., Eshghi M. Combinational Circuit Design Using Agents Algorithm. *IEEE Conference Anthology*. Piscataway, 2014. pp. 1–4.

12. Solution of a Large-Scale Traveling-Salesman Problem. In 50 Years of Integer Programming 1958-2008 From the Early Years to the State-of-the-Art / V. Chantal, W.J. Cook and others. Springer-Verlag Berlin, 2010. pp. 7–28.

13. CORAL. QAPLIB-Problem Instances and Solutions. URL: <https://coral.ise.lehigh.edu/data-sets/qaplib/qaplib-problem-instances-and-solutions/>. (дата звернення 05.05.2025)

14. Faizal H., Adel B. Particle Swarm Algorithm Variants for the Quadratic Assignment Problems – A Probabilistic Learning Approach. *Expert Systems with Applications*. 2015. Vol. 44. 35 p. URL: <https://www.researchgate.net/>

publication/282590022_Particle_Swarm_Algorithm_Variants_for_the_Quadratic_Assignment_Problems_-_A_Probabilistic_Learning_Approach. (дата звернення 20.04.2025)

15. Кравець П., Пасічник В., Проданюк М. Ігрова самоорганізація гамільтонового циклу графа. *Information Systems and Networks*. № 10. 2021. pp. 13–32.

16. An Effective Iterated Two-Stage Heuristic Algorithm for the Multiple Traveling Salesmen Problem / J. Zheng, Y. Hong, W. Xu, W. Li, Y. Chen. *Computational Operations Research*. 2022. Vol. 143. 14 p. URL: <https://arxiv.org/pdf/2201.09424>. (дата звернення 15.04.2025)

17. Helsgaun K. An Effective Implementation of k-opt Moves for the Lin-Kernighan TSP Heuristic. URL: <http://akira.ruc.dk/~keld/research/LKH/KoptReport.pdf>. (23.04.2025)

18. Travelling Salesman Problem using 3OPT and 2OPT Perturbation. URL: https://github.com/source-nerd/tsp_3opt_2opt. (дата звернення 27.04.2025)

19. Rao R.V. Jaya: A Simple and New Optimization Algorithm for Solving Constrained and Unconstrained Optimization Problems. *International Journal of Industrial Engineering Computations*. 2016. Vol. 7. pp. 19–34.

20. Muliarevych O. Cyber-Physical System for Solving Travelling Salesman Problem. *Advances in Cyber-Physical Systems*. 2017. Vol. 2. № 1. pp. 22–28.

21. Рейтинг мов програмування 2025. URL: <https://merehead.com/ua/blog/rate-programming-languages-2025/>. (дата звернення 26.04.2025)

22. Python documentation. URL: <https://docs.python.org/3/index.html> (дата звернення 03.05.2025)

23. Traveling Salesman Problem. University of Waterloo. URL: <http://www.math.uwaterloo.ca/tsp/index.html>. (02.05.2025)
24. A Hybrid Discrete Particle Swarm Algorithm for Hard Binary CSPs. *Advances in Natural Computation* / Q. Yang,, J. Sun, J. Zhang, C. Wang. 2006. Vol. 4222. pp. 184–194.
25. Burkard R.E., Karisch S.E., Rendl F. A Quadratic Assignment Problem Library. *Journal of Global Optimization*. 1997. Vol. 10. pp. 115–119.

ДОДАТКИ

Додаток А
(обов'язковий)
Технічне завдання

ЗАТВЕРДЖУЮ

завідувач кафедри АІТ

д.т.н., проф. Бісікало О. В.

«23» 03 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на бакалаврську кваліфікаційну роботу

«Дослідження ройового композиційного алгоритму

для задач дискретної оптимізації»

08-02.БКР.012.02.000 ТЗ

Керівник роботи:

к.т.н., доц. каф. АІТ

Іванов Ю. Ю.

«24» 03 2025 р.

Виконавець:

ст. гр. ІСТ-216

Лаврінчук О. О.

«24» 03 2025 р.

1. Назва та галузь застосування

Робота спрямована на розв'язання задач комбінаторної оптимізації, які виникають у різних галузях науки та техніки, із використанням композиційного ройового алгоритму.

2. Підстави для розробки

Розробку здійснювати на підставі наказу по університету №97 від 20.03.2025 та завдання до бакалаврської кваліфікаційної роботи, складеного та затвердженого кафедрою "Автоматизації та інтелектуальних інформаційних технологій".

3. Мета та призначення розробки

Метою роботи є підвищення ефективності роботи композиційного ройового алгоритму для розв'язання задач дискретної оптимізації за рахунок застосування стратегії вибору складових евристик.

4. Етапи виконання робіт

Виконавцем перерахованих у даному розділі етапів є студент групи ІІСТ-216 Лаврінчук Олег Олександрович факультету Інтелектуальних інформаційних технологій та автоматизації Вінницького національного технічного університету, а замовником є кафедра автоматизації та інтелектуальних інформаційних технологій.

№	Назва та зміст етапу	Термін виконання	
		початок	закінчення
1	Аналіз предметної області роботи	21.03.2025	04.04.2025
2	Розробка математичного апарату ройового алгоритму	05.04.2025	26.04.2025
3	Розробка програмного забезпечення та експериментальні дослідження	27.04.2025	16.05.2025
4	Оформлення пояснювальної записки і графічного матеріалу	17.05.2025	27.05.2025

5. Джерела розробки

1. Gunduz M., Aslan M. DJAYA: A Discrete Jaya Algorithm for Solving Traveling Salesman Problem. *Applied Soft Computing*. 2021. Vol. 105. 15 p.
2. Discrete Swarm Intelligence Optimization Algorithms Applied to Steel-Concrete Composite Bridges / D. Martínez-Muñoz, C.J. Garcia, J. Martí-Albiñana, V. Yepes. *Engineering Structures*. 2022. Vol. 266. 12 p.
3. A Discrete JAYA Algorithm Based on Reinforcement Learning and Simulated Annealing for the Traveling Salesman Problem / J. Xu, W. Hu, W. Gu, Y. Yu. *Mathematics*. 2023. Vol. 11. 23 p.

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програми: операційна система *Windows 7 / 10* або новіша; наявність клавіатури та миші; процесор *Core i5 (Ryzen 5)* або кращий; оперативна пам'ять – 8 ГБ; відеопам'ять – 512 МБ; жорсткий диск – 200 ГБ і більше.

Вхідні дані: файли зі структурою задачі (кількість точок, координати, потоки та відстані між об'єктами); параметри алгоритму (кількість агентів; кількість ітерацій алгоритму тощо); кількість запусків для збору статистики; набір тестових задач. Результати роботи програми: (квазі)оптимальне значення цільової функції; похибки розрахунків; графічне відображення розв'язку.

Студент гр. ІІСТ-216

(підпис)

Лаврінчук О. О.
(ініціали та прізвище)

Додаток Б
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ДОСЛІДЖЕННЯ РОЙОВОГО КОМПОЗИЦІЙНОГО АЛГОРИТМУ
ДЛЯ ЗАДАЧ ДИСКРЕТНОЇ ОПТИМІЗАЦІЇ



Рисунок Б.1 – Схема роботи рою

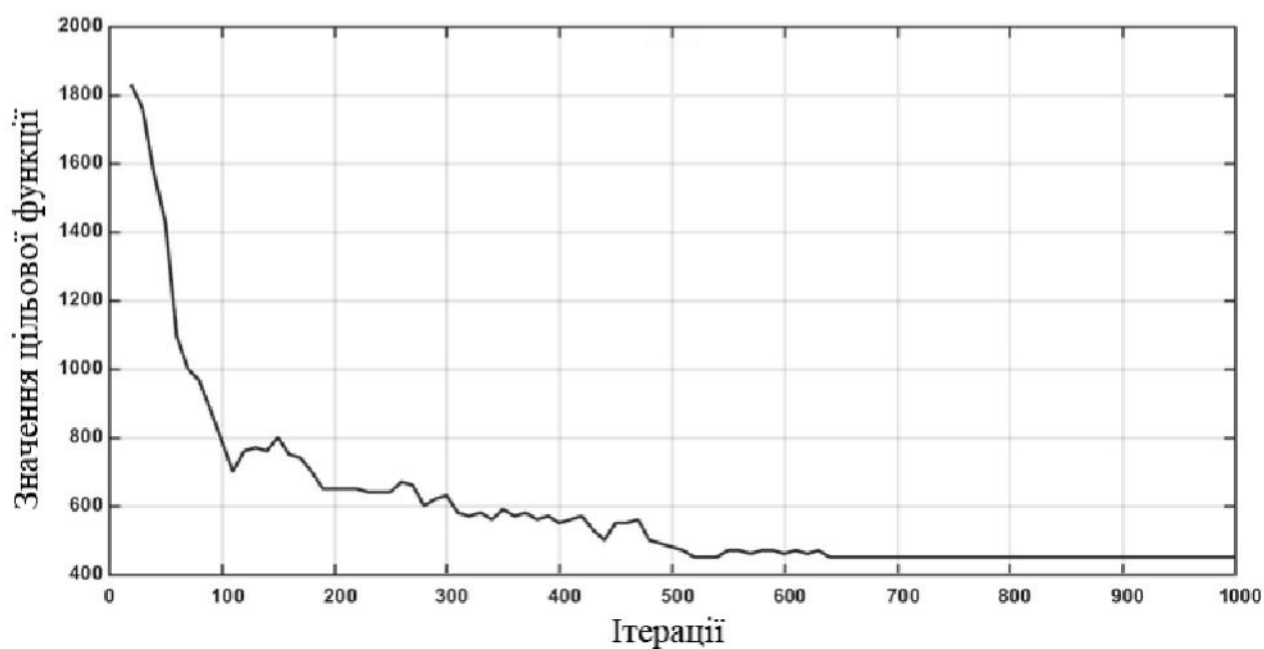


Рисунок Б.2 – Приклад кривої збіжності алгоритму



Рисунок Б.3 – Схема програми

Додаток В
(обов'язковий)

Лістинг програмного забезпечення

```
#####

import matplotlib.pyplot as plt
import numpy as np
import random as r

class Node:
    def __init__(self, idn, x, y):
        self.idn = idn
        self.pos = np.array((float(x), float(y)))

class Agent:
    def __init__(self):
        self.chosen_nodes = []
        self.recuiter = True
        self.distance = 0.0

    def choose_rand_move(self, move, nods):
        for i in range(move):
            if self.is_complete():
                break
            else:
                sel = nods[r.randint(0, len(nodes) - 1)]
                while sel in self.chosen_nodes:
                    sel = nods[r.randint(0, len(nodes) - 1)]
                self.chosen_nodes.append(sel)
        self.total_distance()
```

```

def change_role(self, role):
    self.recuiter = role

def replace_nodes(self, nods):
    self.choosen_nodes = nods
    self.total_distance()

def total_distance(self):
    distance = 0.0
    for i in range(len(self.choosen_nodes) - 1):
        node1 = self.choosen_nodes[i]
        node2 = self.choosen_nodes[i + 1]
        distance += np.linalg.norm(node1.pos - node2.pos)
    distance += np.linalg.norm(self.choosen_nodes[-1].pos - self.choosen_nodes[0].pos)
    self.distance = distance

def is_complete(self):
    if len(self.choosen_nodes) >= len(nodes):
        return True
    else:
        return False

def load_nodes(filename):
    ret = []
    with open(filename) as f:
        nodes_s = f.readlines()
    nodes_s = [x.strip() for x in nodes_s]
    for n in nodes_s:
        node = n.split(' ')
        ret.append(Node(node[0], node[1], node[2]))
    return ret

```

```

nodes = load_nodes("data")

def main():
    epoch = 10000
    n_agent = 50
    n_move = 3
    agents = []
    best_agent = Agent()
    e = 0
    for i in range(n_agent):
        agents.append(Agent())
    while not best_agent.is_complete():
        print ("\nEpoch", e + 1)
        print (forward pass)
        for agent in agents:
            agent.choose_rand_move(n_move, nodes)
        print ("evaluating")
        agents = sorted(agents, key=lambda be: be.distance, reverse=False)
        best_agent = agents[0]
        print ("Best distance so far", best_agent.distance)
        print ("Best route so far", [n.idn for n in best_agent.chosen_nodes])
        print ("Agents are making decision to be recruiter or follower")
        Cmax = max(agents, key=lambda b: b.distance).distance
        Cmin = min(agents, key=lambda b: b.distance).distance
        recruiters = []
        for agent in agents:
            Ob = (Cmax - agent.distance) / (Cmax - Cmin)
            probs = np.e ** (-(1 - Ob) / (len(agent.chosen_nodes) * 0.01))
            rndm = r.uniform(0, 1)
            if rndm < probs:
                agent.change_role(True)

```

```

        recruiters.append(agent)
    else:
        agent.change_role(False)
    print ("number of recruiter", len(recruiters))
    print ("Agents are choosing their recruiter")
    divider = sum([(Cmax - agent.distance) / (Cmax - Cmin) for agent in recruiters])
    probs = [((Cmax - agent.distance) / (Cmax - Cmin)) / divider for agent in recruiters]
    cumulative_probs = [sum(probs[:x + 1]) for x in range(len(probs))]
    for agent in agents:
        if not agent.recruiter:
            rndm = r.uniform(0, 1)
            selected_agent = Agent()
            for i, cp in enumerate(cumulative_probs):
                if rndm < cp:
                    selected_agent = recruiters[i]
                    break
            agent.replace_nodes(selected_agent.chosen_nodes[:])
    e += 1

```

```

def task_sol():
    x = [node.pos[0] for node in nodes]
    y = [node.pos[1] for node in nodes]
    l = [node.idn for node in nodes]
    fig, ax = plt.subplots()
    ax.scatter(x, y)
    for i, lbl in enumerate(l):
        ax.annotate(lbl, (x[i], y[i]))
    plt.show()

```

```

main()

```

```

#####

```


Додаток Д
(обов'язковий)
Протокол перевірки БКР

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Дослідження ройового композиційного алгоритму для задач дискретної оптимізації.

Тип роботи: бакалаврська кваліфікаційна робота (БКР)
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ: кафедра автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації, ІСТ-216
(кафедра, факультет, навчальна група)

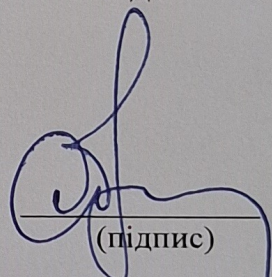
Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,63 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне):

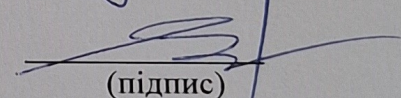
- ☒ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

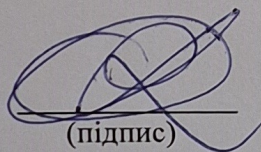
Зав. каф. АІТ, д.т.н., професор Бісікало О. В.
(прізвище, ініціали, посада)


(підпис)

Секретар кафедри АІТ Янчук Л. М.
(прізвище, ініціали, посада)


(підпис)

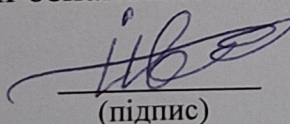
Особа, відповідальна за перевірку


(підпис)

Овчинников К. В.
(прізвище, ініціали)

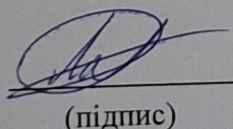
З висновком експертної комісії ознайомлений.

Керівник


(підпис)

доцент Іванов Ю. Ю.
(прізвище, ініціали, посада)

Здобувач


(підпис)

Лаврінчук О. О.
(прізвище, ініціали)