НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

«На правах рукопису»
УДК _____

До захисту допущено:
Завідувач кафедри
_____ Олександр РОЛІК
«_06_»__червня____2022 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-науковою програмою

«*Інформаційні управляючі системи та технології*»

зі спеціальності 126 «*Інформаційні системи та технології*»

на тему:

«Розроблення і дослідження метаевристик для розв'язання квадратичної задачі про призначення»

Виконав:
студент ІІ курсу, групи ІС-01мн
Фам Суан Хоанг _____

Керівник:
професор, доктор техн. наук, ст.наук. співр.
Гуляницький Леонід Федорович _____

Рецензент:
с.н.с. Інституту кібернетики
ім.В.М.Глушкова НАН України,
канд.фіз.-мат.наук, ст.наук. співр.
Ходзінський Олександр Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2022 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – другий (магістерський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-наукова програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«_06_»__червня___2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Фам Суан Хоанг

1. Тема дисертації «Розроблення і дослідження метаевристик для розв'язання квадратичної задачі про призначення», науковий керівник дисертації Гуляницький Леонід Федорович, проф., д.т.н., с.н.с., затверджений наказом по університету від «_26__»____квітня_____ 2022 р. № НС/88/2022

2. Термін подання студентом дисертації 08.06.2022

3. Об’єкт дослідження квадратична задача про призначення

4. Предмет дослідження розроблення та аналіз метаевристик для розв’язування квадратичної задачі про призначення

5. Перелік завдань, які потрібно розробити

- Аналізувати існуючі алгоритми для розв’язування квадратичної задачі про призначення.
- Виконати огляд метаевристик.
- Розробити програмний комплекс, який реалізує розглянуті метаевристичні алгоритми.

– Виконати експериментальне дослідження розроблених алгоритмів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: лінійний графік порівняння значення цільової функції кожного покоління генетичного алгоритму з оптимальним значенням, лінійний графік порівняння значення цільової функції на кожній ітерації алгоритму табу пошуку з оптимальним значенням, лінійний графік порівняння значення цільової функції при кожній температурі в алгоритмі імітаційного відпалу з оптимальним значенням, лінійний графік порівняння значення цільової функції на кожній ітерації алгоритму детермінованого локального пошуку з оптимальним значенням, лінійний графік порівняння значення цільової функції на кожній ітерації алгоритму оптимізації мурашиними колоніями з оптимальним значенням, лінійний графік відносного відхилення кожного алгоритму від оптимального значення, лінійний графік часу виконання кожного алгоритму, стовпчикова гістограма порівняння значень цільової функції та часу виконання між алгоритмами.

Орієнтовний перелік публікацій

9. Дата видачі завдання 31 січня 2022 року.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Отримання завдання	31.01.2022 – 01.02.2022	
2	Огляд постановку задачі	01.02.2022 – 05.02.2022	
3	Огляд ройових алгоритм	06.02.2022 – 25.02.2022	
4	Огляд існуючих алгоритм для розв'язування квадратичної задачі про призначення	26.02.2022 – 01.05.2022	
5	Огляд метаевристик для розв'язування квадратична задачі про призначення	02.05.2022 – 20.05.2022	
6	Розроблення програмного продукту	21.05.2022 – 05.06.2022	
7	Оформлення пояснювальної записки	01.05.2022 – 10.06.2022	
8	Захист роботи		

Студент Суан Хоанг ФАМ

Науковий керівник Леонід ГУЛЯНИЦЬКИЙ

РЕФЕРАТ

Магістерська робота на тему «Розроблення та дослідження метаевристик для розв'язання квадратичної задачі про призначення» містить 122 аркушів, 7 таблиць, 42 рисунків, 8 додатків та 28 посилань на використані джерела.

Мета дослідження. Покращення точності розв'язування квадратичної задачі про призначення.

Об'єкт дослідження. Квадратична задача про призначення (КЗП).

Предмет дослідження. Метаевристичні алгоритми для розв’язування квадратичної задачі про призначення.

Наукова новизна. В процесі дослідження було розроблено програмний комплекс із використанням метаевристичних алгоритмів ~~для~~ розв’язування квадратичної задачі про призначення.

Практичне значення отриманих результатів. Розроблений програмний комплекс може використовуватися для розв’язування різних типів квадратичних задач про призначення, які мають практичне застосування, зокрема при розміщенні електронних компонентів або при плануванні виробничих процесів.

У процесі реалізації алгоритму, використовувались такі технології для розробки програмного забезпечення: Python, Numpy, Matplotlib.

Ключові слова: КВАДРАТИЧНА ЗАДАЧА ПРО ПРИЗНАЧЕННЯ, МЕТАЕВРИСТИКИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ТАБУ ПОШУК, АЛГОРИТМ ІМІТАЦІЙНОГО ВІДПАЛУ, ДЕТЕРМІНОВАНИЙ ЛОКАЛЬНИЙ ПОШУК, ОПТИМІЗАЦІЯ МУРАШИНИМИ КОЛОНІЯМИ.

ABSTRACT

The master's thesis: "Development and research of metaheuristic approaches for solving the quadratic assignment problem" contains 122 sheets, 7 tables, 42 pictures, 8 attachments and 28 references to the used sources.

The aim of the research. Improving the accuracy of solving the quadratic assignment problem (QAP).

The object of research. Quadratic assignment problem.

The subject of research. metaheuristic algorithms for solving the quadratic assignment problem.

Scientific novelty. In the process of the research, a program was developed using metaheuristic algorithms to solve the quadratic assignment problem.

The application value. The developed program can be used to solve various types of quadratic assignment problem, which have practical applications such as placement of electronic components or transportation.

In the process of implementing the algorithm, the following technologies were used for software development: Python, Numpy, Matplotlib.

Key words: QUADRATIC ASSIGNMENT PROBLEM, METAHEURISTICS, GENETIC ALGORITHM, TABU SEARCH, ALGORITHM SIMULATED ANNEALING, DETERMINED LOCAL SEARCH, ANT COLONY OPTIMIZATION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

КЗП – квадратична задача про призначення.

ЗКО – задача комбінаторної оптимізації.

ЗНЛП – задачі неперервного лінійного програмування.

ЗБП – задачі (псевдо)булевого програмування.

ЗЦЛП – задачі лінійного цілочислового програмування.

МГіМ – метод гілок і меж.

ЛП – лінійне програмування.

GRASP – Greedy randomized adaptive search procedure.

ГА – генетичний алгоритм.

ОМК – оптимізація мурашиними колоніями.

ОРЧ – оптимізація роєм частинок.

АІВ – алгоритм імітаційного відпалу.

ВСТУП

Двадцяте століття - це час, коли відбувався стрімкий розвиток технологій людської цивілізації. Це також був період, коли відбувалися конфлікти між націями, наприклад, Перша та Друга світові війни та Холодна війна. Від цих подій, піднімалися на новий рівень важливість науки і техніки. Кожний винахід ґрунтується на наукових дослідженнях, так само, як і комп'ютерна наука.

У 1623 році, вчений Вільгельм Шикард розробив і завершив перший механічний калькулятор. Але до 1920 року, усіма обчислювальними роботами займалися в основному професіонали. Ранні дослідники того, що пізніше стане відомим як комп'ютерна наука, такі як Курт Гедель, Алонзо Черч і Алан Тюрінг, цікавилися питанням про можливість використовування машини для розв'язування складних задач замість людини, наприклад, зламати шифр Енігма у Першій світовій війні. І сьогодні комп'ютери стають все більш потужним, комп'ютерна наука застосовується у повсякденному житті, від великих проблем, як аерокосмічна техніка, до маленьких проблем, як пошук найкоротшого шляху для доставки замовлення. Одним з найвідоміших типів задач комп'ютерної науки є проблема складності NP.

Квадратична задача про призначення (КЗП) — це відома задача комбінаторної оптимізаційної, яка належить до NP-складних задач. З точки зору застосування у науці, економіці та техніці, КЗП виникає у різних сферах, включаючи, наприклад задачі розбиття графів, задача про кліку, задачу лінійного розташування, проблему балансування реактивної турбіни, задачу збірного перевезення (Less-Than-TruckLoad LTL), проєктування інтегральних мікросхем надвеликого рівня інтеграції (Very-large-scale integration – VLSI), проблеми розташування молекули, дизайн клавіатури друкарської машинки, проблем розміщення електричних компонентів, планування кампусу, дизайн

лікарні, проблеми чисельного аналізу та оптимізація схеми пам'яті в процесорах сигналів [1].

Актуальність дипломної роботи зумовлена застосуванням і розвитком індустрії інформаційних технологій у 21-у столітті. Як сказано вище, КЗП застосовується до проблеми розміщення електронних компонентів, що дуже важливо і корисно для розвитку робототехніки та аерокосмічної техніки. КЗП також застосовується до проблем, пов'язаних із транспортом та архітектурою. У 1972 році, Дікі та Хопкінс використали КЗП для вирішення проблеми, пов’язаної з розташуванням кампусу в університеті, з метою оптимізації загальної відстані, яку проходять викладачі та студенти за тиждень. Подібне повністю підходить і для компанії, який працюють в сфері транспортування, з метою мінімізувати витрати на перевезення товарів.

Метою дипломної роботи є аналіз, порівняння та дослідження метаевристичних алгоритмів розв'язування квадратичної задачі про призначення.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- Поставити задачу типу квадратичної задачі про призначення.
- Провести аналіз існуючих алгоритмів розв'язування квадратичної задачі про призначення.
- Розробити метаевристичні алгоритми розв'язування КЗП.
- Вибрати мову програмування та вид імплементації програми.
- Розробити інші алгоритми розв'язування, які не відносяться до метаевристичних для порівняння.
- Провести обчислювальний експеримент з метою дослідження ефективності розроблених алгоритмів, порівняти вихідні результати та зробити висновки.

Об'єктом дослідження є виділені метаевристики (генетичний алгоритм, оптимізація мурашиними колоніями, табуйований пошук, алгоритм імітації відпалу), які будуть використовуватися для розв'язування КЗП, та траєкторний алгоритм детермінованого локального пошуку.

Темою дослідження є розробка та аналіз результатів застосування прикладних алгоритмів для КЗП.

В першому розділі розглядається постановка задачі КЗП та існуючі алгоритми розв’язування цієї задачі.

В другому розділі, детально описуються метаевристичні алгоритми, їх обчислювальні схеми.

В третьому розділі детально описується програмне забезпечення, включаючи мову програмування, технологію, які використані для розроблення програмного продукту.

В четвертому розділі проводиться аналіз якості алгоритмів на основі отриманих результатів обчислювального експерименту за точністю та часом виконання.

ОГЛЯД ПОСТАНОВОК ТА ПІДХОДІВ ДО РОЗВ'ЯЗУВАННЯ

В розділі розглядаються постановки та підходи до розв'язування КЗП. Вперше КЗП була сформульована в 1957 р., коли Тьялінг Купманс і Мартін Йосеф Бекман інтерпретували її як математичну модель призначення декількох видів економічної діяльності або сукупності підприємств на множину місць розташування [1]. Тобто, КЗП спочатку виникла у контексті проблеми розташування об'єктів, яка й досі залишається одним із прикладів її застосування [2].

1.1. Формальна постановка та класифікації ЗКО.

Задачу комбінаторної оптимізації в загальному випадку можна продати у вигляді (1.1.1), якщо простір розв'язків X є комбінаторним простором [3].

$$x = arg \underset{x \in D \subseteq X}{ext} f(x) \qquad (1.1.1)$$

де $f(x)$ - задана цільова функція, $ext \in \{min, max\}$ – напрям оптимізації, $D \subseteq X$ – множина припустимих варіантів розв'язку [3].

Моделі комбінаторної оптимізації можна класифікувати так:

- Моделі дискретної оптимізації.
  - Лінійне програмування.
    a. ЗНЛП.
    b. ЗБП.
    c. ЗЦЛП.
  - Нелінійне програмування.
    a. Квадратичне програмування.
    b. Опукле програмування.

c. Неопукле програмування.

- Моделі власне комбінаторної оптимізації

Приклади ЗКО [3]:

- Задача комівояжера.

  Відома задача оптимізація, основна сутність цієї задачі це знайти найкоротший шлях, який проходить через задані міста по одному разу.

- Задача (псевдо)булевого лінійного програмування.
- Лінійна задача про призначення.

  Найвідоміше формулювання цієї задачі є пошуком комбінаторного набору призначення робіт працівника, щоб загальні витрати є мінімальним.

- Мінімальне кістякове дерево.
- Квадратична задача про призначення.

1.2. Огляд постановок задач оптимізації.

В [3] серед деяких прикладів моделей комбінаторної оптимізації, змістовно квадратичну задачу про призначення було сформульовано таким чином.

Нехай дано множину з n об'єктів і множину з n їх місць призначення. Відомі відстані між місцями призначення та інтенсивність потоків ресурсів між об'єктами. Задача полягає в розміщенні всіх об'єктів у різних місцях призначення таким чином, щоб сума відстаней, помножених на відповідні потоки, була мінімальною.

Математична постановка у вигляді задачі цілочислової оптимізації є такою. Задано матрицю $C = \left(c_{ij}\right)_{n*n}$, де $c_{ij} \geq 0$ – інтенсивність потоків між

об'єктами (виконавцями, що призначаються), $D = (d_{st})_{n*n}$, де d_{st} – відстань між місцями призначення (роботами). Необхідно розмістити n об'єктів на n місцях так, щоб досягти мінімуму цільової функції:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{s=1}^{n}\sum_{t=1}^{n} c_{ij}d_{st}x_{is}x_{jt} \to min \qquad (1.2.1)$$

За виконання умов:

$$\sum_{s=1}^{n} x_{is} = 1, \qquad (1.2.2)$$

$$\sum_{t=1}^{n} x_{jt} = 1. \qquad (1.2.3)$$

$$i, j = 1, \dots, n$$

де

$$x_{kl} = \begin{cases} 1, \text{якщо об}'\text{єкт } k \text{ розміщений на місці } l \text{ призначення} \\ 0 \text{ у протилежному випадку} \end{cases} \quad [3]$$

У різних постановках квадратичної задачі про призначення можливі додаткові обмежувальні умови, наприклад, зафіксовані чи заборонені позиції деяких об’єктів [3].

1.3. Підхід до розв’язування.

Алгоритми розв’язування задач комбінаторної оптимізації можна класифікувати за певними критеріями [3]. За точністю поділяють на наступні типи алгоритми: точні, наближені, евристичні.

Точні алгоритми це алгоритми, які знаходять глобальні розв’язки. Вони поділяють на 2 два під типи:

a. Загальні методи. Представник цього типу є МГіМ, метод гілок і відтинань.
b. Спеціальні методи. Ці методи використовують для специфіку задачі, що розв'язується, так що мають вузьке застосування.

Наближені алгоритми можна поділити на сім класів:

a. Послідовні алгоритми (GRASP,…).
b. Детермінований ЛП (табу-пошук, керований ЛП, стандартний ЛП, …).
c. Стохастичний ЛП (повторюваний ЛП, квантовий відпал, …).
d. Еволюційний алгоритм (генетичні, міметичні алгоритми, …).
e. Ройові алгоритми (ОМК, ОРЧ, алгоритми штучної бджолиної колоніїі).
f. Метод сканування (розсіяний пошук, перекомпоновка маршрутів,…).
g. Спеціальні методи (алгоритм Ліна-Кернігана, ε-наближені алгоритми,…).

Евристичні алгоритми це такі алгоритми, які будуються на основі правдоподібних міркувань.

За типом обчислювальної схеми поділяються на конструктивні та ітераційні.

За складністю структури поділяються на: прості, комбіновані, метаевристики, гібридні метаевристики, гіперевристики.

При розв'язуванні прикладних задачі реальних розмірностей часто спочатку знаходять допустимий розв'язок і вдосконалюють його для наближення до оптимального глобального розв'язку. Однак знайти точні

розв'язки ЗКО зазвичай не вдається, оскільки обчислювальні ресурси обмежені, а більшість проблем задач оптимізації є складними, тобто для них не існує точних алгоритмів із поліноміальною складністю. Метаевристики допомагають вирішити цю проблему, дозволяючи знаходити наближені розв'язки з прийнятною точністю.

Метаевристичний алгоритм, мета означає «поза межами», «вищий рівень», — це комбінований алгоритм, який використовується для розв’язування задачі комбінаторної оптимізації. Метаевристики поділяються на метафоричні і не метаморичні алгоритми, де метафоричні моделюють поведінку вибраного середовища. До метафоричних відносяться такі алгоритми: генетичний алгоритм, метод рою частинок, метод оптимізації водних хвиль, алгоритм імітації відпалу, гармонійний пошук, алгоритм синус-косинусів. А до неметафоричних відносяться такі алгоритми: табу-пошук, пошук по змінюваним околам (Variable Neighborhood Search), Generalized normal distribution optimization algorithm [4].

1.3.1. Artificial Bee Colony – алгоритм штучних бджолиних колоній.

Бджолиний алгоритм – це метаевристик алгоритм, який базується на основі поведінки бджіл і, був запропонований Д. Карабога в 2005 р. [11].

Загальний опис алгоритму.

В цьому алгоритму, бджоли розподіляють на 3 групи:

- Робочі бджоли, відповідають за дослідження їхніх джерел їжі та обмін інформацією для залучення бджіл-глядачів.
- Бджоли-глядачі, чекають інформації від робочих бджіл про кількість нектару на місце положення їжі. Вони чекають інформації від зайнятих бджіл про кількість нектару на позиції їжі. Бджоли-глядачі вибирають місце положення їжі, використовуючи інформацію, яку

передали робочі бджоли, і вони збирають нектар з вибраного місцеположення їжі [5].

– Бджоли розвідники шукають нові випадкові місця положення їжі. Середня кількість розвідників у рої зазвичай становить 10-15% [3].

Кожна бджола починається як бджола-розвідника і досліджує простір пошуку КЗП. Провівши деякий час у процесі дослідження, він повертається у вулик і ділиться своєю інформацією. Бджоли-глядачі спостерігають за танцями робочих бджіл і дізнаються інформацію про джерела їжі залежно від танців. Оцінюючи результати, вони повертаються до найкращих доступних харчових ресурсів. Бджоли використовують різні параметри пошуку Табу і починають більш детально експлуатувати харчові ресурси. На етапах розвідки та експлуатації, алгоритм штучних бджолиних колоній використовується методу Табу пошуку. Пошук табу – це метод оптимізації. Він використовує адаптивну пам'ять для дослідження простору пошуку КЗП. Пам'ять запобігає відвідуванню нещодавно шуканих просторів. Диверсифікація та інтенсифікація є одними з методів, які використовуються під час оптимізації.

На початку алгоритму необхідно генерувати кількість популяції, місці для пошуку нектару. Після цього процесу оптимізація повторюється за допомогою робочих бджіл, бджіл-глядачів та бджіл-розвідників. Бджоли-розвідники починають процес пошуку.

Детальніше схему алгоритму можна подати так:

Крок 1. Генерування популяції;

Крок 2. Генерування місці для пошуку нектару;

Крок 3. Створити порожню базу даних для харчових ресурсів;

Крок 4. while(дана ітерація < загальна кількість ітерації).

// Етап розвідки

Відправити бджіл розвідників на пошук нектару;

Бджоли-розвідники повертаються у вулик і танцюють();

Бджоли-глядачі оцінюють джерела їжі();

// Етап експлуатації

Перевірити раніше відвідувані харчові ресурси();

Визначити найкращі харчові ресурси();

Робочі бджоли літають до джерел їжі();

Табу пошуку для збірного нектару();

Повернутися до вулика();

Збирати розв’язок у вулику();

End while.

Крок 5. Звітувати про найкращий розв’язок.

На рисунку 1.3.1.1 відображає блок-схему алгоритму штучних бджолиних колоній.

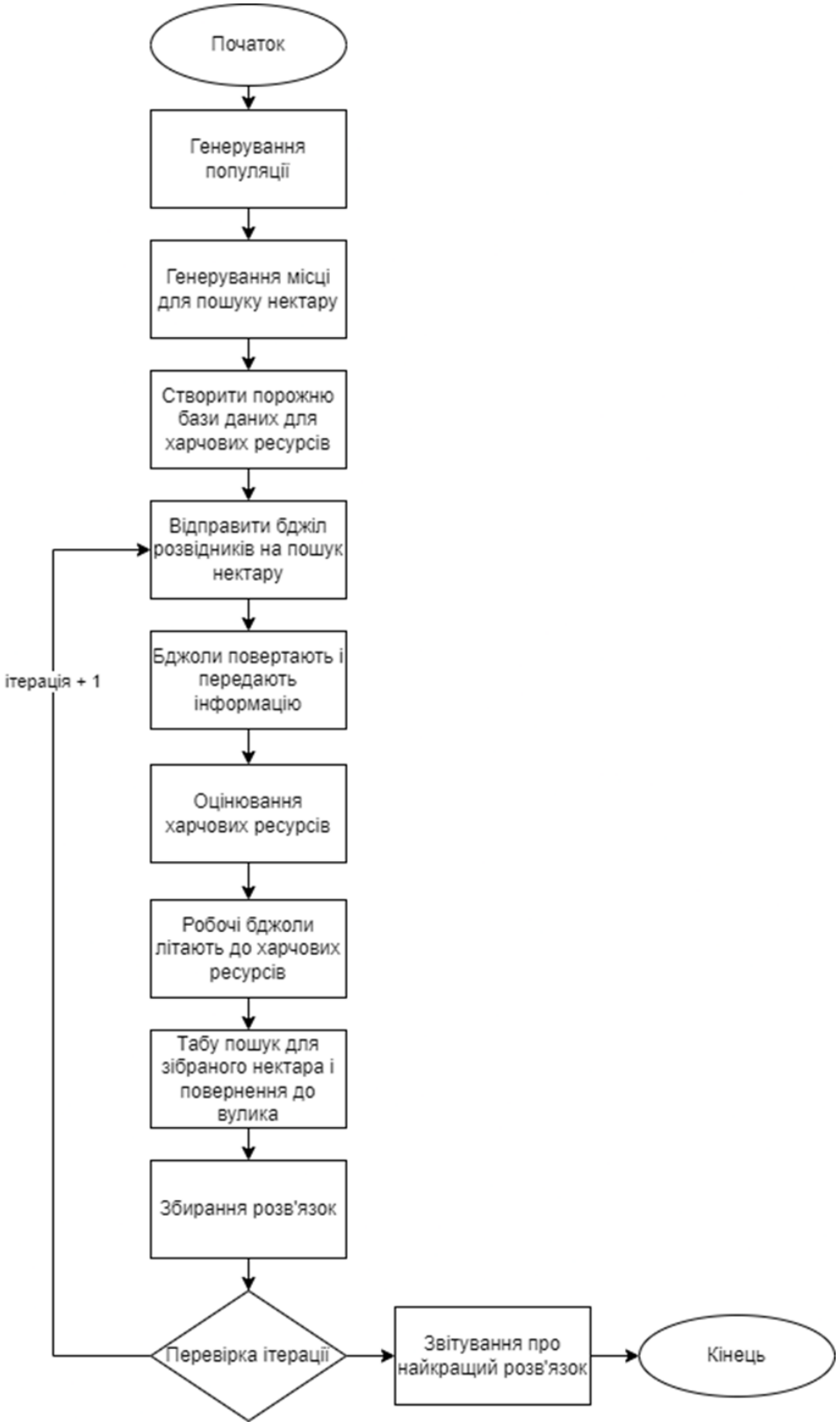


Рисунок 1.3.1.1 – Блок-схема алгоритму штучних бджолиних колоній.

1.3.2. Harmony Search – метод гармонійного пошуку.

Алгоритм гармонійного пошуку відноситься до типу алгоритмів оптимізації ройового інтелекту, він походить від творчого процесу музики і запропонований у 2001 році Зон Ву Гім, Джун Хун Кімом та Г. В. Логанатаном [6]. Усі індивіди з алгоритмом гармонійного пошуку для створення нового індивіду, не залежать від початкових умов, мають просту структуру, алгоритм досить швидко досягає прийнятних характеристик збіжності. Алгоритм гармонійного пошуку широко використовуються в оптимізації енергосистем, оптимізації економічних витрат, оптимізації маршрутизації для бездротових сенсорних мереж, архітектурному проектуванні та в інших практичних проблемах.

Незважаючи на те що, алгоритм гармонійного пошуку має широке застосування, недоліки цього алгоритму є випадковість і нестабільність [7].

Опис алгоритму:

a. Ініціалізація необхідних параметр для гармонійного пошуку.

Створення початкових параметрів як : розмір пам'ять гармонії (HMS), норма пам’яті (HMCR), коефіцієнт регулювання тону (PAR), кількість ітерації (NI) та діапазон між змінними розв’язками.

HMCR і PAR алгоритму допоможе знайти глобальний та локальний розв’язок. На ранньому етапі процесу оптимізації, з невеликим значенням PAR і з великим значенням BW сприяє підвищенню різноманітності вектора розв’язків, який може швидко знайти локальне оптимальне значення. З іншого боку, в фінальному етапі процесу оптимізації, з невеликим значенням BW і з великим значенням PAR сприяє швидкому пошуку глобального оптимального розв’язку. [7]

b. Створення випадкової пам’яті гармонії.

c. Створення допустимого розв’язка.

$$X_{new} = \left(x_{new}(1), x_{new}(2), \dots, x_{new}(j)\right),$$

де

$$x_{new}(j) = \begin{cases} x_{rnd(i),j}, r_1 < HMCR \\ x_{new}(j) \in [L_i, U_i] \end{cases} \quad (1.3.2.1)$$

де $x_{rnd(i),j}$ – елемент матриці пам’яті гармонії в j та випадковий рядок;

$x_{new}(j)$ – випадкове значення j-ої зміни. r_1 – випадкове число, яке рівномірно розподілене в діапазоні від 0 до 1 [7].

Якщо $x_{\text{new}}(j)$ є розв’язком обраних компонентів з матриці пам’яті, тоді:

$$x_{new}(j) = \begin{cases} x_{new}(j) \pm r_3 * BW, r_2 < PAR \\ x_{new}(j) \end{cases} \quad (1.3.2.2)$$

де r_2, r_3 – випадкові числа, які рівномірно розподілені від 0 до 1 [7].

d. Оновити матриці НМ.

Якщо $f(X_{new}) < f(X_{cur})$ тоді $X_{cur} = X_{new}$.

e. Регулювання $HMCR$, PAR, BW.

$HMCR$ – це ймовірність вибору одного значення з історичних значень, що зберігаються в НМ, а (1-HMCR) – це ймовірність випадкового вибору одного допустимого значення, яке не обмежується значенням, що зберігається в HM. Тим менше значення $HMCR$, тим збільшує різноманітність нових рішень. Отже, $HMCR$ має бути динамічним регулювати від найбільшого до найменшого.

Формула динамічного регулювання $HMCR$:

$$HMCR(t) = \begin{cases} HMCR(t-1) * \rho, HMCR_{max} > HCMR(t) > HMCR_{min} \\ HMCR_{min}, HMCR(t) \leq HMCR_{min} \\ HMCR_{max}, t = 0 \end{cases} \quad (1.3.2.3)$$

де $HMCR_{max}, HMCR_{min}$ – максимальне і мінімальне значення $HMCR$;

ρ – параметр, який буде контролювати спаду значення $HMCR$;

t – номер ітерації [7].

Оскільки на ранньому етапі процесу оптимізації, з невеликим значенням PAR і з великим значенням BW сприяє підвищення різноманітності вектора розв'язків, який може швидко знайти локальне оптимальне значення. З іншого боку, в фінальному етапі процесу оптимізації, з невеликим значенням BW і з великим значенням PAR сприяє швидкому пошуку глобального оптимального розв'язку. Отже значення параметру PAR має збільшувати, а значення параметра BW має зменшитись за часом.

Формула динамічного регулювання PAR і BW:

$$PAR(t) = \frac{PAR_{max} - PAR_{min}}{\sqrt{NI}} * \sqrt{t} + PAR_{min} \qquad (1.3.2.4)$$

де PAR_{max}, PAR_{min} – максимальне та мінімальне значення PAR;

NI – кількість ітерації;

t – номер ітерації [7].

$$BW(t) = BW_{min} + (BW_{max} - BW_{min}) * e^{-t} \qquad (1.3.2.5)$$

де BW_{max}, BW_{min} – максимальне та мінімальне значення BW [7].

f. Повторно виконати крок 3 і 4 доки закінчує кількості ітерації.

На рисунку 1.3.2.1 відображає блок-схему алгоритму гармонійного пошуку.

Рисунок 1.3.2.1 – Блок-схему алгоритму гармонійного пошуку.

1.3.3. Bacteria Foraging Optimization – метод імітації поведінки бактерії.

Метод імітації поведінки бактерій або Bacteria Foraging Optimization (BFO) це алгоритм оптимізації, який базується на поведінці бактерії кишкової палички, і був описаний Кевін Пассіно в 2002 і 2010 році [8]. Основна концепція цього алгоритму полягає в усуненні бактерій, які мають слабкі фуражні властивості, і підтримуєці бактерії, які мають проривні фуражні властивості, для збору поживних речовин або максимізації енергії за одиницю часу.

Причина, по якій BFO було обрано для розв’язування проблем КЗП, полягає в тому, що КЗП та BFO геометрично релевантні для пошуку розв’язків. Більше того, BFO являє собою алгоритм оптимізації, який може уникнути локально оптимальних розв’язків на етапі усунення та розповсюдження бактерії та спрямований на пошук глобального розв'язку. Таким чином, багато NP-складних задач, до яких належить і КЗП та які мають багато локальних оптимумів, доречно розв’язувати за допомогою наближених алгоритмів оптимізації, таких як BFO.

Кожна бактерія контактує між собою шляхом посилання сигналів, і бактерії переходять до наступного кроку для збору поживних речовин, якщо попередні фактори були задоволені. BFO складається з чотирьох основних частин:

a. Гемотаксичний крок.

Гемотаксис - це рух організму або сутності у відповідь на хімічний подразник. Соматичні клітини, бактерії та інші одноклітинні або багатоклітинні організми спрямовують свої рухи відповідно до певних хімічних речовин у своєму середовищі. Це важливо для того, щоб бактерії

знайшли їжу (наприклад, глюкозу), пливучи до найвищої концентрації харчових молекул, або тікаючи від отрут. Однією з властивості бактерії кишкової палички є те, що вона може переміщатися двома різними способами: плавання і перевертання. Під час плавання бактерія пливе в тому ж самому напрямку, щоб знайти поживну речовину, а під час перевертання бактерія змінює напрямок на інший.

Нехай $\theta^i(j,k,l)$ – поточна позиція i-ої бактерії, j-ого гемотаксичний крок, k-ого крок розмноження, l-ого крок усунення та розповсюдження бактерії.

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}} \qquad (1.3.3.1)$$

де C_i – кількість кроків випадкового напрямку, який визначений шляхом перевертання бактерії, $\Delta(i)$ - вектор випадкового напрямку, всі елементи якого знаходяться в діапазоні від -1 до 1 [8].

b. Роїння.

Група бактерій кишкової палички можуть розташовуватися у вигляді кільця. При цьому передаються сигнали між клітинами, які можна подати у вигляді функції:

$$J_{cc}\left(\theta, P(j,k,l)\right) = \sum_{i=1}^{S} J_{cc}\left(\theta, \theta^{i}(j,k,l)\right)$$

$$= \sum_{i=1}^{S}\left[-d_{attract}\right.$$

$$\left.* \exp\left(-w_{attract}\sum_{m=1}^{P}\left(\theta_m - \theta_m^i\right)^2\right)\right] \quad (1.3.3.2)$$

$$+ \sum_{i=1}^{S}\left[-h_{repelent}\right.$$

$$\left.* \exp\left(-w_{repelent}\sum_{m=1}^{P}\left(\theta_m - \theta_m^i\right)^2\right)\right]$$

де $J_{cc}\left(\theta, P(j,k,l)\right)$ – цільова функція витрати, на кожному кроці сума цієї цільової функції додається до основної витрати;

θ – положення p-вимірного простору пошуку i-ої бактерії, j-ого кроку гемотаксису , k-ого кроку розмноження та l-ого кроку усунення-розповсюдження бактерії;

P – кількість оптимізованих змінних в кожній бактерії i;

S – загальна кількість бактерій;

$d_{attract}, w_{attract}, h_{repellent}, w_{repellent}$ — коефіцієнти [8].

с. Розмноження.

Після етапу гемотаксису та роїння, бактерії вже мають достатні поживних речовини, щоб почати етап розмноження та усунення бактерій. Для визначення «статусу здоров'я» бактерії використовують таку функцію:

$$J_{health}^{i} = \sum_{j=1}^{N_c} J(i,j,k,l) \quad (1.3.3.3) \quad (1.3.3.3)$$

де N_c – загальна кількість гемотаксичних кроків [8]. Отже «статус здоров'я» бактерії J_{health}^{i} є сумою придатності протягом життя для i-ої бактерії, j-ого кроку гемотаксису , k-ого крок розмноження та l-ого кроку усунення-розповсюдження [8].

Після обчислення «статусу здоров'я» бактерії, результати обчислення сортуються по порядку зростання. Оскільки бактерії з меншим значення «статусу здоров'я» мають більше можливості вижити, звідси перша половина бактерій буде розмножується і замінює другу половину бактерій на основі їх найвищого «статусу здоров'я».

d. Усунення та розповсюдження бактерії.

На етапі усунення та розповсюдження, бактерії переміщуються в різні середовища, щоб уникнути загибелі бактерій і знайти оптимальний локальний розв'язок.

Опис алгоритму

Викладемо обчислювальну схему алгоритму так [8].

Крок 1. Ініціалізація параметрів:

p – розмірність простору пошуку,

S - кількість бактерій в кожному поколінні,

N_c – кількість кроків хемотаксичного етапу,

N_{re} – кількість кроків етапу розмноження,

N_{ed} – кількість кроків етапу усунення та розповсюдження,

P_{ed} – ймовірність для усунення- розповсюдження.

Крок 2. Згенерувати випадкову перестановку для кожної бактерії $i = 1, 2, \dots, S$ та обчислити витрати за допомогою функції пристосованості $J\ (i, j, k)$.

Крок 3. Сортувати витрати по порядку зростання та вибрати мінімальне значення як найкращий розв’язок на даний момент.

Крок 4. Збільшити лічильник ell кроку усунення- розповсюдження бактерії.

Крок 5. Збільшити лічильник k кроку розмноження.

Крок 6. Збільшити лічильник j кроку гемотаксису.

a) Виконати гемотактичний етап для i-ої бактерії.
b) Застосувати мутацію до кожної i-ої бактерії.
c) Обчислити цільову функцію $J\ (i, j, k)$.
d) Сортувати витрати по порядку зростання та замінити мінімальні витрати на найкращі, у випадку, якщо вони менші.
e) Повернути до пункту (b), якщо $(i + 1) \neq S$.
f) Отримати мінімальну витрату.
g) Переходити до кроку 6, якщо $(j + 1) < N_c$.

Крок 7. Сортувати витрати бактерій у порядку спадання та видалити другу половину популяції та продублювати першу половину, щоб зберегти популяцію постійною. Переходити до кроку 5, якщо $(j + 1) < N_{re}$.

Крок 8. Відновити випадкову перестановку для всіх бактерій на основі випадкового значення ймовірності P_{ed}.

Крок 9. Отримати мінімальну витрату, замінивши мінімальну витрату на найкращу, у випадку, якщо вона менша.

Крок 10. Застосували алгоритм пошуку табу, – найкращий розв’язок оптимізується за допомогою алгоритму пошуку табу і замінюється на старий розв’язок, якщо він кращий. Переходити до кроку 5, якщо $(ell + 1) < N_{ed}$.

1.3.4. Particle Swarm Optimization – Алгоритм оптимізації роєм частинок.

Алгоритм оптимізації роєм частинок або **ОРЧ** – це алгоритм, який натхнених біологічною аналогією. Цей алгоритм був запропонований Кеннеді та Ебергартом у 1995 році [9]. Основна ідея цього алгоритму полягає у тому, як вважають соціобіологи, що косяк риб або зграя птахів, які рухаються групою, «можуть отримати вигоду від досвіду всіх інших членів». Іншими словами, наприклад, коли птах летить і випадково шукає їжу, усі птахи у зграї можуть поділитися своїми відкриттями та допомогти всій зграї отримати найкраще полювання.

Основний алгоритм PSO виглядає таким чином.

a. Почати з початкового набору частинок, зазвичай випадковим чином розподілені по всьому простору розв’язків.
b. Обчислити вектор швидкості для кожної частинки з рою
c. Оновити положення кожної частинки, використовуючи її попередню позицію та оновлений вектор швидкості
d. Перейти до кроку 2 і повторити до знаходження наближених розв’язків.

На рисунку 1.3.4.1 відображає блок-схему алгоритму ОРЧ.

Рисунок 1.3.4.1 – Блок-схему алгоритму ОРЧ.

Запропоновані алгоритм PSO для розв'язання КЗП був запропонований в [9], де описана нечітка матриця, яка призначена для подання КЗП.

В цій статті нечітка матриця призначена для представлення КЗП.

Припустимо, маємо $F = \{F_1, F_2, \dots, F_n\}, L = \{L_1, L_2, \dots, L_n\}$, тоді нечітка нечітке призначення відношення від між F до та L представляється в такому вигляді:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \qquad (1.3.4.1)$$

де a_{ij} – ступінь приналежності j-ого елемента F та i-го елемента L [9].

$$a_{ij} = \mu R\left(F_j, L_i\right), i = 1,2, \dots, n, j = 1,2, \dots, n \qquad (1.3.4.2)$$

де μR – фунція приналежності [9].

Значення a_{ij} означає ступінь приналежності об'єкта F_j до місцезнаходження L_i у можливому розв’язку.

Для квадратичної задачі про призначення елементи розв’язку мають задовольнити наступні умови:

$$a_{ij} \in \{0; 1\}, i = 1,2,..,n, \qquad j = 1,2,..,n, \qquad (1.3.4.3)$$

$$\sum_{i=1}^{n} a_{ij} = 1, \qquad i = 1,2,..,n, \qquad j = 1,2,..,n, \qquad (1.3.4.4)$$

$$\sum_{j=1}^{n} a_{ij} = 1.\ i = 1,2,..,n, \qquad j = 1,2,..,n. \qquad (1.3.4.5)$$

Згідно з нечіткою матрицею, положення частинки X та швидкість частинки V можна подати в такому вигляді:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nn} \end{bmatrix}, V = \begin{bmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \vdots \\ v_{n1} & \cdots & v_{nn} \end{bmatrix} \qquad (1.3.4.6)$$

Елемент матриці має задовольнити такі умови:

$$x_{ij} \in \{0; 1\}, \qquad i = 1,2,..,n, \qquad j = 1,2,..,n, \qquad (1.3.4.7)$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad i = 1,2,..,n, \qquad j = 1,2,..,n, \qquad (1.3.4.8)$$

$$\sum_{j=1}^{n} x_{ij} = 1.\, i = 1,2,..,n, \qquad j = 1,2,..,n. \tag{1.3.4.9}$$

Так як матриця положення частинки та швидкості частинки адаптована для даної задачі, отже формула для оновлення цих матриць буде такою:

$$V(t) = w \otimes V(t-1) \oplus (c_1 r_1) \otimes \left(X^{\#}(t-1) \ominus X(t-1)\right) \oplus (c_2 r_2) \otimes \left(X^{*}(t-1) \ominus X(t-1)\right), \tag{1.3.4.10}$$

$$X(t+1) = X(t-1) \oplus V(t). \tag{1.3.4.11}$$

де $X^{\#}(t-1)$ – найкраще попереднє положення частинки на ітерації t;

$X^{*}(t-1)$ – найкраще попереднє положення найкращої частинки, сусідньої до частинки на ітерації t;

$w, (c_1 r_1), (c_2 r_2)$ – соціально-когнітивні довірчі коефіцієнти [9].

Матриця положення може порушити поставлені умови після деяких ітерацій, тому необхідно нормалізувати матрицю положення. Спочатку ми знаходимо всі від'ємні елементи в матриці та прирівнюємо їх до нуля. Якщо всі елементи в стовпці матриці дорівнюють нулю, їх потрібно повторно оцінити за допомогою ряду випадкових чисел у інтервалі [0,1]. І тоді матриця положення нормалізується таким чином [9]:

$$X_{normalize} = \begin{bmatrix} \frac{x_{11}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{12}}{\sum_{i=1}^{n} x_{i2}} & \cdots & \frac{x_{1n}}{\sum_{i=1}^{n} x_{in}} \\ \frac{x_{21}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{22}}{\sum_{i=1}^{n} x_{i2}} & \cdots & \frac{x_{2n}}{\sum_{i=1}^{n} x_{in}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{n1}}{\sum_{i=1}^{n} x_{i1}} & \frac{x_{n2}}{\sum_{i=1}^{n} x_{i1}} & \cdots & \frac{x_{nn}}{\sum_{i=1}^{n} x_{in}} \end{bmatrix} \tag{1.3.4.12}$$

Оскільки матриця положення вказує потенційний розв’язок, нечітка матриця може бути «декодована» до можливого розв’язку. Ми вибираємо елемент, який має максимальне значення в стовпці, потім позначаємо його як «1», а інші числа в стовпці та рядку встановлюються як «0» у матриці призначення. Після обробки всіх стовпців і рядків ми отримуємо розв’язок задачі призначення без порушення постановлених умов, а потім обчислюємо витрати.

Висновок до розділу.

У першому розділі розглянуто класифікацію АКО, постановку задачі КЗП та проаналізовано деякі існуючі алгоритми для розв’язування КЗП.

Квадратична задача про призначення (КЗП) була введена Купмансом і Бекманом у 1957 році як математична модель набору розташування неподільних економічних видів діяльності. КЗП застосовується до проблеми розміщення електронних компонентів, що дуже важливо для розвитку сучасних пристроїв на основі чіпів, зокрема, робототехніки та аерокосмічної техніки.

Спостерігається, що метаевристика алгоритм штучних бджолиних колоній добре працює для вирішення КЗП. Для розв’язування КЗП з великої розмірностізастосування цього алгоритму все ще залишається проблемним.

Алгоритм гармонійного пошуку – це новий інтелектуальний алгоритм оптимізації. Він має концептуально простий, досить легкий у реалізації та вимагає менших налаштувань параметрів. Однак він також має недолік випадковості, наприклад, невизначеність напрямку пошуку тощо. В цьому розділі також показаний спосіб покращення, завдяки якому обчислювальна продуктивність цього алгоритму була покращена, а також покращується точність наближення.

Метод імітації поведінки бактерій – це алгоритм оптимізації, що базується на поведінки бактерії кишкової палички, який було представлений Кевін Пассіно в 2002 і 2010 році. Запропоновано покращення даного алгоритму за допомогою методу мутації підкачки в гемотактичній частині та покращення отримуваного розв’язку за допомогою локального алгоритму пошуку табу для досягнення локального оптимуму.

Алгоритм оптимізації роєм частинок – це алгоритм, натхненний соціальною поведінкою зграй птахів або косяків риб. ОРЧ зазвичай мав кращі середні показники для задач великої розмірності. ОРЧ витрачає менше часу на призначення об’єктів на місцях.

2. РОЗРОБЛЕННЯ ПРИКЛАДНИХ АЛГОРИТМІВ.

В розділі розглядають та описують різні метаевристичні алгоритми, які будуть використовувати для розв’язання КЗП.

2.1. Генетичний алгоритм.

Багато винаходів людини були натхненні природою. Одним із прикладів є штучні нейронні мережі. Іншим прикладом є генетичні алгоритми (ГА). Генетичний алгоритм був введений Джоном Голландом на початку 1970-х років [11]. Генетичний алгоритм – це еволюційний алгоритм пошуку, створений на основі механіки природної генетики. Основна ідея ГА є моделювання процес еволюції, починаючи з початкового набору хромосомів (розв’язку) до генерування послідовних «поколінь» розв’язку.

Мета даного алгоритму є підвищенням загального значення пристосованості популяції.

Основні поняття генетичного алгоритму:

- Хромосом – ланцюг послідовності генів. Для ГА, хромосоми являється розв’язками алгоритму.
- Гени – частина розв’язку (хромосома).
- Локус – місця розміщення гена в хромосомі.
- Алель – значення окремого гена.
- Генотип – набір хромосомів особини..
- Фенотип –декодована структура або набір параметрів задачі.

Опис алгоритму.

На рисунку 2.1.1 відображає блок-схему генетичного алгоритму.

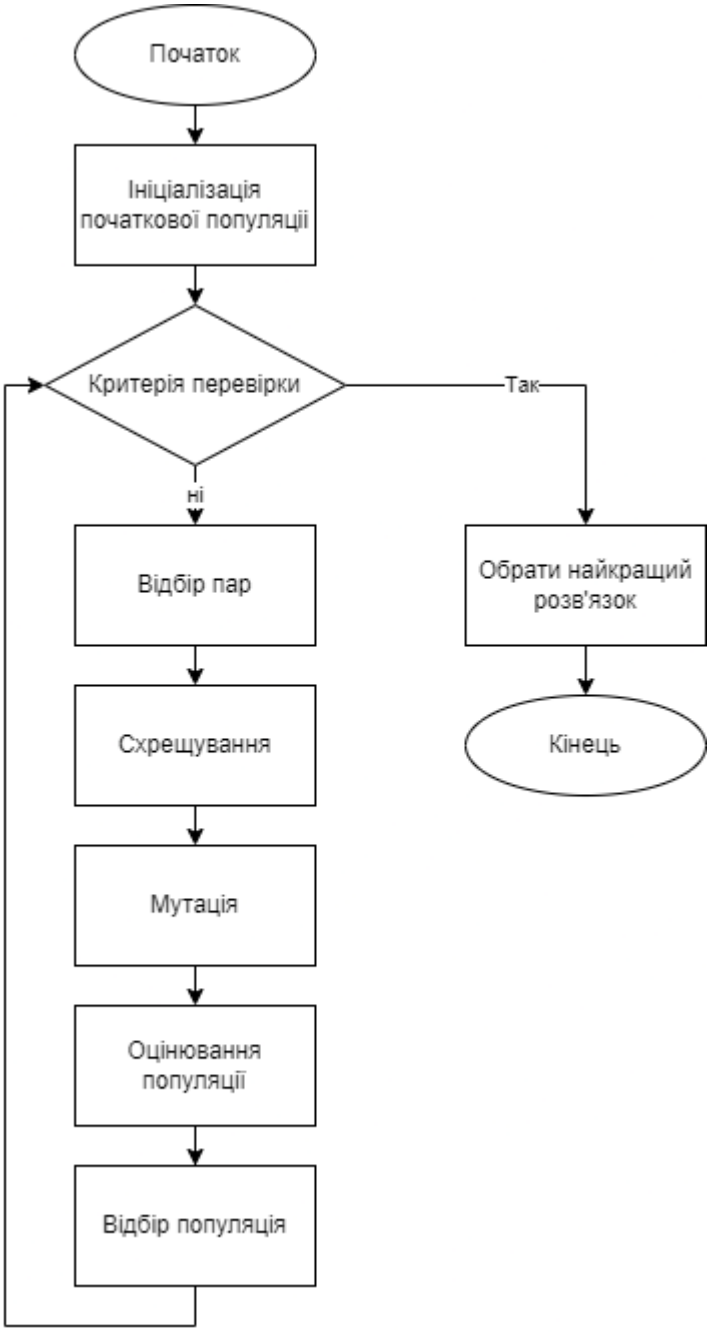


Рисунок 2.1.1 – Блок-схема генетичного алгоритму.

Етап ініціалізація початкової популяції.

Розмір популяції залежить від проблеми, але зазвичай початкова популяція формується випадковим чином, охоплюючи весь простір пошуку.

Етап відбору пар.

Відбір пар – це процес відбору батьків, які об’єднуються, щоб створити потомство для наступного покоління. Вибір батьків дуже важливий для значення наближення ГА, оскільки "хороші" батьки сприяють знаходженню кращих допустимих розв’язків.

Існує різні стратегії відбору пар:

- **Відбір пропорційно придатності.**

Є одним із найпопулярніших способів відбору батьків. При цьому кожна хромосома може стати батьком з ймовірністю, пропорційною до її пристосованості. Таким чином, хромосоми, які більш придатні, мають вищу ймовірність схрещуватися та передавати свої особливості наступному поколінню і з часом сприяють розвитку кращих хромосомів.

- **Турнірний відбір (Tounament Selection).**

Відбирають n особин із популяції випадковим чином і вибирають найкращу з них, щоб стати одним із батьків. Такий процес повторюється для вибору інших батьків. Турнірний відбір надзвичайно популярний, оскільки він може працювати навіть з негативними значеннями пристосованості.

- **Відбір на основі рангів (Rank Selection).**

Відбір рангу також працює з від’ємними значеннями пристосованості і в основному використовується, коли хромосоми в популяції мають дуже близькі значення придатності. Це призводить до того, що кожна із хромосом має приблизно однакову ймовірність бути обраною як батько і це змушує ГА робити відбір батьків у таких ситуаціях, які близькі за придатністю, що звужує можливості пошуку.

- **Випадковий вибір (Random Selection).**

У цій стратегії батьків із існуючої популяції вибирають випадково.

Етап схрещування.

Найбільш популярними способами схрещування у випадку, коли простір розв'язків містить лише булеві вектори, є одноточковий та двоточковий кросовер. Крім цього, існують ще різні методи, наприклад узагальнений кросовер та рівномірний кросовер.

- **Одноточковий кросовер.**

Основна ідея цього метода полягає в розрізанні хромосоми на дві частини в місці t. Після цього, перша частина однієї хромосоми об’єднує з другою частиною іншої хромосоми, а потім навпаки. Таким чином, створюється дві нові хромосоми.

Математичне відображення цього способу:

$$x' = \begin{cases} x_i, \text{якщо } i \leq t, \\ y_i, \text{якщо } i > t, \end{cases}' \tag{2.1.1}$$

$$y' = \begin{cases} y_i, \text{якщо } i \leq t, \\ x_i, \text{якщо } i > t. \end{cases} \tag{2.1.2}$$

На рисунку 2.1.2 ілюструє виконання процедури одноточкового кросовера.

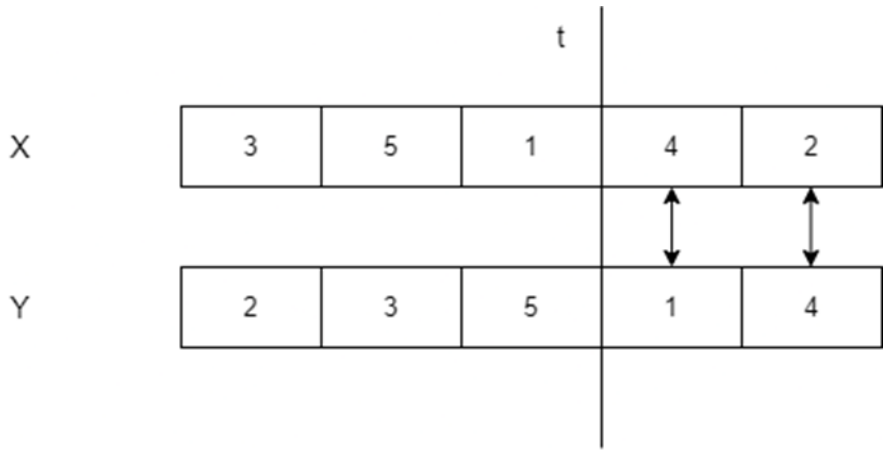


Рисунок 2.1.2 – Одноточковий кросовер.

- **Двоточковий кросовер.**

Основна ідея цього метода полягає в розрізанні хромосом на 3 частини в місці t_1, t_2. Після цього, гени з точки t_1 до t_2 першої хромосоми обмінюються

замінюються з генами з точки t_1 до t_2 другої хромосоми, а решта залишається без змін. Далі батьки міняються ролями і таким чином, створюються дві нові хромосоми.

Математичне представлення цього метода:

$$x' = \begin{cases} x_i, \text{якщо } i \notin [t_1, t_2] \\ y_i, \text{якщо } i \in [t_1, t_2] \end{cases}, \quad (2.1.3)$$

$$y' = \begin{cases} y_i, \text{якщо } i \in [t_1, t_2] \\ x_i, \text{якщо } i \notin [t_1, t_2] \end{cases}. \quad (2.1.4)$$

На рисунку 2.1.3 ілюструє виконання процедури двоточкового кросовера.

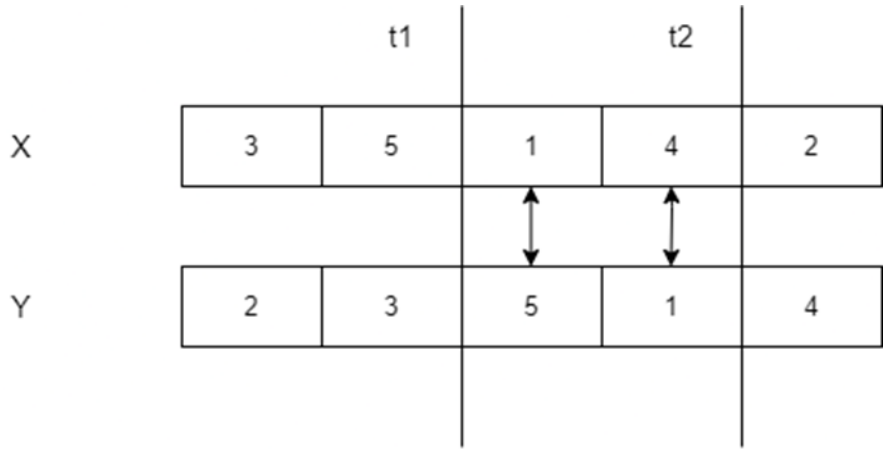


Рисунок 2.1.3 – Двоточковий кросовер.

- **Узагальнений кросовер.**

Основна ідея цього метода полягає в обміну генів між хромосомами, якщо значення $M_i = 1$, де M – маска кросовера, тобто булевий вектор такої ж розмірності, як і вектор розв'язку.

Математичне представлення цього метода:

$$x' = \begin{cases} x_i, \text{якщо } M_i \neq 1 \\ y_i, \text{якщо } M_i = 1 \end{cases}, \quad (2.1.5)$$

$$y' = \begin{cases} y_i, \text{якщо } M_i \neq 1 \\ x_i, \text{якщо } M_i = 1 \end{cases}. \quad (2.1.6)$$

На рисунку 2.1.4 ілюструє виконання процедури узагальненого кросовера.

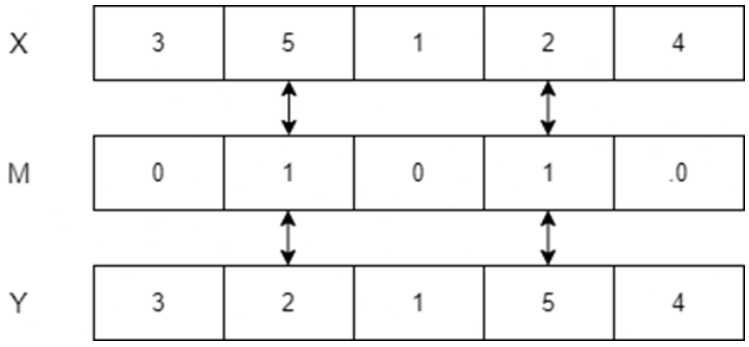


Рисунок 2.1.4 – Узагальнений кросовер.

- **Рівномірний кросовер.**

Основна ідея цього методу полягає в тому, що гени нової хромосоми з імовірністю p беруться з першої хромосоми та з імовірність імовірністю $(1 - p)$ беруться – з другої хромосоми.

Етап мутації.

Мутація – це процес випадкового порушення генетичної інформації. Відповідні процедури працюють на бітовому рівні; коли біти копіюються з поточної хромосоми в нову хромосому, існує ймовірність того, що кожен біт може стати мутованим. Ця ймовірність зазвичай є досить невеликою величиною, яка називається ймовірністю мутації p_{m}.

Застосування оператора мутації може сприяти пошуку кращого наближення до оптимуму за рахунок зміни частини генетичного коду, що буде корисно на наступних етапах. З іншого боку, це може створити «слабкий» розв’язок (хромосому), яка ніколи не буде обрана для подальших етапів.

Існує різні оператори мутації, серед них найвідоміші:

- **Інвертування біта.**

Оператор інвертування дзеркально відображає генів в хромосомі [3].

На рисунку 2.1.5 ілюструє виконання процедури інвертування біта.

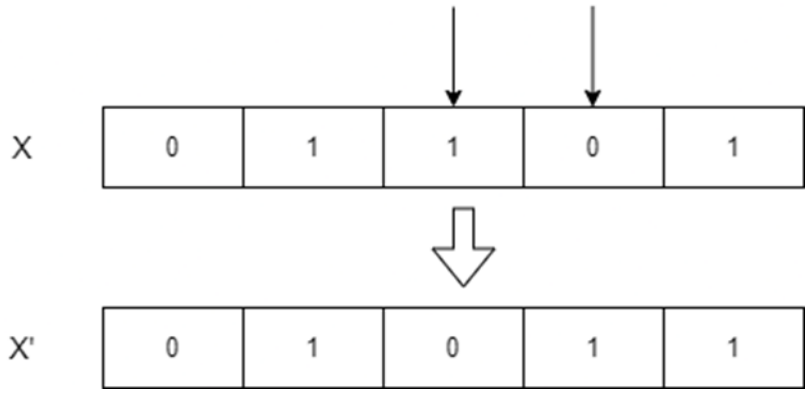


Рисунок 2.1.5 – Інвертування біта.

- **Інверсія.**

Виділити довільний фрагмент генів з від певної першої координати до другої координати в хромосомі, і повністю розвернути цей фрагмент, як показано на рисунку 2.1.6.

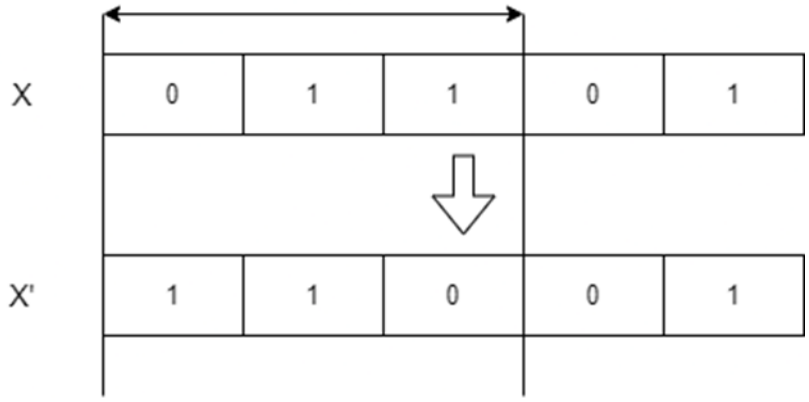


Рисунок 2.1.6 – Інверсія.

- **Транспозиція.**

Довільно вибрати два гени в хромосомі та обміняти їхні місця розташування. На рисунку 2.1.7 ілюструє виконання процедури транспозиція.

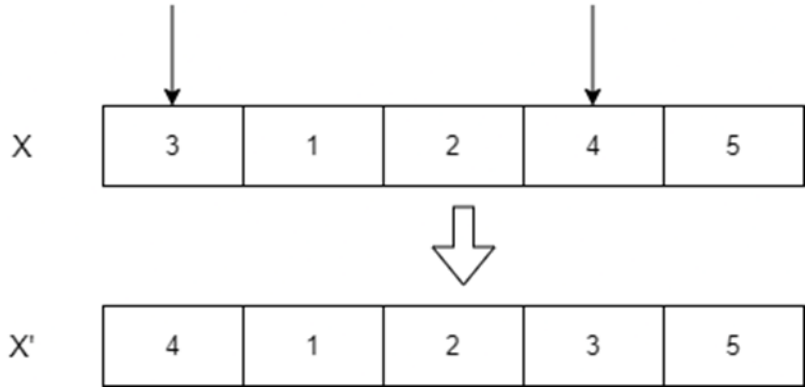


Рисунок 2.1.7 – Транспозиція.

Відбір популяції.

В [3] описують різні стратегії відбору для виживання.

- Заміна покоління. Це коли усі батьки замінюються їх потомство.
- Відбір стійких станів. Це коли кількість потомків менше ніж кількості батьків, це призводить до механізму визначення, які батьки мають бути заміненим.
- (μ, λ)-відбір. Це коли μ батьків замінюють кращими з λ потомків
- $(\mu + \lambda)$-відбір. Це коли з μ батьків та λ потомків, вибирати μ найкращих особин.

Критерії завершення.

Всі вищевказані етапи повторюються, доки не досягається критерії завершення алгоритму. Умови завершення може бути:

- Знайдений розв’язок, який задовольняє мінімальну критерію.
- Досягти максимальної кількості ітерацій.
- Досягти інше обмеження (час виконання, …).
- Послідовне покоління не дає кращий результат.

Обчислювальна схема ГА представляється таким чином.

1. Генерувати початкову популяцію особин.
2. Оцінити пристосованості кожної особини в цій

популяції.

3. Повторити цикл, доки не задовольняються умови завершення: (обмеження часу, досягнута кількість ітерацій тощо):

a. Вибирати особини, які найкраще підходять для схрещування.
b. Утворити нові особини за допомогою кросовера.
c. Мутація для потомства.
d. Оцінити пристосованість нових особини.
e. Провести селекцію, утворивши нову популяцію.

2.2. Алгоритм табу пошуку.

Табу пошук або табуйований пошук був запропонований Гловером як метод виходу із локального оптимуму [12]. Основна ідея полягає в тому, щоб заборонити деякі напрямки пошуку (переміщення) на поточній ітерації, щоб уникнути зациклення і зайві ітерації побудови неефективних розв’язків. Всі чи частина проглянутих розв'язків зберігаються в пам’яті під назвою табу-списків [3].

Табу пошук починається з початкового розв’язка, зазвичай цей розв’язок є не оптимальним. Коли процес пошуку відбувається, розв’язок поліпшується час від часу. Наступні знайдені у процесі пошуку розв'язки можуть бути не завжди кращим за попередні (найкращі знайдені запам'ятовуються), але вони сприяють виходу із локальних екстремумів, чим сприяють покращенню результатів пошуку у порівнянні із стандартним локальним пошуком.

Табу пошук при формуванні списків використовує три основні принципи [3]:

- Строге табуювання – коли в пам’яті (табу-списку) зберігаються всі досліджені розв’язки з метою заборонити потрапляння до усіх відвіданих місць.

- Фіксоване табуювання – в пам’яті зберігаються певна кількість розв’язків, тобто довжина табу-списку є обмеженою заданою константою.
- Реактивне табуювання – довжина табу-списку не є константною і може змінюватися в процесі роботи алгоритму.

Обчислювальну схему табу пошуку [3]:

1) Створити початковий допустимий розв’язок.
2) Створити пустий табу-список.
3) repeat.
   a. Пошук прийнятного варіанту $y \in O(x) \backslash T$.
   b. Додати розв’язок до табу-списку.
   c. Якщо поточний розв’язок краще ніж попереднього, тоді попередній розв’язок замінює поточний .
4) until досягти умову завершення.
5) Повертати розв’язок.

Успішне застосування табу-пошук до різноманітних класичних і практичних задач оптимізації описано в роботі Гловера, Герца і де Вера [13]; застосування табу-пошуку до класичної задачі комівояжера [14], мотивують застосувати пошуку табу для іншої класичної та більш загальної і складної проблеми КЗП.

Останнім часом, табу-пошук часто використовують як проміжна процедура для покращення локального розв’язку. Для прикладу, табу пошук був використаним для локальної оптимізації генетичного алгоритму та алгоритму штучної бджолиної колонії.

2.3. Алгоритм імітаційного відпалу.

Імітаційний відпал — це поширений метаевристичний алгоритм стохастичного локального пошуку, який використовується для розв’язування

багатьох задач комбінаторної оптимізації. З моменту введення розробки в 1983 році Кіркпатріком, Джелаттом, Векчі [15] як загального наближеного методу для дискретної оптимізації, імітаційний відпал став популярним інструментом для розв'язування дискретних і безперервних задач у широкому діапазоні сфер застосування.

У металургії та матеріалознавстві, відпал — це термічна обробка, яка змінює фізичні, а іноді й хімічні властивості матеріалу, щоб підвищити його пластичність та зменшити його твердість, що робить його більш придатним для обробки. Вона включає нагрівання матеріалу вище температури його рекристалізації, підтримання заданої температури протягом відповідного часу, а потім покрокового охолодження.

Процес відпалу має 3 етапи, які відбуваються з підвищенням температури матеріалу: відновлення, перекристалізація та зростання зерна. Звертаю увагу, зерна в металургії і матеріалознавстві це випадково розподілених, малих розмірів, які утворюють твердий метал. Довільно орієнтовані зерна контактують один з одним на поверхнях, які називаються межами зерен. Структура і розмір зерен визначають фізичні властивості твердого металу. Зерна металевого злитка можуть бути подовжені та скріплені разом шляхом прокатки для поліпшення механічних властивостей у напрямку довжини зерна. Внутрішні напруження на межах зерен можна зняти шляхом відпалу для відновлення вичерпаної пластичності в деяких сплавах або для зміцнення інших сплавів. Першим етапом є відновлення, яке призводить до розм'якшення металу шляхом видалення переважно лінійних дефектів, які називаються дислокаціями, та внутрішніх напруг, які вони викликають. Відновлення відбувається на нижчій температурній стадії всіх процесів відпалу і до появи нових недеформованих зерен. Розмір і форма зерна не змінюються. Другим етапом є рекристалізація, коли нові зерна без деформації

зароджуються і ростуть, замінюючи деформовані внутрішніми напруженнями. Якщо відпал продовжити після завершення прекристалізації, то відбувається ріст зерна (третій етап). Під час росту зерна мікроструктура починає грубіти і може призвести до втрати металом значної частини своєї початкової міцності. Однак це можна відновити за допомогою загартовування.

Основна ідея даного алгоритму базується на аналогії з процесом оптимізації та фізичним процесом відпалу [3].

В таблиці 2.3.1 показано аналоги в термінології термодинаміки та комбінаторної оптимізації, які надані в [3].

Таблиця 2.3.1 – Аналоги термінології термодинаміки та комбінаторної оптимізації.

Термодинаміка	Комбінаторна оптимізація
Фізична система	ЗКО
Узагальнена енергія	Значення цільової функції
Температура	Керуючий параметр Т
Стан	Варіант розв’язку
Метастабільний стан	Локальний розв’язок
Основний стан	Глобальний розв’язок

Відомо, що ймовірність переходу від одного стану до іншого стану обчислюється за допомогою Больцмана-Гіббса:

$$p = e^{-\frac{\Delta E}{kT}} \tag{2.3.1}$$

де ΔE – зміна узагальненої енергії;

T – температура, k – стала Больцмана [3].

На основі формули (2.3.1), ймовірність переходу від поточної точки x простору розв'язків до сусідньої точки y формалізується таким чином:

$$p = e^{-\frac{\Delta}{T}} \quad (2.3.2)$$

де $\Delta = f(y) - f(x)$;

$f(x)$, $f(y)$ – значення цільової функції попередньої та поточної температури [3].

Отже, при негативній зміні енергії, нова структура (точка, енергія і температура) буде прийнята, але якщо зміни позитивні, то новий розв'язок може бути прийнятим згідно правила Метрополіса.

Згідно правила Метрополіс процес переміщується до нової точки на основі співвідношення поточної точки та нової запропонованої випадкової точки з ймовірністю $min(1, e^{-\frac{\Delta}{T}})$, можливо, з деякими додатковими елементами для асиметричних розподілів. Якщо ця величина не менше 1, тобто, нова точка має вищу ймовірність, ніж поточна, то процес переміститься до цієї нової точки. В іншому випадку, якщо нова точка має меншу ймовірність, алгоритм переміститься до цієї точки з певною ймовірністю - на основі співвідношення. У цьому випадку алгоритм генерує випадкове значення від 0 до 1, якщо відношення менше цього значення, то він відхилить нову точку, інакше він прийме нову точку.

Обчислювальна схема для алгоритму імітаційного відпалу [15].

1) Створити початковий допустимий розв'язок та лічильник ітерацій.

2) Задати початкову температуру.

3) While не виконується умова завершення.

3.1)While не досягнута рівновага (стан рівноваги це коли кожна частина знайде своє місце і взаємодіє з іншими).

3.1.1) Генерування нових розв'язків і обчислення цільової функції.

3.1.2) Обчислення різниці між цільовими функціями.

3.1.3) Обчислення ймовірність переходу.

3.1.4) Оновлення розв’язка якщо зміна енергії більше 0, інакше, прийняти новий розв’язок згідно правила Метрополіс.

3.2) End while.

3.3) Оновлення значення лічильника і температура.

4) End while.

5) Повертати найкращий розв’язок.

На рисунку 2.3.1 відображає блок-схему алгоритму імітаційного відпалу.

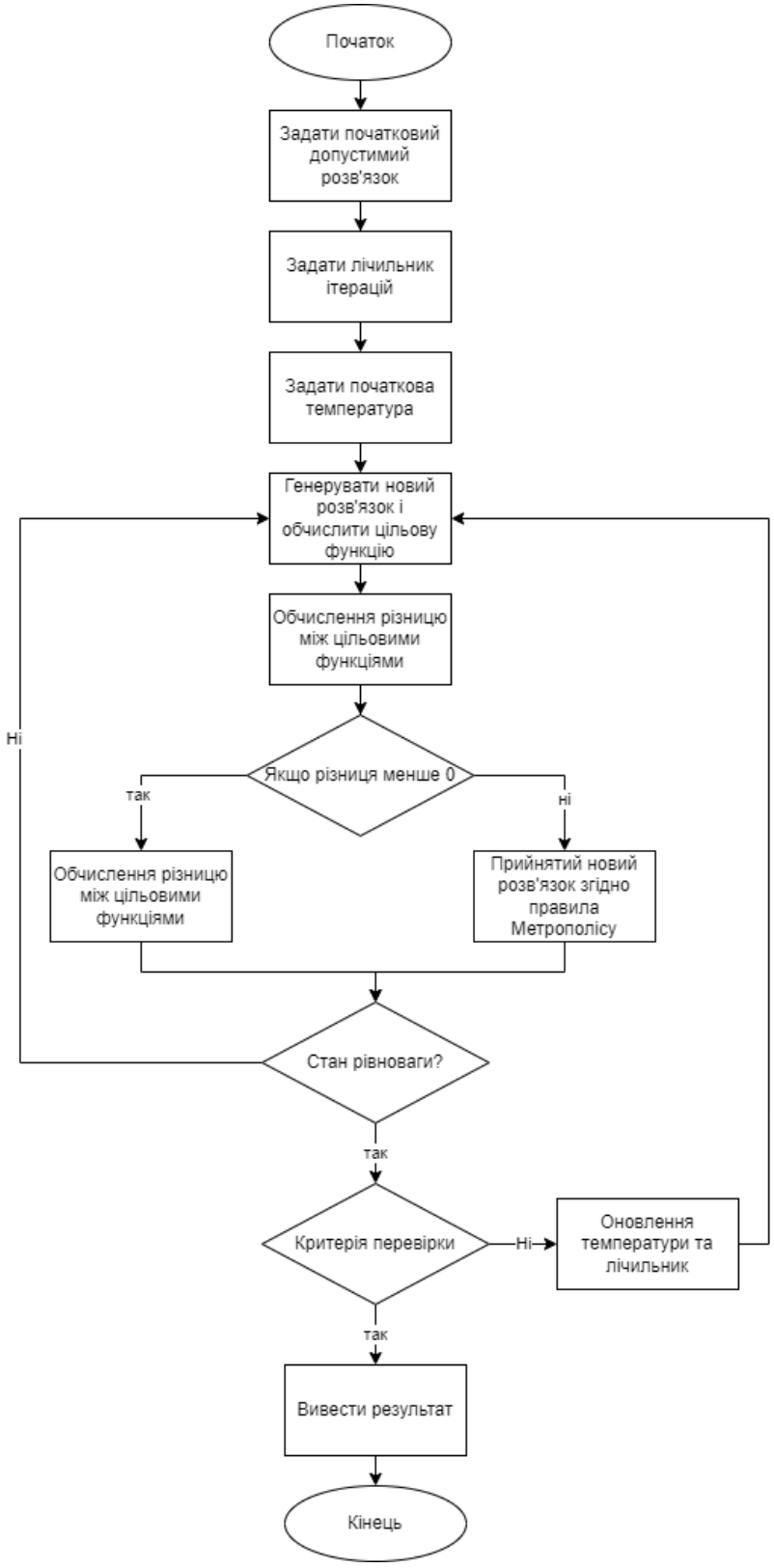


Рисунок 2.3.1 – Блок-схема алгоритму імітаційного відпалу.

2.4. Алгоритм детермінованого локального пошуку.

В [3] дає визначення детермінованого локального пошуку, як сім’я ітераційного пошуку, в якому буде частково перебирати варіантів на кожні ітерації серед сусідніх точків до поточної точки.

В деяких алгоритмах детермінованого локального пошуку було запропоновано для подолання проблеми потрапляння у локальний мінімум, що наявна в стратегії локального пошуку, використати концепцію, яка називається збуренням (Lourenço, Martin, & Stützle, 2003) [18]. Локальний пошук створено для розв'язування складних задач, зокрема задач, що належать до класів NP-складних та NP-повних [книга Гери, Джонсон, 1979]. Локальний пошук — це евристика, здатна вирішувати комбінаторні задачі NP клас, знаходячи розв’язки шляхом максимізації цільової критерії (так званого цільової функції) серед низки можливих розв’язків. Алгоритм локального пошуку досліджує та оцінює різні розв’язки (у просторі пошуку, просторі розв'язків), застосовуючи локальні зміни, доки не буде досягнуто результуючий розв’язок [17].

Базова обчислювальна схема детермінованого локального пошуку є такою:

1) Створити початковий допустимий розв’язок.

2) Чергова ітерація.

На цьому кроці, формує окіл $O(x)$ поточного варіанту розв'язку й наближено знаходити варіант $y \in O(x)$, який є супобтимальним в цьому околі. Якщо $y \neq x$, то здійснює оновлення розв’язку та починає чергову ітерація [3].

3) Завершення.

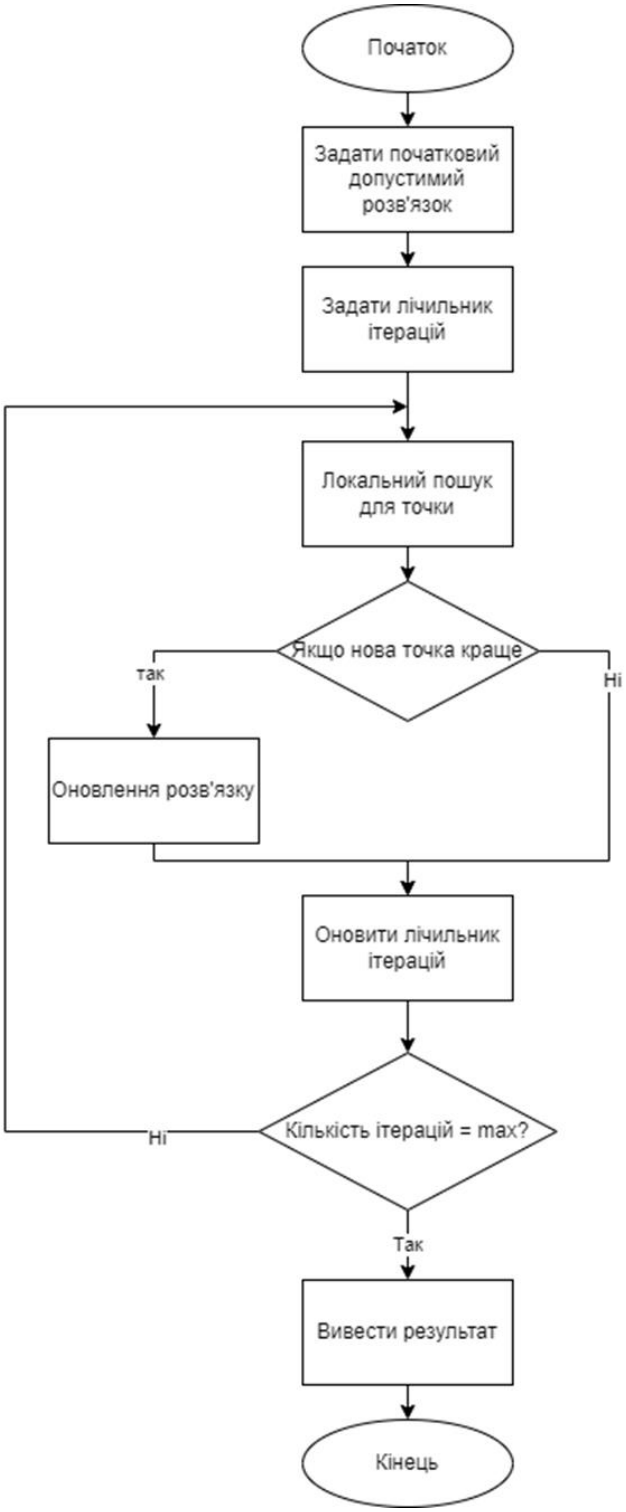
Початковий допустимий розв’язок зазвичай генерується випадково. Звичайно, малоймовірно, що такий початковий розв’язок є оптимальним. Тому далі продовжується пошук у системі введених у просторі розв'язків околів,

тому ефективність локального пошуку залежить від способу визначення таких околів. Окіл з більшим радіусом має більше шансів включати оптимальний розв’язок. З іншої сторони, пошук в околі здійснюється шляхом повного перебору точок околу, цей процес займає багато затрат часу і не гарантує, що кінцевий результат є кращим. Через це пошук в околі до першого покращення є одним з ефективних варіантів алгоритму. Іншим варіантом є використання околу зі змінним радіусом.

Критерієм завершення роботи алгоритму можуть бути такі умови:

- Відсутність точки в поточному околі, яка покращує результат.
- Вичерпання ліміту часу.
- Виконання заданої кількості ітерацій.

На рисунку 2.4.1 відображає блок-схему алгоритму детермінованого локального пошуку.



На рисунку 2.4.1 відображає блок-схему алгоритму детермінованого локального пошуку.

2.5. Алгоритм оптимізації мурашиними колоніями.

Оптимізація мурашиними колоніями (ОМК) була вперше запропонована Марко Доріго у 90-х роках у його докторській дисертації [19]. Цей алгоритм створено на основі імітації поведінки мурах при пошуці їжі для пошуку шляху між своєю колонією та джерелом їжі. Спочатку він використовувався для розв'язування відомої задачі комівояжера. Тепер він використовується для розв'язування багатьох складних задач оптимізації.

Алгоритм ОМК був створений після спостереження та дослідження поведінки колонії аргентинських мурашок [28]. Поведінка мурах керується метою пошуку їжі. Під час пошуку мурахи мандрують навколо своїх колоній. Мураха неодноразово переміщується з одного місця на інше, щоб знайти їжу. Алгоритми ОМК є багатоагентними системами, тому щоб передати інформації між агентами, система має використовувати певну форму або спосіб взаємозв'язку. Для штучних мурах, як і для справжніх через феромон, існує непрямий спосіб взаємозв'язку між агентами, який змінює навколишнє середовище, що вплине на поведінку агентів у наступних поколіннях. Рухаючись, мурахи виділяють на землю органічну сполуку, яка називається феромон. Мурахи спілкуються один з одним за допомогою феромонних шляхів. Коли мураха знаходить деяку кількість їжі, вона при поверненні природно відкладає феромони на своєму шляху. Кожна мураха може відчувати запах феромона. Отже, інші мурахи можуть відчувати відкладений феромон і йти цим шляхом. Чим вищий рівень феромонів на деякому відрізку шляху, тим більша ймовірність вибору цього шляху іншими мурахами, і тим більше мурах підуть по цьому шляху. Так на більш коротких шляхах відкладається більше феромону, то це означає, що рівень феромону на короткому (більш

оптимальному) шляху неперервно зростає, тому з певного моменту майже всі мурахи будуть рухатися по оптимальному із досліджених шляхів.

Основні компоненти обчислювальної схеми алгоритму ОМК є [3]:

- Модель задачі.
- Феромонні значення.
- Евристична інформація.
- Пам’ять (локальна та глобальна).

Обчислювальна схема ОМК:

1) Ініціалізація необхідні параметри.

2) **while** не досягнутий критерій завершення **do.**

2.1) Формування мурашина колонія.

2.2) **foreach** мурах в колонії.

2.2.1) Відправлення мурах на пошуку їжу.

2.2.2) Оновлення матриці пам’яті мураха.

2.2.3) Прокладати феромон на відвіданій дузі.

2.3) **end foreach**;

2.4) Випаровування феромона.

2.5) Оновлення результату.

2.6) Дія демона.

3) **end while.**

4) Виведення найкращого знайденого розв’язку.

Правила переходу до нової вершини.

У більшості алгоритмів ОМК імовірність переміщення мурах k з вершини i до вершини j графа задачі визначається за допомогою формули [3]:

$$p_{ij}^{k} = \frac{a_{ij}(t)}{\sum_{r \in N_i^k} a_{ir}(t)} \qquad (2.5.1)$$

де $a_{ij}(t)$ – елемент матриці мурашиних маршрутів [3].

Більш детально, формула обчислення ймовірностей є такою:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{r \in N_i^k} \tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)} \quad (2.5.2)$$

або такою

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha}(t) + \eta_{ij}^{\beta}(t)}{\sum_{r \in N_i^k} [\tau_{ij}^{\alpha}(t) + \eta_{ij}^{\beta}(t)]} \quad (2.5.3)$$

де τ_{ij} – феромонний слід, α, β – параметри алгоритму [3].

Оновлення феромону.

Метою оновлення феромонів є збільшення значень феромонів, пов’язаних з хорошими розв’язками, і зменшення тих, які пов’язані з поганими. Зазвичай це досягається (i) шляхом зменшення всіх значень феромонів за рахунок випаровування феромонів і (ii) шляхом збільшення рівнів феромонів, пов'язаних з вибраним набором "хороших" розв’язків.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta \quad (2.5.4)$$

де Δ – параметр алгоритму, t – ітерація алгоритму [3].

Випаровування феромону.

З метою зменшити ймовірність використовувати «поганий шлях» для здобуття їжі та уникнення зациклювання, випаровування феромона обчислює наступною формулою:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij} \quad (2.5.5)$$

де ρ – коефіцієнт випаровування феромону [3].

В деяких випадках використовується складніша формула, де процес оновлення та випаровування феромону відбувається одночасно, як показано в формулі 2.5.5:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + (1-\rho)\tau_0 \quad (2.5.6)$$

де τ_0 – початкова кількість феромона [3].

Дія Демона.

Після того, як розв’язки були створені, і перед оновленням значень феромонів часто можуть знадобитися певні дії, специфічні для задачі, що розв'язується. Їх часто називають діями демона і їх можна використовувати для реалізації конкретних проблемних та/або централізованих дій, які не можуть виконувати окремі мурахи. Найбільш використовувана дія демона полягає у застосуванні локального пошуку до створених розв’язків: локально оптимізовані розв’язки потім використовуються, щоб вирішити, які значення феромонів оновити [19].

Критерій завершення.

Умови завершення роботи алгоритму можуть бути такими:

- Відсутність покращення результату на одній чи декількох останніх ітерацій (змінами поколінь).
- Вичерпання заданого ліміту часу.
- Виконання максимального значення ітерацій.

Детальніша схема показана на рисунку 2.5.1.

Рисунок 2.5.1: Блок-схема алгоритму МКО.

Висновок до розділу.

В цьому розділі розглянуто п'ять різних алгоритмів, які будуть використовуватися далі для розв’язування КЗП. Розглянуто історії виникнення, головні ідеї, та визначили концепції обчислювальних схем та основні компоненти кожного алгоритму.

Генетичний алгоритм є одним з видів еволюційного алгоритму. Основна ідея цього алгоритму береться з природи. Перевага даного алгоритму є простий концепт та покращенням результату після кожного покоління. Крім цього, генетичний алгоритм дає можливість розв’язувати багатокритеріальні задачі та змішані дискретно-неперервні задачі. Але так як даний алгоритм включає в себе деякі трудомісткі процедури (наприклад, схрещування), зазвичай це збільшує часові затрати на розв'язування задачі.

Основна ідея алгоритму табу пошуку полягає в тому, щоб штрафувати переміщення, які переводять до розв'язків, що вже досліджувалися (занесені до списку табу). Основна перевага даного алгоритму є уникнення зациклювання та повернення до старих розв’язків; цей алгоритм можна використувати як допоміжний інструмент для інших алгоритмів (метаевристик).

Основна ідея алгоритму імітаційного відпалу базується на аналогії з процесом оптимізації та фізичним процесом відпалу в металургії та матеріалознавстві. Основна перевага алгоритму полягає в тому, що він приймає стани (розв'язки), які можуть не відразу покращити значення функції вартості. Прийняття таких станів обмежене деяким імовірнісним критерієм прийняття. Це дозволяє алгоритму уникнути локальних мінімумів і підвищує ймовірність знаходження оптимального (глобально мінімального) розв’язку.

Детермінований локальний пошук має багато позитивних характеристик: він відносно простий для реалізації, надійний і високоефективний. Основна ідея алгоритму полягає в тому, щоб зосередити пошук не на повному просторі розв’язку, а на меншому підпросторі – околі поточного розв’язку, що дозволяє гарантовано знаходити розв'язки, які є локально оптимальними для даної системи оптимізації. Ефективність алгоритму залежить від вибору локального пошуку, типу і обсягу околу, критерію прийняття.

Основна ідея алгоритму ОМК базується на основі поведінки колонії мурах і для розв’язування складних проблем додаються деякі штучні функції, такі як пам'ять. Головні проблеми алгоритму: вибір адекватного способу побудови моделі задачі, яка розв’язується, та складність у налаштуванні багатьох параметрів.

ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

3.1. Засоби розробки.

Перед тим як розробити програми для реалізації метаевристичних алгоритмів, було проведено загальний огляд засобів та технології, які є найбільш зручним та підходять для розробки. Засоби та технології які були використані включають наступне:

- Мова програмування Python.
- Допоміжні бібліотеки: numpy, matplotlib, random.
- Кодовий редактор Visual Studio Code.

Python — це високорівнева, інтерпретована, інтерактивна й об'єктно-орієнтована мова програмування. Python розроблений так, щоб його можна було легко читати, так як він часто використовує ключові слова англійською мовою і має менше синтаксичних конструкцій, ніж в інших мовах. Python був розроблений Гвідо ван Россумом в кінці 1989 і на початку 1990 р. у Національному науково-дослідницькому інституті математики та інформатики у Нідерландії.

Існує величезна різноманітність варіантів використання Python в різних галузях. Звичайно, перше, що спадає на думку про найпоширеніші способи використання мова програмування це створення веб-додатків, мобільних і настільних додатків, а також їх тестування. Але Python — це мова, яка слугує багатьом цілям. В основному, це області використання Python ідеально підходить для:

- Розробка веб-додатків,
- Наука про дані,
- Програмування баз даних,

- Штучний інтелект,
- Машинне навчання.

Мова дозволяє полегшити аналіз і візуалізацію даних. Вона має багато потужних наборів інструментів, які містить ефективні бібліотеки для обробки даних і це допомагає науковцям даних у виконанні складних числових обчислювальних операцій.

Основні переваги Python включають:

- **Інтерпретованість.**

Python обробляється під час виконання за допомогою інтерпретатора. Інтерпретатор – це перекладач комп'ютерної мови, який перекладає даний код рядок за рядком у машиночитані байт-коди. Якщо зустрічається будь-яка помилка, Інтерпретатор зупиняється, доки помилка не буде виправлена. Через це не потрібно компілювати програму перед її запуском.

- **Інтерактивність.**

Коли введено команду Python, а за ним натиснута клавіша Enter, якщо введені дані є вірними, результат буде виведений на екран терміналу. Це є великою перевагою в процесі налагодження програми. В інтерактивному режимі роботи Python використовується так само, як командний рядок Unix або термінал. Таким чином, якщо виникли сумніви, наприклад: чи правильний синтаксис, чи існує модуль, який імпортували, чи щось подібне. Можна виправити ці помилки за допомогою інтерактивного режиму.

- Python є **об'єктно-орієнтованою**.

Об'єктно-орієнтоване програмування (ООП) — це модель комп'ютерного програмування, яка організовує проектування програмного забезпечення навколо даних або об'єктів, а не функцій і логіки. Об'єкт можна визначити як поле даних, яке має унікальні атрибути та поведінку. ООП зосереджується на об'єктах, над якими розробники хочуть маніпулювати, а не

на логіці, яка маніпулює ці об’єкти. Такий підхід до програмування добре підходить для великих, складних програм, які активно оновлюються. Переваги ООП включають можливість повторного використання коду, масштабованість та ефективність. Python підтримує об'єктно-орієнтований стиль. Усе в Python є об’єктом. За допомогою Python ми можемо створювати класи та об’єкти, а також техніку програмування, що інкапсулює код в об'єкті.

- Python — це мова для початківців (**простота**).

Python є чудовою мовою програмування для програмістів-початківців, яка підтримує розробку широкого спектру додатків від простої обробки тексту до веб додатку та ігор. А також, вона розроблена таким чином, щоб вона була простою і легкою для читання, а також був зрозумілою і лаконічною.

- **Гнучкість**.

Незважаючи на те що, Python робить акцент на простоті та читабельності коду, а не на гнучкості, ця мова програмування всеодно має це. Python можна використовуватися в різних проектах. Це дозволяє розробникам вибирати між об’єктно-орієнтованим і процедурним програмуванням. Python також гнучкий у типі даних. Існує 5 типів даних в Pythoon: Number, String, Tuple, List і Dictionary, і кожен підтип даних відповідає одному з цих кореневих типів. В результаті, дослідницький аналіз даних стає легшим у виконанні завдяки гнучкості Python.

- **Портативність**.

Python розроблено для переносимості. Її програми підтримуються будь-якою сучасною операційною системою комп'ютера. Завдяки тому, що мова програмування - високорівнева, код Python інтерпретується, тому її можна написати для подальшої інтерпретації в Linux, Windows, Mac OS і UNIX, не вимагаючи змін. Програми Python також дозволяють реалізувати портативні графічні інтерфейси.

- **Потужний набір інструментів.**

За своєю суттю програми на Python є текстовими файлами, що містять код для інтерпретатора і записані в текстовому редакторі або інтегрованому середовищі розробки (ICP). ICP є повнофункціональними і пропонують вбудовані інструменти, такі як перевірка синтаксису, налагоджувач та система керування версіями. Текстові редактори зазвичай не включають функції IDE, але їх можна налаштувати. Python також має величезний набір сторонніх пакетів, бібліотек і фреймворків, які полегшують процес розробки. Таким чином, ці можливості роблять Python чудовим вибором для великомасштабних проектів.

- **Бази даних.**

Мова програмування Python має потужні можливості для програмування баз даних. Python підтримує різноманітні бази даних, такі як SQLite, MySQL, Oracle, Sybase, PostgreSQL тощо. Python також підтримує мову визначення даних (DDL), мову маніпулювання даними (DML) та мову запитів даних (DQL). Стандартом Python для інтерфейсів баз даних є Python DB-API.

Недоліки цієї мови програмування є:

- **Розробки мобільного додатку.**

Python це не дуже гарна мова для мобільної розробки. Вона не має вбудованих можливостей мобільної розробки, але є пакети, які можна використовувати для створення мобільних додатків, як-от Kivy, PyQt або навіть бібліотека Toga від Beeware.

- **Споживання пам'яті.**

Python не є хорошим вибором для завдань із інтенсивною пам’яттю. Завдяки гнучкості типів даних, споживання пам’яті Python також є високим. Окрім цього, Python не звільняє пам'ять назад у систему відразу після того, як знищив деякий екземпляр об'єкта. В неї є деякі пули об’єктів, які називаються

аренами, і потрібен деякий час, перш ніж вони будуть звільнені. У деяких випадках, можете страждати від фрагментації пам’яті, що також спричиняє зростання використання пам’яті.

- **Доступ до бази даних.**

У порівнянні з іншими популярними технологіями, такими як з'єднання з базами даних на Java (JDBC) і Open DataBase Connectivity (ODBC), рівень доступу до бази даних Python виявляється трохи недостатньо розвиненим і примітивним. Однак він не може застосовуватися на підприємствах, яким потрібна плавна взаємодія складних успадкованих даних.

- **Споживання енергії.**

Дослідження дослідників з Мінюського уніерситету, Португалія свідчить, що Python кращий за Perl з точки зору споживання енергії під час виконання певних завдань [20].

На рисунку 3.1.1 зображено вимірювання споживання енергії, час виконання та використання пам’яті 27 відомих мов програмування.

Table 4. Normalized global results for Energy, Time, and Memory

Total					
	Energy		Time		Mb
(c) C	1.00	(c) C	1.00	(c) Pascal	1.00
(c) Rust	1.03	(c) Rust	1.04	(c) Go	1.05
(c) C++	1.34	(c) C++	1.56	(c) C	1.17
(c) Ada	1.70	(c) Ada	1.85	(c) Fortran	1.24
(v) Java	1.98	(v) Java	1.89	(c) C++	1.34
(c) Pascal	2.14	(c) Chapel	2.14	(c) Ada	1.47
(c) Chapel	2.18	(c) Go	2.83	(c) Rust	1.54
(v) Lisp	2.27	(c) Pascal	3.02	(v) Lisp	1.92
(c) Ocaml	2.40	(c) Ocaml	3.09	(c) Haskell	2.45
(c) Fortran	2.52	(v) C#	3.14	(i) PHP	2.57
(c) Swift	2.79	(v) Lisp	3.40	(c) Swift	2.71
(c) Haskell	3.10	(c) Haskell	3.55	(i) Python	2.80
(v) C#	3.14	(c) Swift	4.20	(c) Ocaml	2.82
(c) Go	3.23	(c) Fortran	4.20	(v) C#	2.85
(i) Dart	3.83	(v) F#	6.30	(i) Hack	3.34
(v) F#	4.13	(i) JavaScript	6.52	(v) Racket	3.52
(i) JavaScript	4.45	(i) Dart	6.67	(i) Ruby	3.97
(v) Racket	7.91	(v) Racket	11.27	(c) Chapel	4.00
(i) TypeScript	21.50	(i) Hack	26.99	(v) F#	4.25
(i) Hack	24.02	(i) PHP	27.64	(i) JavaScript	4.59
(i) PHP	29.30	(v) Erlang	36.71	(i) TypeScript	4.69
(v) Erlang	42.23	(i) Jruby	43.44	(v) Java	6.01
(i) Lua	45.98	(i) TypeScript	46.20	(i) Perl	6.62
(i) Jruby	46.54	(i) Ruby	59.34	(i) Lua	6.72
(i) Ruby	69.91	(i) Perl	65.79	(v) Erlang	7.20
(i) Python	75.88	(i) Python	71.90	(i) Dart	8.64
(i) Perl	79.58	(i) Lua	82.91	(i) Jruby	19.84

Рисунок 3.1.1 - Вимірювання споживання енергії, час виконання та використання пам’яті 27 відомих мов програмування [20].

Python погано працює з точки зору споживання енергії, часу та пам’яті.

Для розробки також використовують допоміжні бібліотеки для мови програмування Python як numpy та matplotlib.

NumPy, що розшифровується як Numerical Python — це бібліотека, що складається з багатовимірних об’єктів масиву та набору підпрограм для обробки цих масивів. За допомогою NumPy можна виконувати математичні та

логічні операції над масивами. Numeric, попередник NumPy, був розроблений Джимом Хугуніном. Також був розроблений інший пакет Numarray, який має деякі додаткові функції. У 2005 році Тревіс Оліфант створив пакет NumPy, включивши функції Numarray у пакет Numeric.

Використовуючи NumPy, розробник може виконувати наступні операції:

- Математичні та логічні операції над масивами.
- Перетворення Фур'є.
- Операції, пов'язані з лінійною алгеброю. NumPy має вбудовані функції для лінійної алгебри та генерації випадкових чисел.

NumPy часто використовується разом із такими пакетами, як SciPy (Scientific Python) і Matplotlib (бібліотека графіків). Ця комбінація широко використовується як заміна MatLab, популярної платформи для технічних обчислень. Основні переваги Numpy - це масиви NumPy, які швидші та компактніші, ніж типи даних List Python. Масив споживає менше пам'яті і зручний у використанні. NumPy використовує набагато менше пам'яті для зберігання даних і забезпечує механізм визначення типів даних. Це дозволяє ще більше оптимізувати код.

Matplotlib — це кросплатформна бібліотека візуалізації даних та графічного побудови графіків для Python. Розробники можуть використовувати прикладний програмний інтерфейс matplotlib (matplotlib API) для вбудовування графіків у графічному інтерфейсі користувача додатку. Бібліотека matplotlib включають наступні переваги:

- Matplotlib підтримує різні типи графічних представлень, таких як стовпчикові діаграми, гістограми, лінійні діаграми, точкові діаграми, діаграми «стовбур — листя» тощо.
- Matplotlib можна використовувати кількома способами, наприклад код Python, iPython Shell, Jupyter Notebook.

- Matplotlib — це бібліотека для 2-вимірних графіків. Але є деякі розширення, які дозволяють використовувати для створення розширених візуалізацій, таких як 3-вимірні графіки тощо.

Програмне забезпечення буде реалізувати як консольний додаток. Консольний додаток - це комп'ютерна програма, призначена для використання лише через текстовий комп'ютерний інтерфейс, наприклад текстовий термінал, інтерфейс командного рядка деяких операційних систем. Більшість результатів програм друкується на терміналі як кінцеве призначення, цільова функція, час виконання.

Крім вище вказані засоби та технології, Visual Studio Code було використано для розробки. Visual Studio Code (відомий як VS Code) — це безкоштовний текстовий редактор з відкритим вихідним кодом від Microsoft. VS Code доступний для Windows, Linux і macOS. Хоча редактор відносно легкий, він містить деякі потужні функції, які зробили VS Code одним з найпопулярніших інструментів середовища розробки останнім часом. VS Code має багато переваги, які включають:

- **Простота.**

Від перших кроків до встановлення нових розширень все в VS Code виглядає просто й інтуїтивно зрозуміло. Завдяки розширюваній архітектурі, VS Code являє цінною альтернативою код редактор порівняно з іншими більш складними IDE. На відміну від цих IDE, VS Code вдається зробити ці функціонали компактними та зручними для користувача.

- **Розширення (Extensions).**

В магазині VS Code є тисячі розширень, нові з’являються щодня. Розширення можуть служити багатьом цілям. Від розширення, яке служить для інтерфейсу користувача, до підтримки мови програмування, налагодження, а

також інтеграції Git. Розширення дуже важливі, оскільки вони роблять VS Code таким, яким він є зараз, дуже потужним програмним забезпеченням.

- **Кросплатформенний**.

Visual Studio Code підтримує macOS, Linux і Windows, тому можна почати роботу незалежно від платформи.

- **Простота в редагування, створення, налагодження**.

Visual Studio Code є швидким редактором вихідного коду, ідеальний для повсякденного використання. З підтримкою сотень мов VS Code допомагає миттєво працювати з підсвічуванням синтаксису, узгодженням у дужках, автоматичним відступом, виділенням поля, тощо. Інтуїтивно зрозумілі комбінації клавіш, легка налаштування та допоміжні розширення, які були створені співробітників, дозволяють легко переміщатися по коду.

Visual Studio Code включає інтерактивний налагоджувач, тож ви можете переходити через вихідний код, перевіряти змінні, переглядати стеки викликів та виконувати команди на консолі.

VS Code підтримує Git, тому можна працювати з керуванням кодом, не виходячи з редактора, включно з переглядом змін, що очікують на розгляд.

3.2. Вимоги до системи.

Мінімальні вимоги до технічного забезпечення:

- Операційна система – Window 8, 64 –bit.
- Оперативна пам’ять – 4 ГБ.
- Місткість жорсткого диску – до 50 ГБ.
- Процесор – 1.6 ГГц.
- Версія Python 3.6.

Рекомендовані вимоги до технічного забезпечення:

- Операційна система – Window 11, 64 –bit.

- Оперативна пам’ять – 8 ГБ.
- Місткість жорсткого диску – більше 50 ГБ.
- Процесор – більше 1.6 ГГц.
- Версія Python 3.8.

3.3. Архітектура програмного забезпечення.

Програма складається з кількох текстових файлів, що містять основні логіки програми. Програма структурована як один основний файл верхнього рівня, а також нуль або більше додаткових файлів, відомих як модулі в Python. У програмі Python файл верхнього рівня містить основний потік, який запускають для виконання алгоритмів.

Файли модулів — це бібліотеки інструментів, які використовуються для збору компонентів, які використовуються у файлі верхнього рівня та, можливо в інших місцях. Файли верхнього рівня використовують інструменти, визначені у файлах модулів, а модулі використовують інструменти, визначені в інших модулях. У Python файл імпортується модуль, щоб отримати доступ до інструментів, які він визначає. Інструменти, визначені модулем, відомі як імена змінних його атрибутів, прикріплені до таких об’єктів, як функції.

На рисунку 3.3.1 відображено структуру програмного забезпечення.

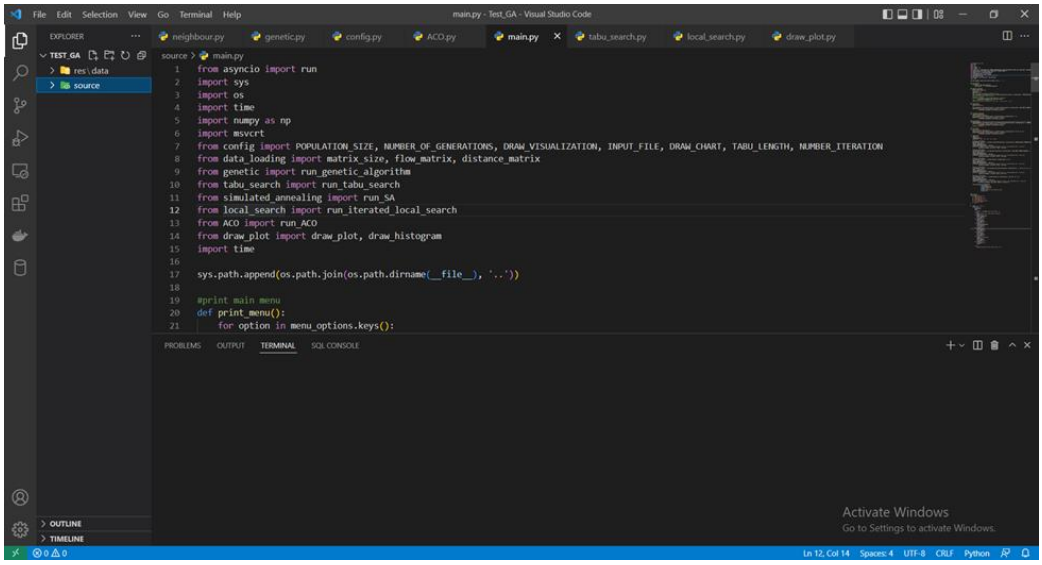


Рисунок 3.3.1 – Структура програмного забезпечення.

В папку res/data містить тестові дані для перевірки роботи метаевристичних алгоритмів.

В папку містить основні файли вихідного коду програмного забезпечення, що відображає на рисунку 3.3.2.

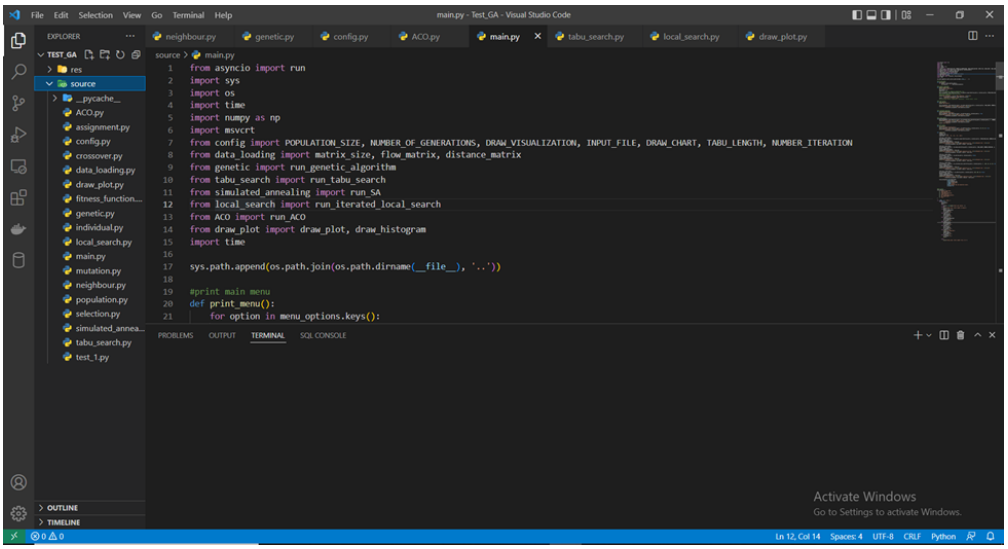


Рисунок 3.3.2 – Основні файли вихідного коду.

На рисунку 3.3.3 відображає структуру вихідного коду.

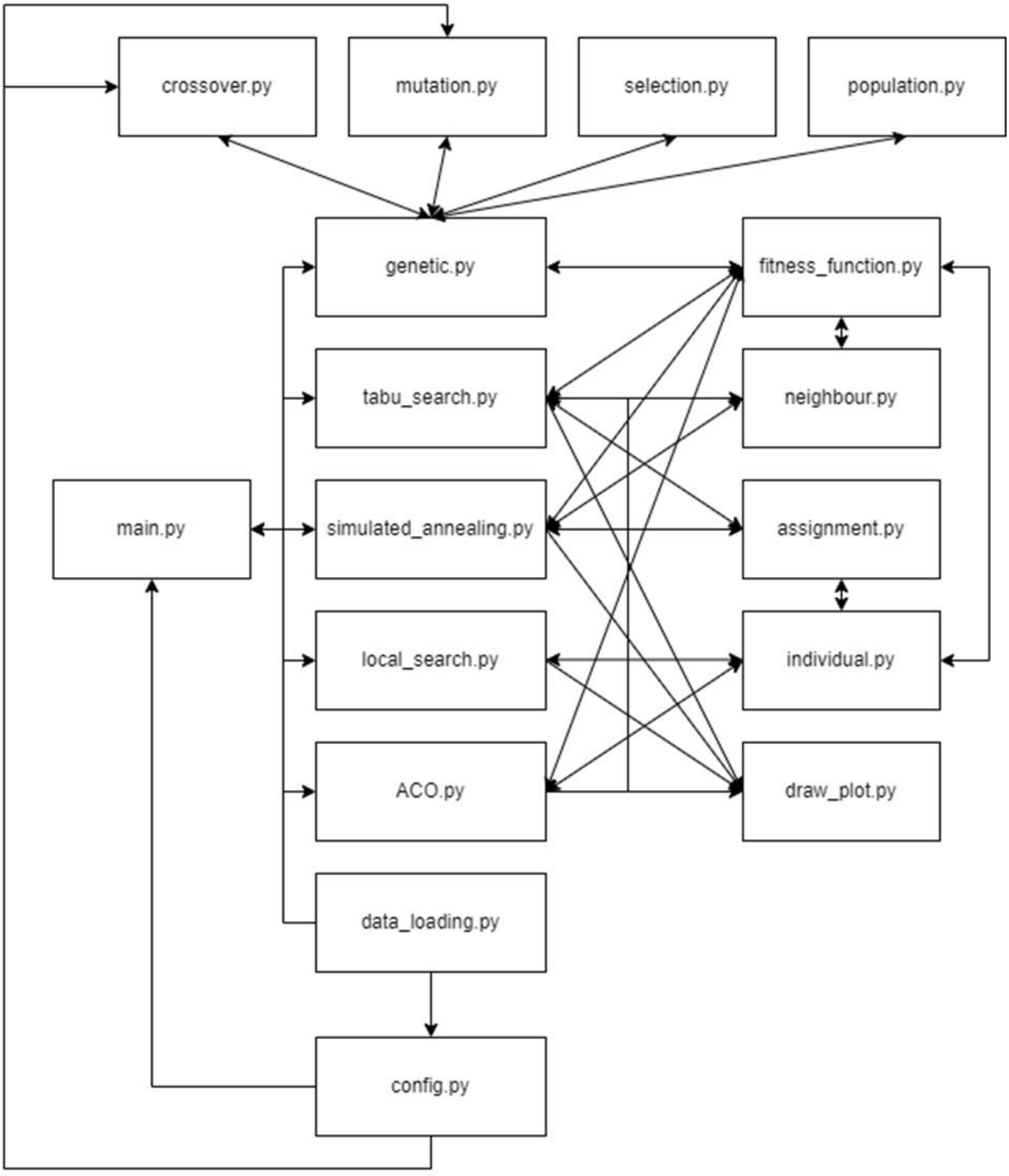


Рисунок 3.3.3 – Діаграма взаємодії між модулями програмного забезпечення

Для розробки також використовує деякі стандартні бібліотеки мови програмування Python як numpy, random, time, matplotlib. Перелік файлів

вихідного коду, їх призначення та які модулі були імпортовані подано в таблиці 3.2.1

Таблиця 3.2.1 – Перелік файлів вихідного коду

Назва файлу	Призначення	Імпортовані модулі	Основні функції
main.py	Файл верхнього рівня, який керує потік роботи програми	asyncio, sys, os time, numpy, msvcrt, config, data_loading, genetic, tabu_search, simulated_annealing, local_search, ACO, draw_plot	print_menu(), genetic_algorithm(), tabu_search(), simulated_annealing(), local_search(), ant_colony(), comparing()
ACO.py	Файл містить код призначений для виконання алгоритму оптимізації мурашиними колоніями	numpy, random, individual, fitness_function, tabu_search, assignment, draw_plot	generate_solution(), new_pheromone, update_pheromone(), update_ant(), run_ACO()
assignment.py	Файл містить код призначений для виконання роль нормалізації кінцевого результату	random	normalize_final_assignment()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
config.py	Файл містить необхідні вхідні параметри	-	-
crossover.py	Файл містить код призначений для виконання операція мутація	copy, random, config	Class Crossover • __init__() • crossover() Class BasicCrossover • __init__() • __call__() • perform_crossover() • crossover_population() • choose_chromosomes_to_crossover() • create_crossover_tuples() • crossover_chromosomes()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
data_loading.py	Файл містить код призначений для читання даних з файлу тестових даних	os, numpy, config	read_square_matrix()
draw_plot.py	Файл містить код призначений для малювання гістограми та стовпчикові діаграми	matplotlib, numpy	draw_plot(), draw_histogram()
fitness_function.py	Файл містить код призначений для обчислення цільової функції	numpy	compute_fitness_scores_list(), get_normalized_result_of_fitness_function_scores_list(), calculate_objective_vvalue(),calculate_value_swap()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
genetic.py	Файл містить код призначений для виконання генетичного алгоритму	population, selection, mutation, crossover, fitness_function, numpy, draw_plot	run_genetic_algorithm()
local_search.py	Файл містить код призначений для виконання алгоритму локального пошуку	numpy, individual, random, copy, draw_plot	generate_initial_population(), sort_population(), shuffle_population(), two_opt_improvement(), local_improvement(), run_iterated_local_search()
mutation.py	Файл містить код призначений для виконання операція мутації	random, config	class Mutation • __init__() • mutate() class BasicMutation • __init__() • __call__() • mutate_population() • mutate_chromosome()

			• generate_random_gen_indexes()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
Individual.py	Файл містить код призначений для класу розв’язку	fitness_function assignment	• __init__() • calculate_value_swap() • exchange() • calculate_objective_value() • normalize_final_assignment __str__()
local_search.py	Файл містить код призначений для виконання алгоритму локального пошуку	numpy, individual, random, copy, draw_plot	generate_initial_population(), sort_population(), shuffle_population(), two_opt_improvement(), local_improvement(), run_iterated_local_search()
mutation.py	Файл містить код призначений для	random, config	class Mutation • __init__() • mutate() class BasicMutation

	виконання операція мутації		• __init__() • __call__() • mutate_population() • mutate_chromosome() • generate_random_gen_indexes()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
population.py	Файл містить код призначений для генерації популяції	random	generate_random_population()
neighbor.py	Файл містить код призначений для класу сусідів	fitness_function, numpy	__init__(), get_assignments(),get_tabu_identifiers(), set_assignments(), set_tabu_identifiers(), calculate_objective_value()
selection.py	Файл містить код призначений для	numpy, random	class Selection • __init__() • select() class RouletteSelection

	виконання операція вибір батьків		• __init__() • __call__() • selection() • select_chromosome() class TournamentSelection • __init__() • __call__() • tournament_selection ()

Продовження таблиці 3.2.1.

Назва файлу	Призначення	Імпортовані модулі	Основні функції
simulated_-annealing.py	Файл містить код призначений для виконання алгоритму імітаційного відпалу	numpy, random, copy, neighbor, fitness_function, assignment, draw_plot	generate_neighbors(), accProb(), run_SA()
tabu_search.py	Файл містить код призначений для	copy, neighbor, collections, numpy, fitness_function,	generate_neighbors(), is_in_tabu_list(), run_tabu_search()

	виконання алгоритму табу пошуку	assignment, draw_plot	

3.4. Алгоритми.

3.4.1. Генетичний алгоритм.

Генетичний алгоритм програми виконує згідно з блок-схемою, яка відображена на рисунку 2.1.1.

- **Вибір батьків.**

Наступний етап є вибором батьків для схрещування. Основний метод було використано для програми є турнірним відбором. Турнірний відбір також є стратегією відбору, яка відбирає осіб на основі їхніх значень пристосованості. Основна ідея цієї стратегії полягає в тому, щоб вибрати хромосому з найвищим показником пристосованості з певної кількості особин в популяції у наступне покоління. У відборі турніру немає арифметичного обчислення на основі значення придатності, а лише порівняння між особами за значенням пристосованості.

Згідно з результатом дослідження [21], продуктивність турнірного відбору краще ніж відбору колеса рулетки.

На рисунку 3.4.1.1 відображає код реалізації турнірного відбору.

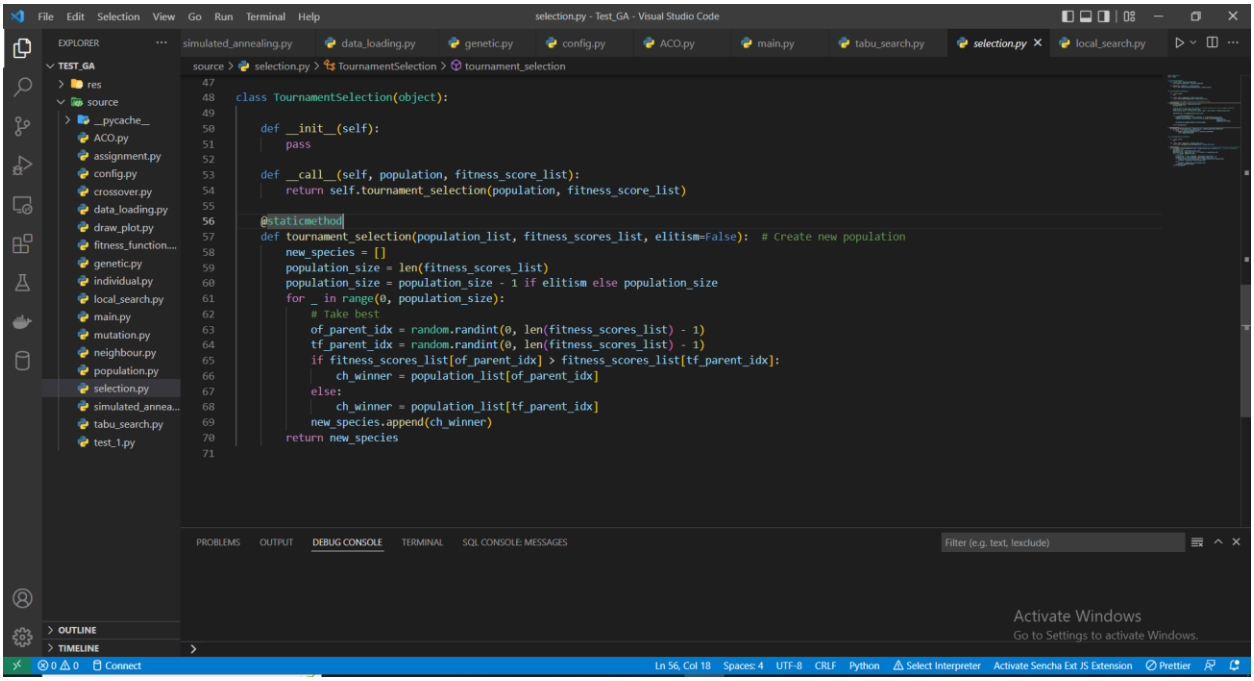


Рисунок 3.4.1.1 – Код реалізації турнірного відбору.

- **Операція кросовер.**

В програмі було використано одноточковий кросовер – класичний та найпростіший оператор схрещування. Основна ідея цього метода полягає в розрізанні хромосоми на дві частини в місці t. Після цього, перша частина однієї хромосоми об’єднує з другою частиною іншої хромосоми, а потім навпаки. Таким чином, створюється дві нові хромосоми.

Імовірність кросовер – це ймовірність того, що кросовер відбудеться при певному спарюванні; тобто не всі спарювання повинні розмножуватися за допомогою кросовера. Зазвичай в генетичних алгоритмах, ймовірність кросовера знаходиться на проміжці від 0.6 до 0.9. Чим вища ймовірність кросовера, тим більше можливості досліджувати простір розв’язку і надає алгоритму можливість знаходитися глобального оптимуму. В програмі, ймовірність кросовер була налаштована значення 0.9.

На рисунку 3.4.1.2 відображає код реалізації одноточкового кросовера.

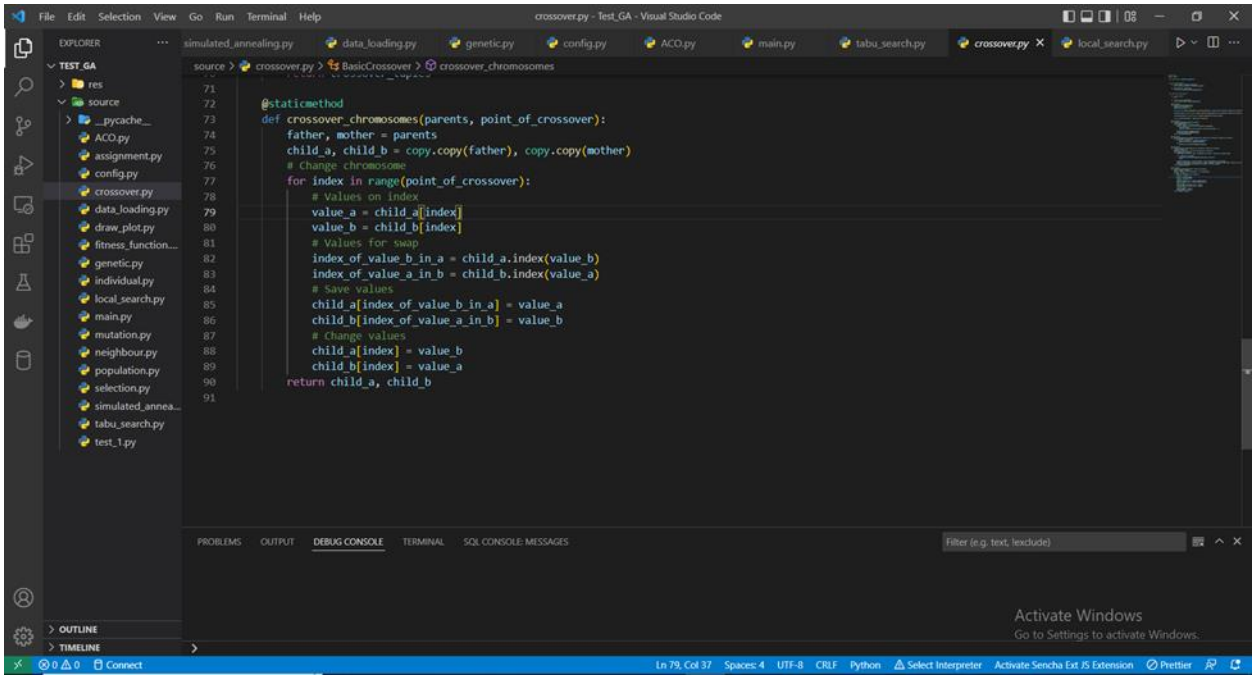


Рисунок 3.4.1.2 – Код реалізації одноточкового кросовера.

- **Операція мутація.**

Основним оператором мутації, яке було використано у програмі є транспозиція. Основна ідея цього оператора це довільно вибрати два гени в хромосомі та обміняти їхні місця розташування. Згідно з результатом дослідження [22], оператор транспозиція надає найкращий результат час виконання, а також забезпечує кінцевий результат з малим відхиленням.

Імовірність мутації — це ймовірність того що випадкові хромосоми в популяції будуть перетворені в щось інше.

В генетичному алгоритмі, оператори мутації в основному використовуються для дослідження, а оператори кросовер широко використовуються, щоб підштовхнути популяцію до того, щоб знайти оптимальний розв’язок (експлуатація). Отже, кросовер намагається зблизитися до певної точки оптимуму даному області, мутація робить все можливе, щоб уникнути зближення і досліджувати більше областей. Низьке значення ймовірності мутації надає можливість знайти локальний оптимум. Зазвичай в

генетичних алгоритмах, ймовірність кросовера знаходиться на проміжці від 0.6 до 0.9, а ймовірність мутацій — набагато меншим (близько до нуля, наприклад 0.001, 0.005). В програмі, ймовірність кросовер була налаштована значення 0.05.

На рисунку 3.4.1.3 відображає код реалізації оператор мутації транспозиції.

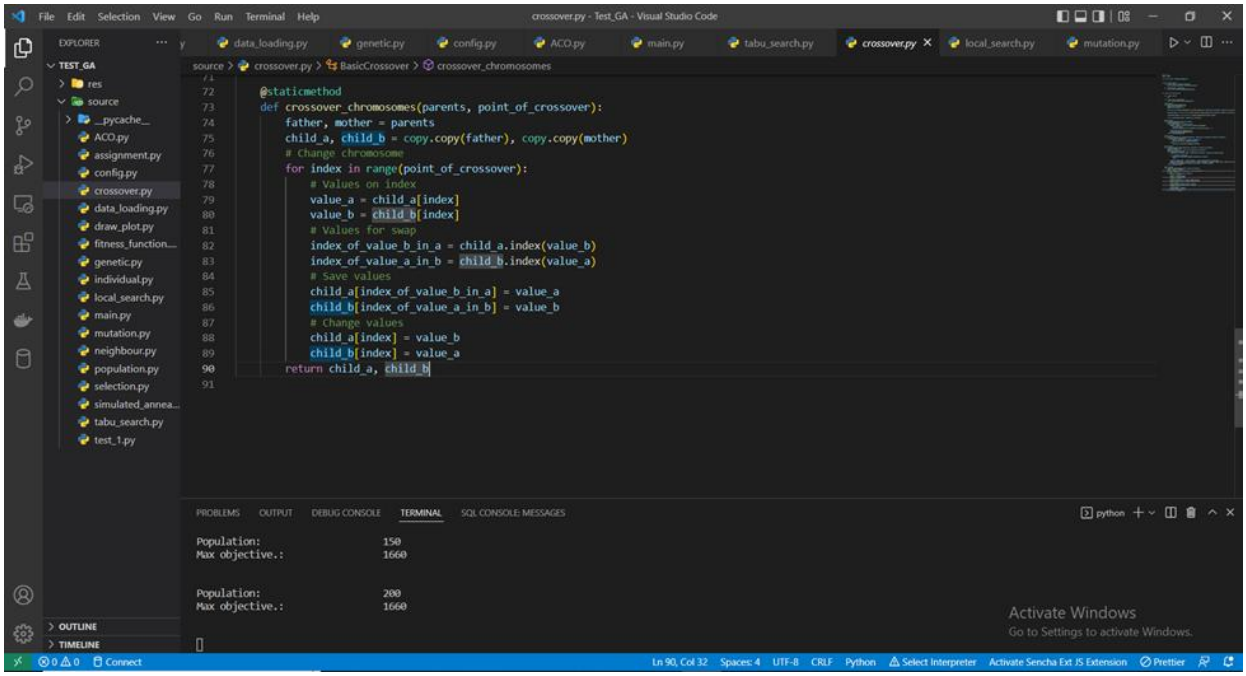


Рисунок 3.4.1.3 – Код реалізації оператора мутації транспозиції.

- **Кількість популяцій та поколінь.**

Розмір популяції є важливим параметром, який безпосередньо впливає на можливість пошуку оптимального розв’язку в просторі пошуку. Багато дослідників виявили, що наявність великої кількості популяції призводить до точності отримання оптимального розв’язку. Але наявність великої кількості населення не буде гарною ідеєю, якщо простір для пошуку невеликий, а також велика кількість популяції збільшує час виконання, а також зменшує продуктивність програми. Як правило, розмір популяції залежить від кількості генів. Отже, для 9 генів потрібно 16 хромосом, для 16 генів потрібно 32

хромосоми. Отже, кількість хромосом в 1.5 до 2 рази більше ніж кількості генів хромосому. В багатьох дослідженнях [23][24], для задачі з розміром менше 50, налаштоване значення кількості популяції дорівнює 100.

Кількість поколінь це кількість ітерацій циклу повторення послідовності операцій вибір батьків, кросовер, мутація. В багатьох дослідженнях, мінімальна кількість поколінь є 100 поколінь. В дослідження [22], використовували 200 поколінь як вхідний параметр.

3.4.2. Алгоритм табу пошук.

Основні параметри для алгоритму табу пошуку є розмір списку табу та кількість ітерацій. Розмір списку табу безпосередньо впливає на якість розв'язку. Якщо розмір занадто малий, відбудеться циклічне переміщення, а якщо він занадто великий, це обмежить пошук у «перспективних областях». На рисунку 3.4.2.1 відображає графік залежності між розміром списку табу та значенням цільової функції

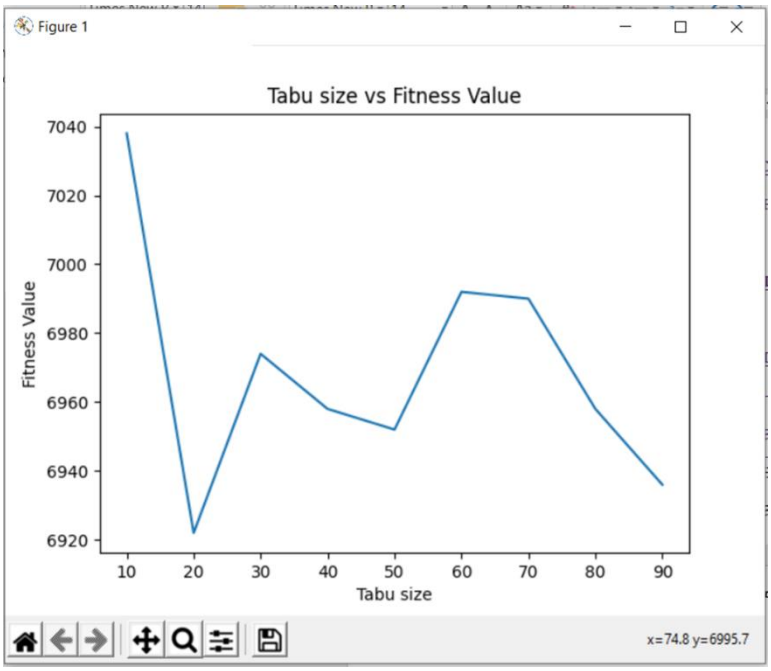


Рисунок 3.4.2.1 – Графік залежності між розміром списку табу та значенням цільової функції.

В програмі, розмір списку табу було налаштовано значення від 20 до 50, залежно від розміру задачі, а кількість ітерацій дорівнює 200.

3.4.3. Алгоритм імітаційного відпалу.

Основні вхідні параметрі являються початкова, кінцева температура та коефіцієнт охолодження.

Коефіцієнт охолодження α рекомендовано вибирати в проміжці від 0.5 до 0.99. В програмі, цей параметр був налаштовано значення 0.8. Для температурного розклад використовували формулу:

$$T_{i+1} = \alpha T_i \qquad (3.4.3.1)$$

де α – коефіцієнт охолодження [3].

За допомогою формули 3.4.3.1, температурний розклад відбувається рівномірно, тобто температура змінюється рівномірно [3].

Існує декілька способів обчислення початкової температури, в роботі [25] запропонували наступну формулу:

$$T_0 = \Delta E_{max} \qquad (3.4.3.2)$$

де T_0 – початкова температура, ΔE_max – різниця цільової функції між найкращим і найгіршим сусідом.

Незважаючи на те що, за допомогою формули 3.4.2, значення початкової температури буде динамічне для кожної задачі, але так як початковий розв’язок згенерує випадково, різниця цільової функції може бути дуже мала і кінцевий результат буде не оптимальним. Отже, для початкової температури, це значення задається власноруч, тобто значення буде фіксованим. Перевага цього способу це кінцевий результат, який буде більш оптимальний, оскільки користувач керує початковою температурою. З іншої сторони, для задачі з маленьким розміром, програма буде витрачати більше час для виконання.

3.4.4. Алгоритм детермінованого локального пошуку.

Продуктивність детермінованого локального пошуку залежить від вибору чотирьох компонентах: початкового розв'язка, локального пошука ,критерії прийнятності та процедурі збурення.

Програма реалізує алгоритм детермінований локальний пошук згідно з блок-схемою, що відображає на рисунку 2.4.1. В програмі, початковий розв'язок генерує випадково. Для локального пошуку, реалізує найпростіший алгоритм локального пошуку з назвою ітераційне вдосконалення або 2-opt. Найкраще покращення 2-opt для КЗП отримує переваги від того факту, що ефект від обміну двома призначеннями можна швидко розрахувати, використовуючи інформацію попередніх ітерацій [26]. Критерії прийнятності це коли закінчуються кількість ітерацій, а процедура збурення відбувається коли немає жодне покращення в поточному області.

3.4.5.Алгоритм оптимізації мурашиними колоніями.

В алгоритмі, ймовірність переміщення мурахів обчислюється за допомогою формули 2.5.1. Оскільки Дія Демона є необов'язковою процедурою, програма реалізує алгоритм оптимізація мурашиними колоніями згідно з блок-схемою, яка відображає на рисунку 2.5.1 без Дії Демона. Окрім цього, для покращення результату, додали табу пошук в кінці кожної ітерації. Недоліки цієї операції являється збільшення час виконання програми.

На рисунку 3.4.5.1 та 3.4.5.2 відображає значення цільової функції без табу пошуком та без табу пошуку.

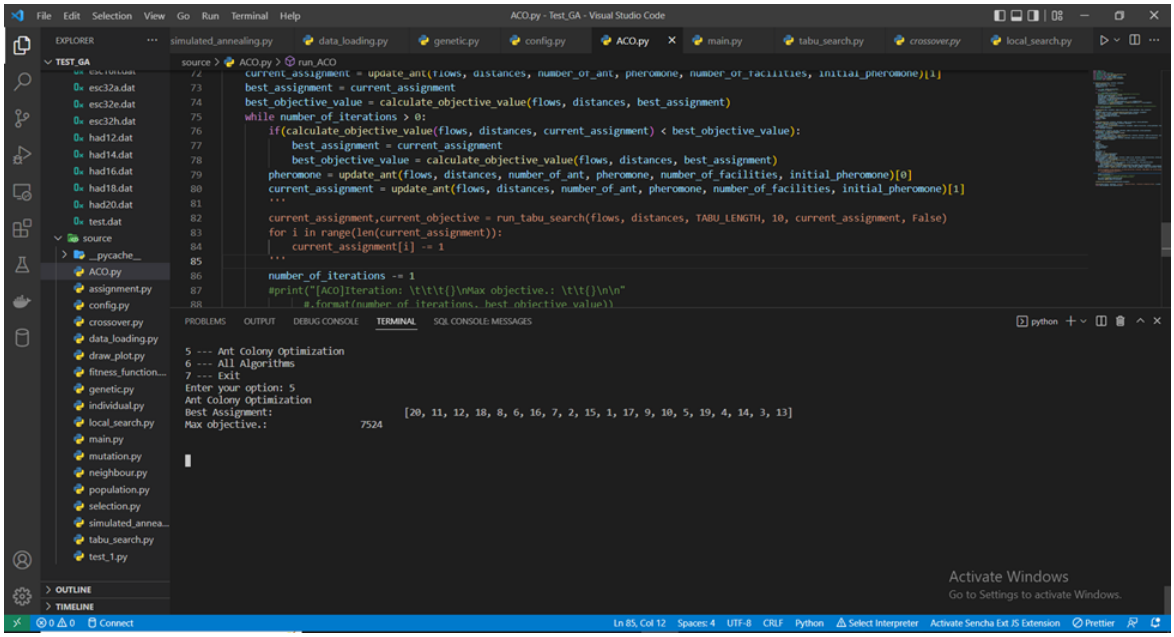


Рисунок 3.4.5.1 – Значення цільової функції без табу пошуку.

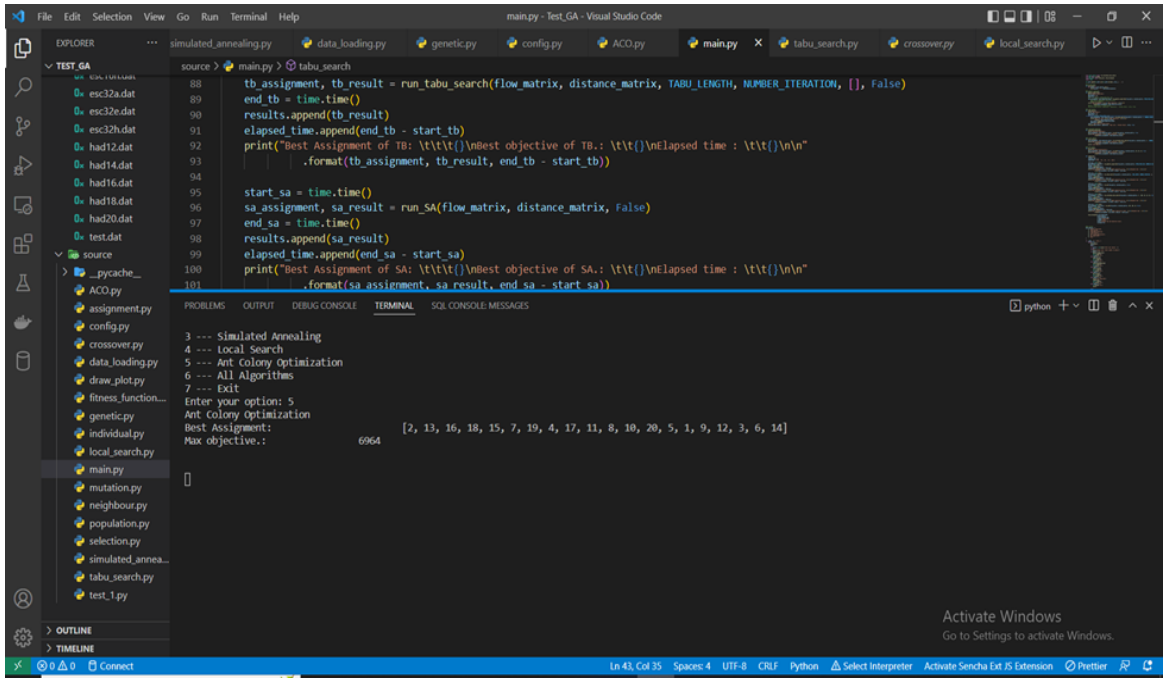


Рисунок 3.4.5.2 – Значення цільової функції з табу пошуком.

3.5. Специфікація функцій.

Програми реалізації алгоритмів для розв’язування КЗП містить дуже багато функції, тому потрібно переглянути основні функції, що виконує ключові ролі в програмі. В таблиці 3.5.1 наведено основні функції програми.

Таблиця 3.5.1 – специфікація основних функцій.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
genetic_algorithm	main.py	-	-	Виклик генетичного алгоритму.
tabu_search	main.py	-	-	Виклик алгоритму табу пошуку.
simulated_annea-ling	main.py	-	-	Виклик алгоритму імітацій-ного відпалу.
simulated_annea-ling	main.py	-	-	Виклик алгоритму імітацій-ного відпалу.

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
local_search	main.py	-	-	Виклик алгоритму локального пошуку.
ant_colony	main.py	-	-	Виклик алгоритму оптимізації мурашиними колоніями.
comparing	main.py	-	-	Виклик всіх алгоритмів з метою порівняння значення цільової функції та часу виконання.
generate_solu-tion	ACO.py	• flows. • distan-ces. • phero-mone.	• Матриця потік між місцями. • Матриця відстаней.	Генерація нових розв’язків.

			• Значення феромону.	

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
new_phero-mone	ACO.py	• index. • assignment. • initial_phe-romone. • flows. • distances.	• Індекс. • Призна-чення. • Початкове значення феромону. • Матриця потоків. • Матриця відстаней.	Обчислення значення феромону.
update_phe-romone	ACO.py	• flows. • distances. • pheromone. • initial_phe-romone,	• Матриця потоків. • Матриця відстаней. • Поточний феромон. • Початкове значення феромону.	Оновлення феромону.

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
update_ant	ACO.py	• flows. • distances. • ant_count. • pheromone. • initial_phero-mone.	• Матриця потоків. • Матриця відстаней. • Кількість мурахів. • Початкове значення феромону.	Оновлення мурахів.
run_ACO	ACO.py	• flows. • distances. • number_of ant. • number_of-iteration • initial_phero-mone • is_draw	• Матриця потоків. • Матриця відстаней. • Кількість мурахів. • Кількість ітерацій. • Початкове значення феромону. • Показник, чи	Запуск алгоритму оптимізації мурашиними колоніями.

			малювати графік.	

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
normalize-final_assign-ments	assignment.py	assignments	Призна-чення.	Приведення кінцевий результат до правильного вигляду.
crossover_ chromosomes	crossover.py	• parents • point_of cross-over	• Батьки. • Точка розрізу.	Функція виконує одноточ-ковий кросовер.
choose_chromo-somes_to_cross-over	crossover.py	• popula-tion. • species_-not_-crosso-vered • species_-	• Попу-ляція. • Хросом не для кросове р. • Хросом для	Визначення хромосомів для кросовера.

		to_cross-over	кросовер.	

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
read_square-_matrix	data_loading.py	matrix_-file	Назва файла з даними	Читання даних.
draw_plot	draw_plot.py	• y_axis • x_axis • xlabel • ylabel • title • is_draw	• Значення осі Y. • Значення осі X. • Текст осі X. • Текст осі Y. • Заголовок. • Показник, чи малювати графік.	Малювання лінійний графік
draw_histo-gram	draw_plot.py	• y_axis • x_axis • z_axis • xlabel	• Значення першого стовпчика.	Малювання стовпчикові діаграми.

		• ylabel • title • is_draw	• Значення осі X. • Значення другого стовпчика. • Текст осі X. • Текст осі Y. • Заголовок. • Показник, чи малювати графік.	

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
compute_-fitness-_scores_list	fitness_-function.py	• population • distance_-matrix • flow_matrix	• Популяція • Матриця відстаней. • Матриця потоків	Обчислення значення пристосованості.

Продовження

Назва функції	Файл	• Вхідні параметри	Семантика параметрів	Призначення функції
calculate_-objective_-value	fitness_-function.py	• flows. • distances • assignment.	• Матриця потоків. • Матриця відстаней. • Призначення. •	Обчислення значення цільової функції.
run_-genetic_-algorithm	genetic.py	• flows • distances • population-_size • number_of_-generation • is_draw	• Матриця потоків. • Матриця відстаней. • Розмір популяції. • Кількість поколінь. • Показник, чи малювати графік.	Запуск генетичний алгоритм.

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
exchange	individual.py	• facility1. • facility2.	Індекс призначення 1, 2.	Обмін місця двох призначень.
generate_initial-_population	local_search.py	• flows. • distances. • number_-of_individuals.	• Матриця потоків. • Матриця відстаней. • Кількість розв’язків.	Генерація почат-кових розв’язків
two_opt_-improvement	local_search.py	individual.	Розв’язок.	Запуск ітераційне покращення
local_improvement	local_search.py	individual.	Розв’язок.	Виклик ітераційне покращення

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
run_iterated_local_search	local_search.py	• flows. • distances. • number_-of_individuals. • number_-of_-iterations. • number-_of_-shuffles. • is_draw.	• Матриця потоків. • Матриця відстаней. • Кількість розв’язків. • Кількість ітерацій. • Кількість тасування. • Показник, чи малювати графік.	Запуск алгоритму локального пошуку.
mutate_-chromosome	mutation.py	• chromosome. • random_-indexes.	• Хромосом. • Випадкові індекси в хромосомі.	Виконання оператор мутації.
generate_-random_-gen_indexes	mutation.py	chromosome	Хромосом.	Генерація випадковий індекс для мутації

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
generate_-random-_population	population.py	• number_-of_objects number_of_-random_-chromo-somes	• Кількість генів. Кількість хромосо-мів.	Випадково згенерувати початкову популяцію.
tournament_selection	selection.py	• population_list • fitness_scores_list	• Список хромосомів в популяції. • Список значення цільової функції.	Виконання турнірний відбір.
accProb	simulated_annealing.py	• delta • T	• Різниця значення цільової функції. • Початкова темпера-тура.	Обчис-лення значення ймовірності за допомо-гою формули Больцмана-Гіббса.

Продовження таблиці 3.5.1.

Назва функції	Файл	Вхідні параметри	Семантика параметрів	Призначення функції
generate_neighbors	tabu_search.py	• assignments • number_of_facilities • flows distances	• Призначення. • Кількість фабрики. • Матриця потоків. Матриця відстаней.	Генерувати сусідів для поточного розв’язку.
is_in_-tabu_-list	tabu_search-.py	• tabu_iden-tifiers • tabu_list	• Іденти-фікатор табу. • Список табу	Перевірка присутності в списку табу
run_-tabu_-search	tabu_search-.py	• flows • distances • tabu_size • number_-of_iterations • assignment • is_draw	• Матриця потоків. • Матриця відстаней. • Розмір списку табу. • Кількість ітерацій. • Призначення.	Запуск алгоритм табу пошуку.

			• Показник, чи малювати графік.	

3.6. Керівництво користувача.

Програма було розроблена як консольний додаток, тому графічний інтерфейс було мінімізовано. Щоб запускати програми необхідно ввести командну лінію python main.py, як показано на рисунку 3.6.1. Після запуску, консоль відразу відображає меню, як показано на рисунку 3.6.2.

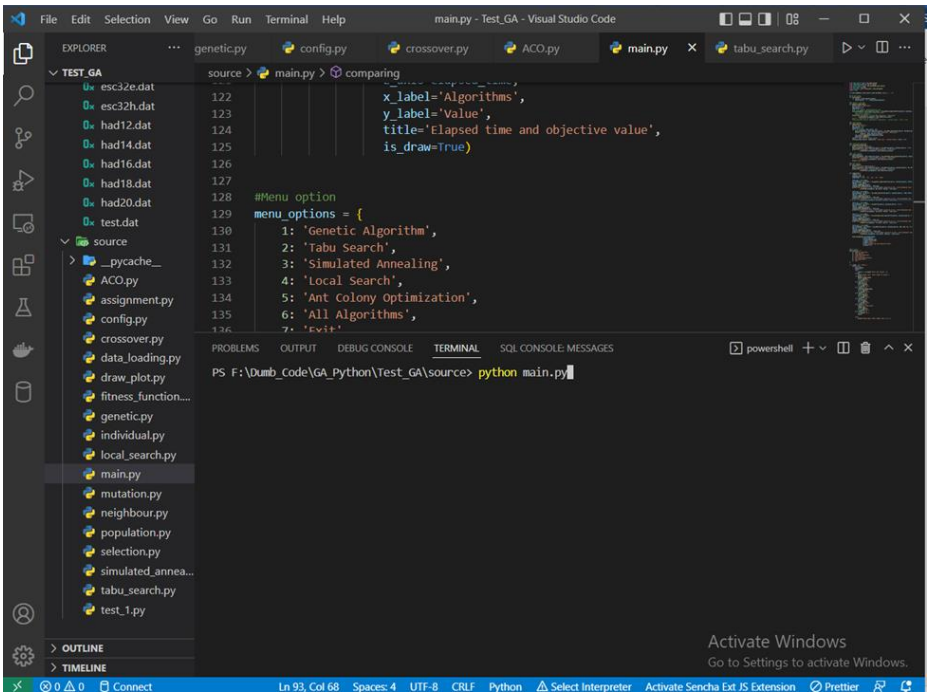


Рисунок 3.6.1 – Командна лінія для запуску програми.

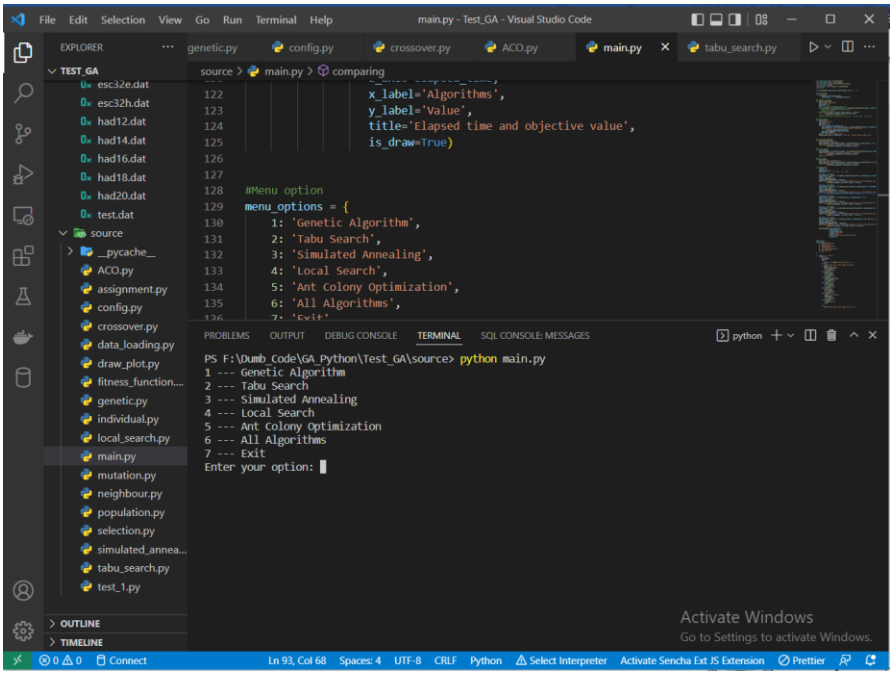


Рисунок 3.6.2 – Головний меню.

Для запуску алгоритма, необхідно ввести номер пункт меню відповідно до алгоритму. Ввести номер 1 та натиснути Enter, на консольний екран відображається найкраще призначення та найкраще значення цільової функції, який був розв’язаний за допомогою генетичного алгоритма. На рисунку 3.6.3, 3.6.4 відображається результат алгоритму та лінійний графік залежності цільової функції від покоління.

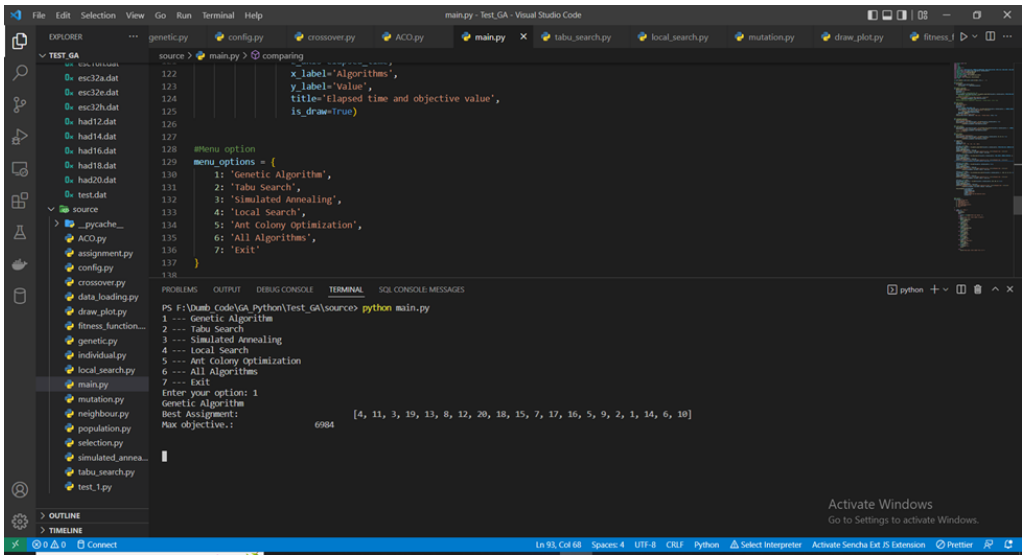


Рисунок 3.6.3 – Відображення результат на консольному екрані.

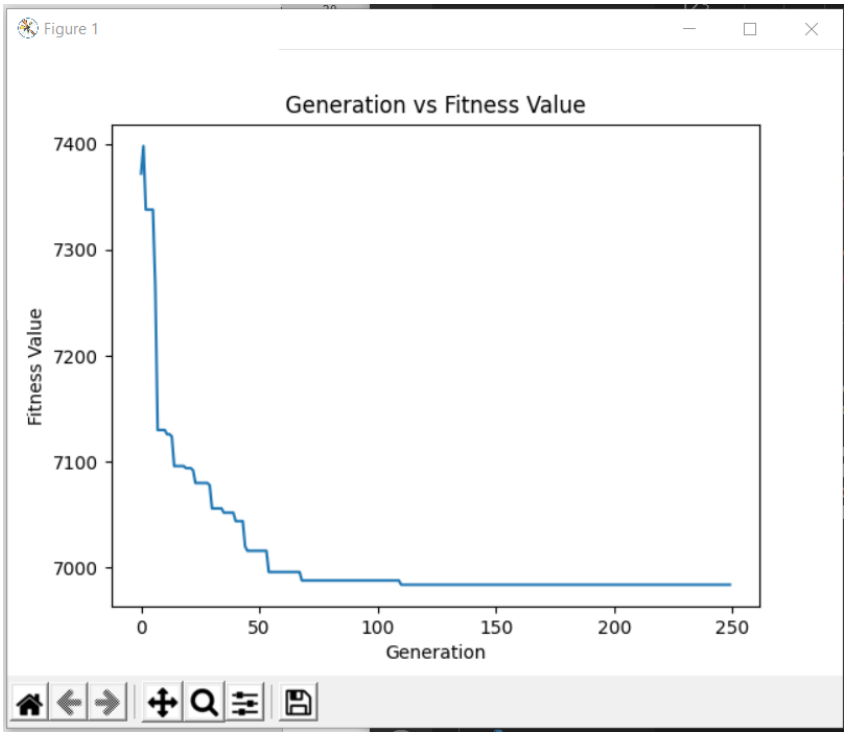


Рисунок 3.6.4 – Лінійний графік залежності цільової функції від покоління.

Для запуску інших алгоритмів, необхідно натиснути будь-які кнопки на клавіатурі, щоб очистити консольний екран, а також відображення нового меню як показано на рисунку 3.6.5. На рисунку 3.6.6, 3.6.7 відображається обробка іншого алгоритму після очищення екран.

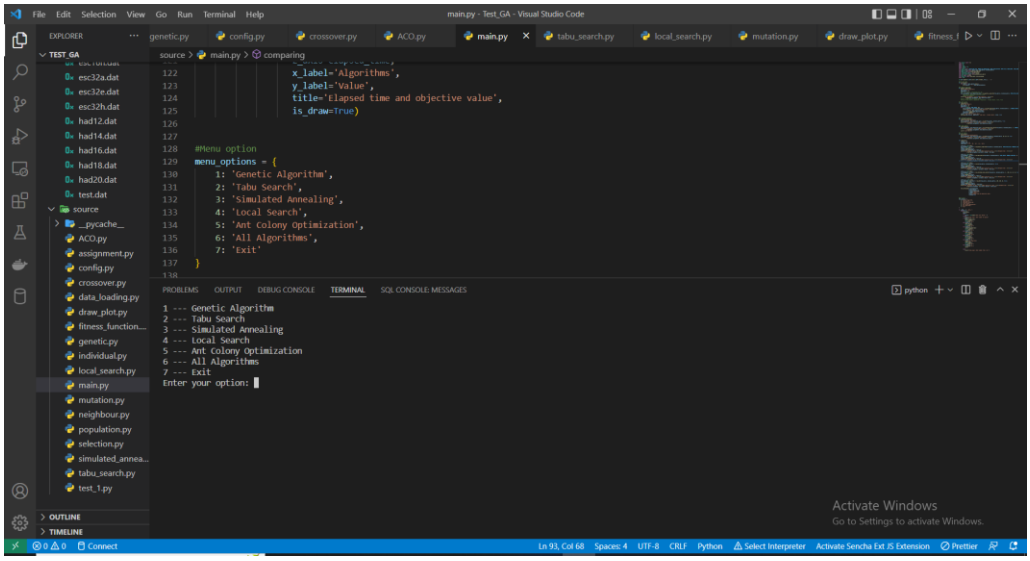


Рисунок 3.6.5 – Консольний екран після натискання будь-які кнопки на клавіатурі.

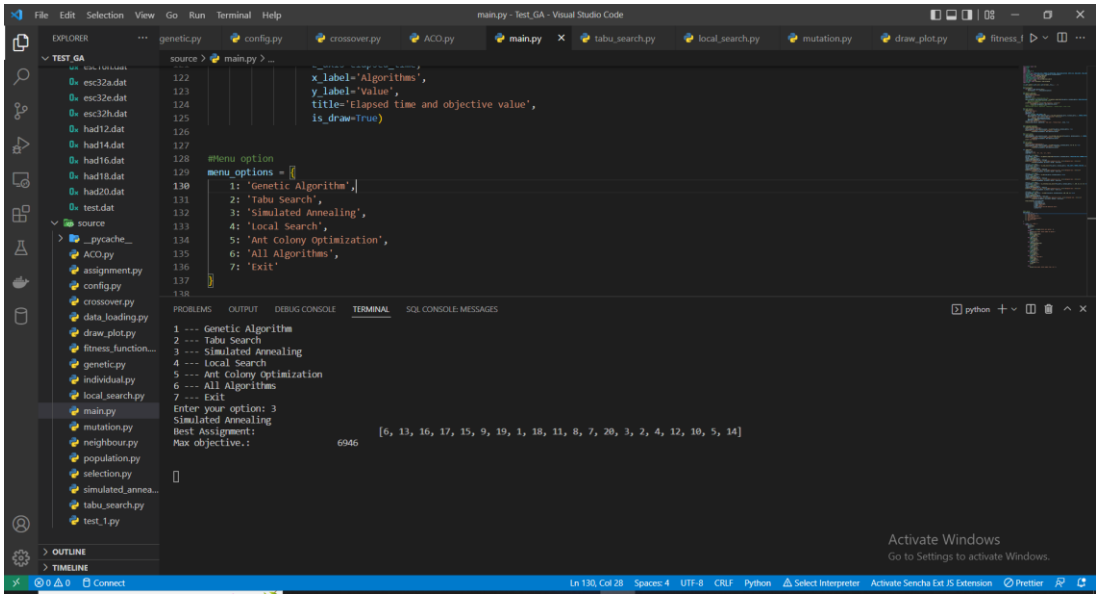


Рисунок 3.6.6 – Обробка алгоритм імітаційного відпалу після очищення.

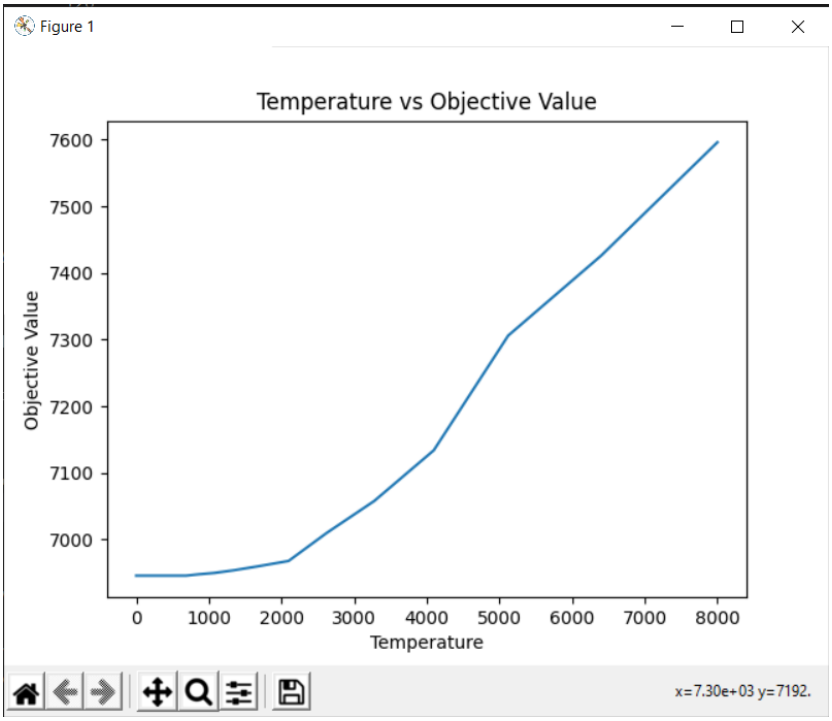


Рисунок 3.6.7 – Лінійний графік залежності значення цільової функції від температури.

Якщо ввести пункт 6 меню, всі алгоритми будуть запущені послідовно і на консольному екрані буде відображено найкраще призначення, найкраще значення цільової функції та час виконання кожного алгоритму. На рисунку 3.6.8, 3.8.9 відображено результати кожного алгоритму та стовпчикова діаграма.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   SQL CONSOLE: MESSAGES

Enter your option: 6
All algorithm
Best Assignment of GA:                [4, 1, 8, 19, 11, 12, 6, 18, 7, 13, 20, 2, 5, 16, 17, 14, 15, 10, 3, 9]
Best objective of GA.:          7082
Elapsed time :          34.21523475646973


Best Assignment of TB:                [2, 16, 13, 18, 8, 6, 10, 4, 17, 11, 12, 7, 14, 5, 1, 9, 15, 19, 3, 20]
Best objective of TB.:          6980
Elapsed time :          25.722004413604736


Best Assignment of SA:                [4, 15, 13, 17, 12, 9, 3, 1, 18, 11, 8, 10, 20, 19, 2, 5, 16, 6, 7, 14]
Best objective of SA.:          6938
Elapsed time :          5.075294017791748


Best Assignment of LS:                [ 9 15 16 17 13  6  7  1 18 11 12 10 20  4  2  3  8 19  5 14]
Best objective of LS.:          6922
Elapsed time :          26.000805377960205


Best Assignment of ACO:                     [9, 15, 13, 17, 16, 4, 7, 1, 18, 11, 8, 10, 14, 5, 2, 3, 12, 19, 6, 20]
Best objective of ACO.:                6928
Elapsed time :          371.0821304321289

Ln 130, Col 28   Spaces: 4   UTF-8   CRLF   Python   Select Interpreter
```

Рисунок 3.6.8 – Результати обробки кожного алгоритму.

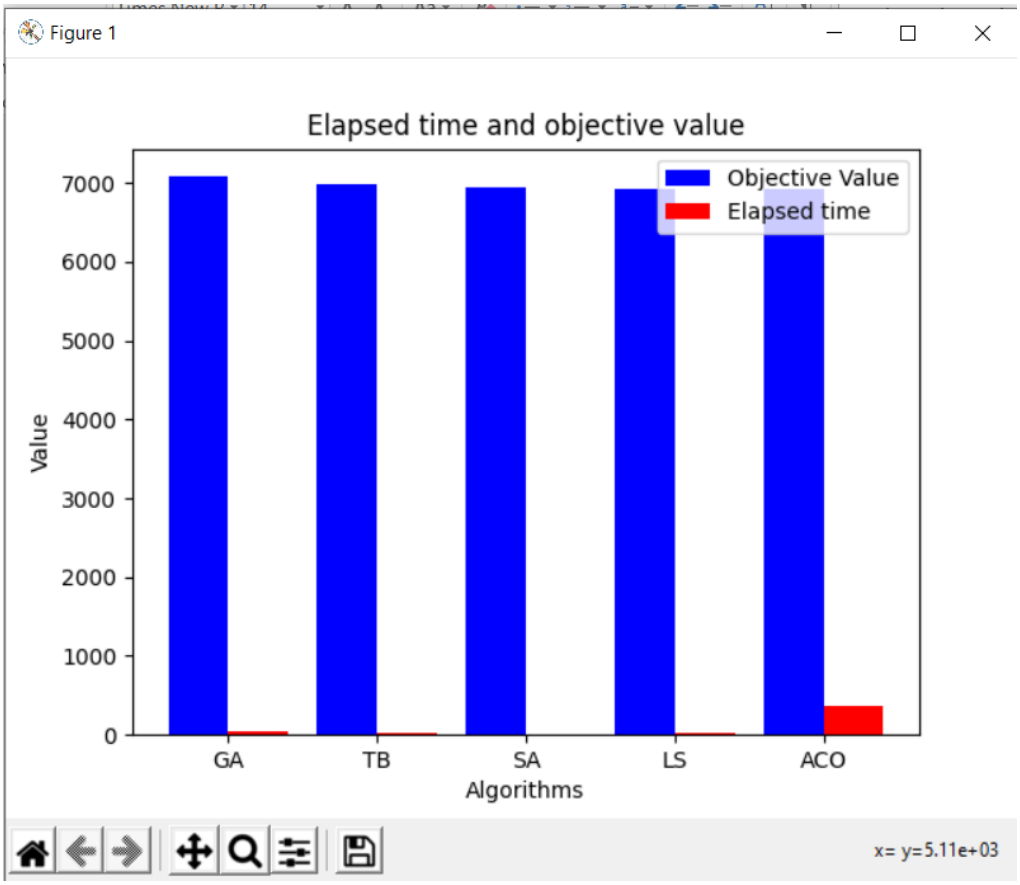


Рисунок 3.6.8 – Стовпчикова діаграма.

Щоб вийти з програми, ввести 7 та натиснути Enter, консольний екран буде відображати екран термінала, як показано на рисунку 3.6.10.

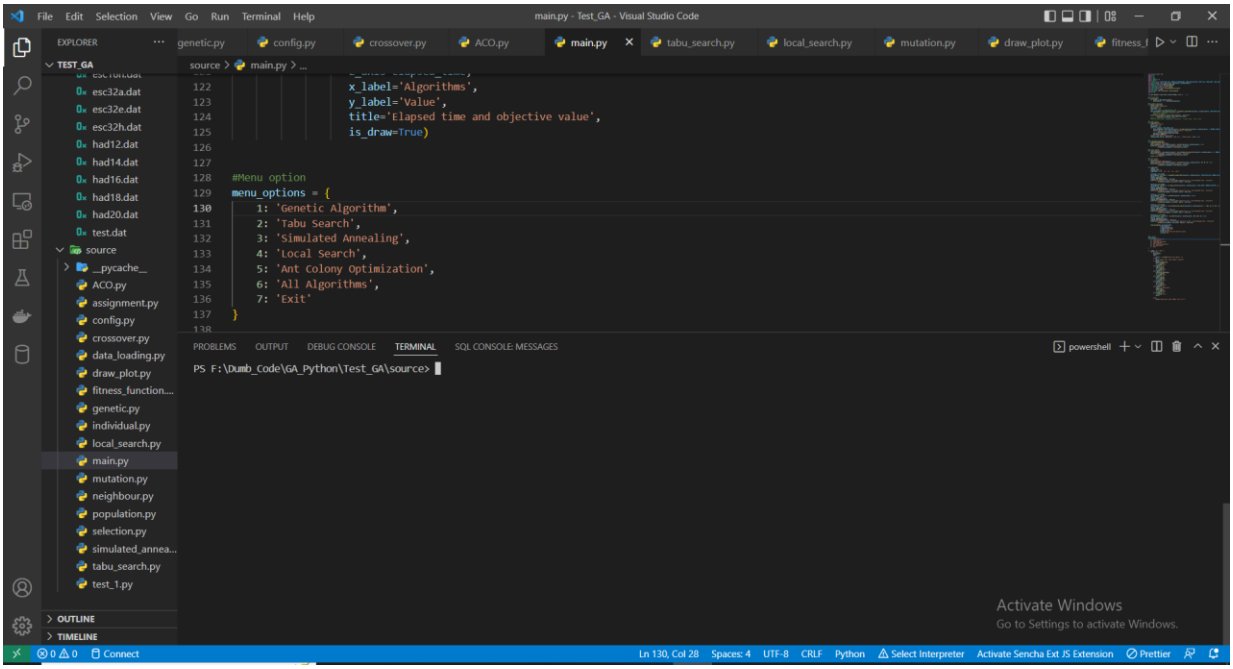


Рисунок 3.6.10 – Консольний екран після завершення роботи програми.

Висновок до розділу.

В цьому розділі було розглянуто основні засоби та технології для розробки програмного забезпечення, та проаналізовано переваги та недоліки кожного засобу. Мова програмування була обрана Python. Останнім часом, Python являється однією з найпопулярніших мов програмування, Python був використаний в різних галузях, від розробки додаток веб-сайту до науки даних та інші проекти з науковими цілями. Крім мови програмування, інструмент для програмування був використаний Visual Studio Code або VS Code. Visual Studio Code це кодовий редактор, який був розроблений Microsoft. VS Code підтримує багато мов програмуваннь, VS Code також надає багато функціоналу, але реалізує компактніше і зручніше для використання порівняно з іншими IDE.

Програмне забезпечення був реалізований як консольний додаток. Через мінімізації ресурсів для графічного інтерфейсу, можна виділити більше ресурсів для логіки обробки алгоритмів, оскільки деякі алгоритми при обробки надає навантаження на процесорі та споживає пам’яті.

ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

4.1. Вхідні дані.

Дані для дослідження метаевристичних алгоритмів беруться для з веб-сайту QAPLIB. QAPLIB це бібліотека, яка містить постановки КЗП та розв'язок до них. Всі дані представляється у вигляді як показано на рисунку 4.1.1.

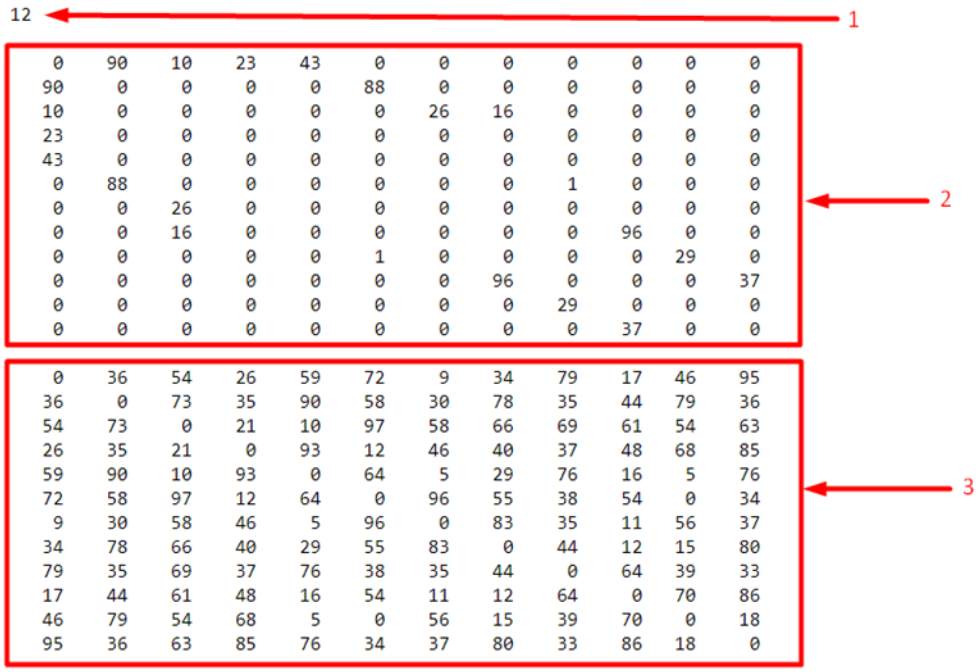


Рисунок 4.1.1 – Формат даних для КЗП.

де 1 – це розмір задачі, 2 – матриця потоків між закладами, 3 – матриця відстаней між містами.

QAPLIB вміщує понад 100 наборів даних, які використовувалися в попередніх дослідженнях, і частково вони походять із реальних додатків, таких як проблема пов'язана з поліклінікою (наприклад, набір даних проблеми kra32)

або проблема пов’язана з дизайном друкарської машинки (наприклад, екземпляри bur26) [26]. Проблеми QAPLIB поділяють на чотири основні класи:

- Неструктуровані, випадково згенеровані проблеми.
- Екземпляри матриці відстані на основі сітки.
- Реальні проблеми.
- Проблеми подібні до реального життя.

4.2. Аналіз отриманих результатів.

Для виконання експериментів було використано комп’ютер з наступними конфігураціями:

- Операційна система комп’ютера – Window 10 Pro, 64-bit.
- Розмір оперативної пам’яті – 8 ГБ.
- Процесор комп’ютера – Intel(R) Core(TM) i5-6200U, 2.3 ГГц
- Графічний процесор 1 – Intel(R) HD Graphics 520.
- Графічний процесор 2 – AMD Radeon R5 M335.

Інструмент для розробки має наступні конфігурації:

- Версія Python – 3.8.7.
- Версія numpy – 1.19.5.
- Версія matplotlib – 3.3.4.
- Версія Visual Studio Code – 1.67.2.

В таблиці 4.2.1 наведено список проблеми, які будуть використовуватися для перевірки програмного забезпечення, їх розмір та оптимальне значення.

Таблиця 4.2.1 – Таблиця назви, розміру та оптимального значення проблеми.

Назва проблеми	Розмір проблеми	Оптимальне значення
Bur26a	26	5426670
Chr22a	22	6156
Ecs16a	16	68

Продовження таблиці 4.2.1.

Назва проблеми	Розмір проблеми	Оптимальне значення
Esc32a	32	130
Esc16h	16	996
Esc16b	16	292
Had18	18	5358
Had20	20	6922
Nug24	24	3488
Nug25	25	3744
Scr15	15	51140
Scr12	12	31410
Tai15a	15	388214
Tai20a	20	703482
Tai30a	30	1818146
Tai40a	40	3139370
Wil50	50	48816

Для перевірки чи результат поліпшує за кожним кроком, використовував проблема Had20. На рисунку 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5 відображає покращення результат обробки алгоритму за кожної ітерації.

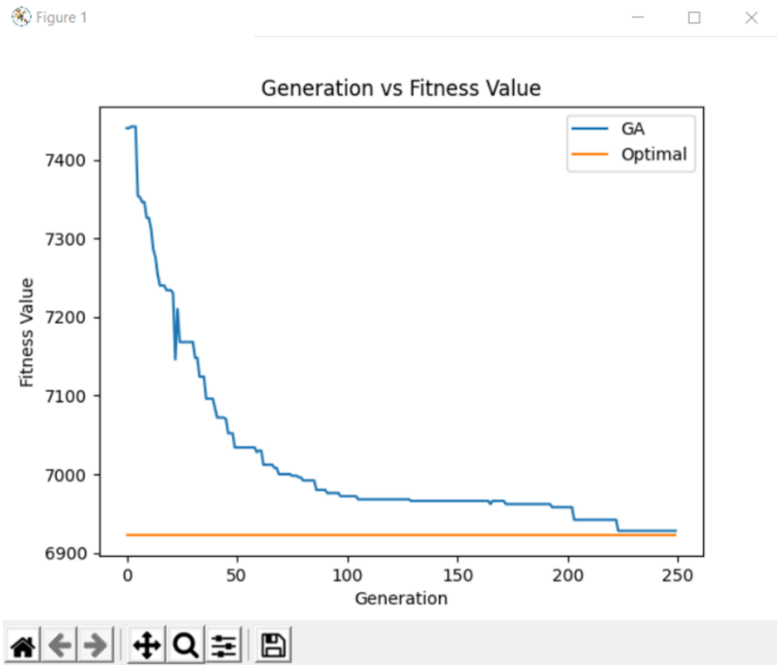


Рисунок 4.2.1 – Покращення результату генетичного алгоритму.

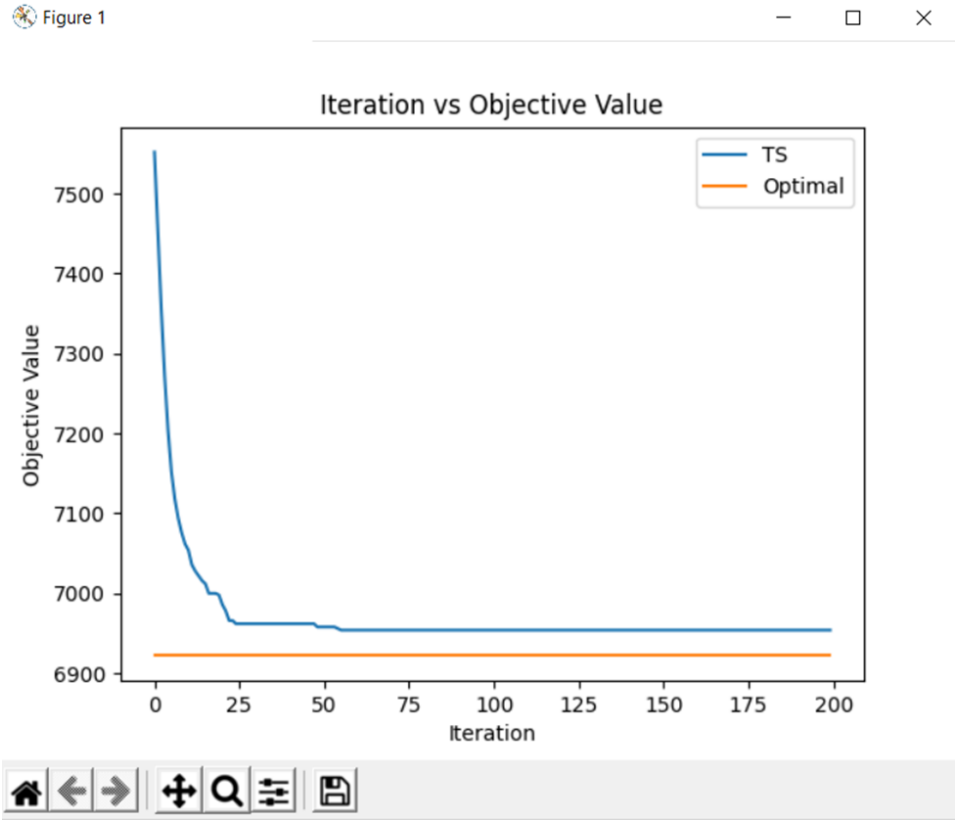


Рисунок 4.2.2 – Покращення результату алгоритму табу пошуку.

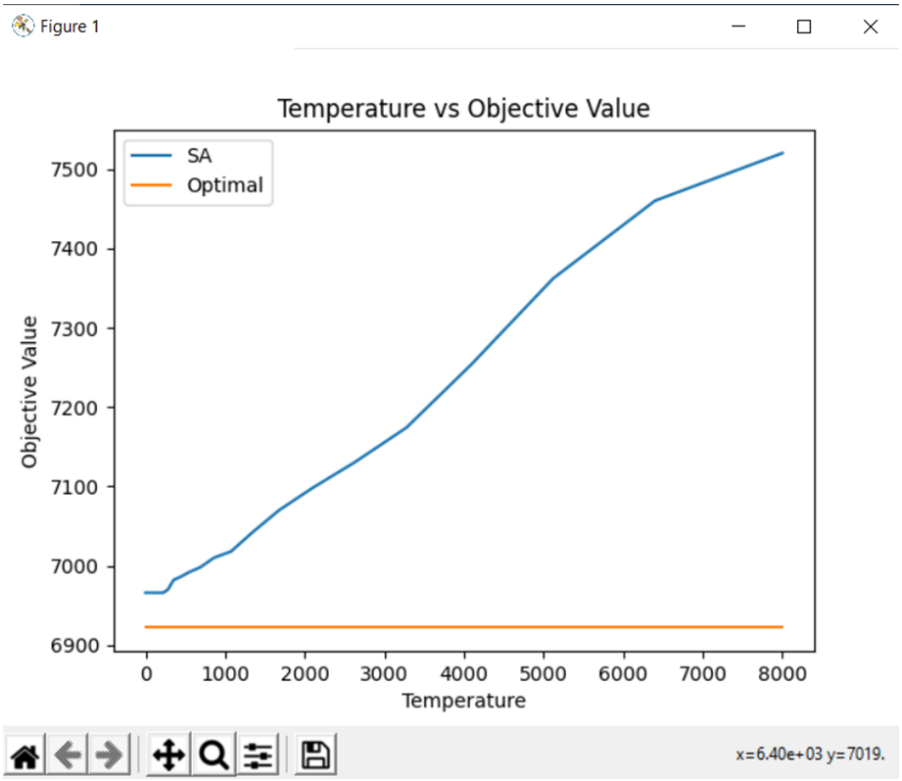


Рисунок 4.2.3 – Покращення результату алгоритму імітаційного відпалу.

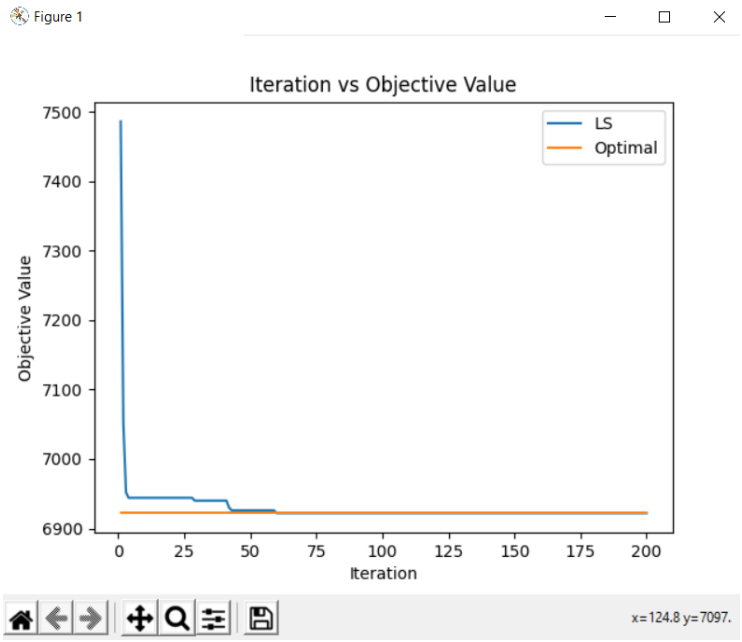


Рисунок 4.2.4 – Покращення результату алгоритму детермінованого локального пошуку.

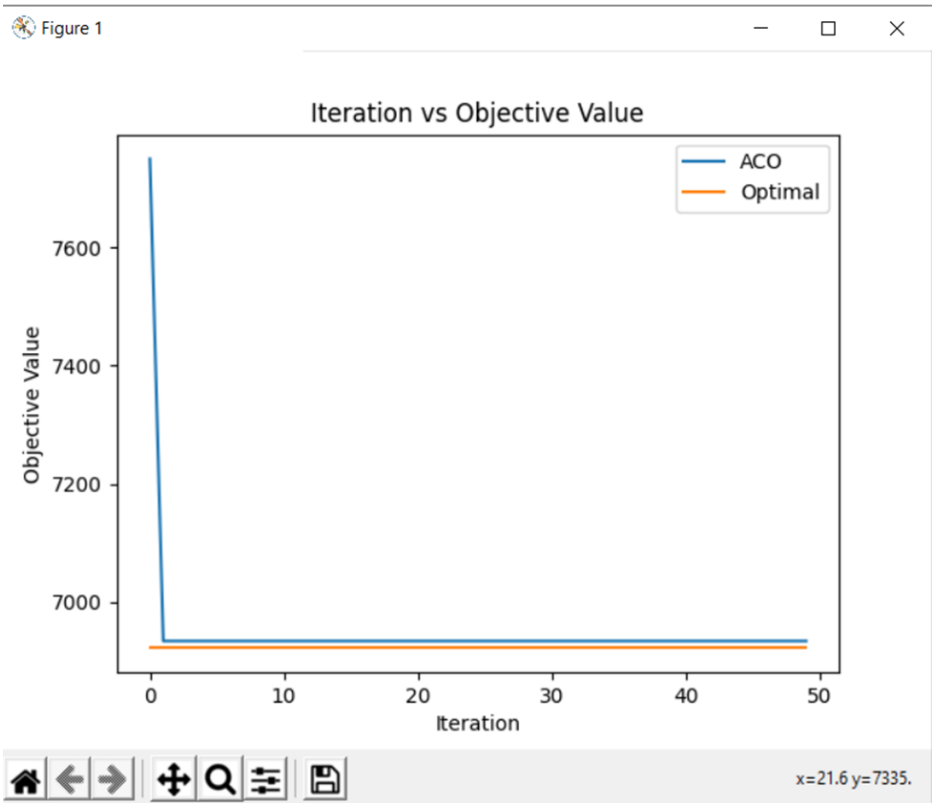


Рисунок 4.2.5 – Покращення результату алгоритму оптимізації мурашиними колоніями.

В таблиці 4.2.2 показує кінцевий результат після обробки кожного алгоритму.

Таблиця 4.2.2 – Таблиця кінцевих результатів виконання алгоритмів.

Назва проблеми	Результат ГА	Результат табу пошуку	Результат AIB	Результат локального пошуку	Результат ОМК
Bur26a	5442926	5435221	5452866	5428397	5439985
Chr22a	7206	6518	7444	6504	6840
Ecs16a	68	68	68	68	68
Esc32a	142	136	174	132	146

Продовження таблиця 4.2.2.

Назва проблеми	Результат ГА	Результат табу пошуку	Результат AIB	Результат локального пошуку	Результат ОМК
Esc16h	996	996	996	996	996
Esc16b	292	292	292	292	292
Had18	5358	5358	5410	5358	5360
Had20	7054	6948	7002	6926	6950
Kra32	95410	91140	93740	91250	99290
Nug20	2674	2596	2734	2584	2608
Nug24	3708	3510	3600	3500	3626
Nug25	3952	3750	3834	3768	3898
Scr15	54922	54930	55192	51140	53684
Scr12	35288	31410	34382	31410	31410
Tai15a	407928	392258	400150	388250	399230
Tai20a	761480	710898	742814	726952	743532
Tai30a	1913808	1857592	1867540	1865526	1959208
Tai40a	3281124	3202812	3298490	3222114	3262504
Wil50	50252	49694	49590	49005	51008

Відносне відхилення результату алгоритму від оптимального обчислюється за формулою:

$$p = \frac{(RES - OPT)}{OPT} * 100\% \qquad (4.2.1)$$

де p – відносне відхилення алгоритму, RES – результат виконання алгоритму, OPT – оптимальне значення, яке введено в QAPLIB.

В таблиці 4.2.3 показує відносне відхилення результатів кожного алгоритму від оптимального.

Таблиця 4.2.3 – Відносне відхилення кінцевого результату кожного алгоритму порівняно з оптимального.

Назва проблеми	Відносне відхилення ГА	Відносне відхилення табу пошуку	Відносне відхилення AIB	Відносне відхилення локального пошуку	Відносне відхи-лення ОМК
Bur26a	0.300 %	0.158 %	0.483 %	0.032 %	0.245 %
Chr22a	17.057 %	5.880 %	20.923 %	5.653 %	11.111 %
Ecs16a	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %
Esc32a	9.231 %	4.615 %	33.846 %	1.538 %	12.308 %
Esc16h	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %
Esc16b	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %
Had18	0.000 %	0.000 %	0.971 %	0.000 %	0.037 %
Had20	1.907 %	0.376 %	1.156 %	0.058 %	0.405 %
Kra32	7.565 %	2.751 %	5.682 %	2.875 %	11.939 %
Nug20	4.047 %	1.012 %	6.381 %	0.545 %	1.479 %
Nug24	6.307 %	0.631 %	3.211 %	0.344 %	3.956 %
Nug25	5.556 %	0.160 %	2.404 %	0.641 %	4.113 %
Scr15	7.395 %	7.411 %	7.923 %	0.000 %	4.975 %
Scr12	12.346 %	0.000 %	9.462 %	0.000 %	0.000 %
Tai15a	3.972 %	0.889 %	3.834 %	0.000 %	1.824 %
Tai20a	8.244 %	1.054 %	5.591 %	3.336 %	5.693 %
Tai30a	5.262 %	2.170 %	2.717 %	2.606 %	7.759 %

Продовження таблиці

Назва проблеми	Відносне відхилення ГА	Відносне відхилення табу пошуку	Відносне відхилення АІВ	Відносне відхилення локального пошуку	Відносне відхи-лення ОМК
Tai40a	4.515 %	2.021 %	5.069 %	2.636 %	3.922 %
Wil50	2.942 %	1.799 %	1.586 %	0.387 %	4.490 %

На рисунку 4.2.6 відображає лінійний графік відносного відхилення кожного алгоритму.

Рисунок 4.2.6 – Графік відносного відхилення кожного алгоритму.

В таблиці 4.2.4 відображає час виконання кожного алгоритму.

Таблиця 4.2.4 – Таблиця часу виконання кожного алгоритму.

Назва проблеми	Час виконання ГА	Час виконання табу пошуку	Час виконання АІВ	Час виконання локального пошуку	Час виконання ОМК
bur26a	45.32 с	65.08 с	14.17 с	59.04 с	869.5 с

Продовження таблиці 4.2.4.

Назва проблеми	Час виконання ГА	Час виконання табу пошуку	Час виконання AIB	Час виконання локального пошуку	Час виконання ОМК
Chr22a	35.21 с	29.7 с	7.16 с	39.17 с	528.48 с
Ecs16a	19.21 с	10.93 с	1.92 с	12.37 с	206.97 с
Esc32a	69.17 с	134.82 с	35.47 с	111.05 с	605.98 с
Had18	23.83 с.	13.29 с.	3.36 с.	19.17 с.	1237.71 с.
Had20	28.6 с.	24.4 с.	5.14 с.	27.04 с.	387.77 с.
Nug25	39.9 с.	43.57 с.	11.36 с.	48.16 с.	705.42 с.
Scr15	20.33 с.	7.89 с.	1.89 с.	12.03 с.	159.86 с.
Scr12	13.03 с.	2.67 с.	0.74 с.	5.33 с.	109.34 с.
Tai15a	17.12 с.	7.44 с.	1.63 с.	11.46 с.	190.84 с.
Tai20a	29.42 с.	17.7 с.	5.23 с.	26.62 с.	386.45 с.
Tai30a	62.2 с.	107.47 с.	23.79 с.	78.2 с.	470.87 с.
Tai40a	102.88 с.	324.03 с.	82.48 с.	239.92 с.	1279.19 с.
Wil50	165.27 с.	818.64 с.	208.21 с.	450.88 с.	3024.35 с

На рисунку 4.2.7 відображає лінійний графік часу виконання кожного алгоритму.

Рисунок 4.2.7 – Графік часу виконання кожного алгоритму.

Висновок до розділу.

В цьому розділі, використовували генетичний алгоритм, алгоритм табу пошука, алгоритм імітаційного відпалу, алгоритм детермінованого локального пошуку та алгоритм оптимізації мурашиними колоніями для розв’язування КЗП з різними розмірами. Дослідження присвячено порівнянні відхилення кінцевих результатів від оптимальних, яких введено в QAPLIB та час виконання алгоритму. Згідно з результатом дослідження, алгоритм детермінованого локального пошуку надає результат з найменшим відхиленням. Алгоритм імітаційного відпалу можна вважати як найшвидший алгоритм, але кінцевий результат цього алгоритму не так добре порівняно з іншими. Генетичний алгоритм та табу пошук має не поганий результат та час виконання. Для задачі з маленьким розміром, ці алгоритми надають майже оптимальні результати. Оскільки в алгоритмі оптимізації мурашиними колоніями виконує табу пошук для покращення результати в кожній ітерації, цей алгоритм займає найбільше часу для виконання. У майбутніх дослідженнях, необхідно розробити та оптимізувати метаевристичні алгоритми з метою розв’язання великих КЗП з мінімальною витратою часу та оптимальним результатом.

ВИСНОВОК

В магістерській дисертації проведений аналіз метаевристичних алгоритмів та розроблено програму реалізації цих алгоритмів для розв’язування КЗП.

Для реалізації програмного забезпечення було використано мову програмування Python та її допоміжні бібліотеки. Останнім часом Python є однією із найпопулярніших мов програмування. Python ідеально підходить для таких галузей, як розробка веб-додатків, наука про даних, штучний інтелект та машинне навчання. Завдяки тому, що Python має простий синтаксис та легкий для початківців, спільнота розробників Python дуже швидко зростає і це допомагає іншим розробникам швидко знайти відповідь на їхнє питання. Окрім Python, для розробки також використовували інші допоміжні бібліотеки. За допомогою numpy, всі операції, пов’язані з масивом, стали швидшими та компактнішими, ніж використання типу даних List Python. Масив споживає менше пам’яті і зручний у використанні. NumPy використовує набагато менше пам'яті для зберігання даних і забезпечує механізм визначення типів даних. Це дозволяє ще більше оптимізувати код.

Результати обробки алгоритму порівняні з оптимальним значенням, який подано у відомій бібліотеці QAPLIB. Як показано в розділі дослідження, отримуваний результат кожного алгоритму має відносно невелике відхилення від оптимального. Алгоритм імітаційного відпалу є найшвидшим серед всіх, але результат виконання для деяких задач за точністю є найгіршими. Генетичний алгоритм та табу пошук мають непогані результати та час виконання. З процедурою покращення, алгоритм ОМК для обробки займає найбільше часу, але в певних випадках, результат цього алгоритму краще ніж генетичного алгоритму та табу пошуку.

СПИСОК ПОСИЛАНЬ

1. A comprehensive review of quadratic assignment problem: variants, hybrids and applications / M. Abdel-Basset et al. *Journal of Ambient Intelligence and Humanized Computing*. 2018. URL: https://doi.org/10.1007/s12652-018-0917-x.

2. A survey for the quadratic assignment problem / E. M. Loiola та ін. *European Journal of Operational Research*. 2007. Т. 176, № 2. С. 657–690. URL: https://doi.org/10.1016/j.ejor.2005.09.032.

3. Л. Ф. Гуляницький, О. Ю. Мулеса. Прикладні методи комбінаторної оптимізації, 2016. 142 с.

4. Metaheuristic Algorithms: A comprehensive review / Abdel-Basset, M., Abdel-Fatah, L.,; Sangaiah, A. K. Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications. 2018. С. 185–231. URL: https://doi.org/10.1016/b978-0-12-813314-9.00010-4.

5. Multirobot coordination through bio-inspired strategies / De Rango, F., Palmieri, N.,; Tropea, M. Nature-Inspired Computation and Swarm Intelligence. 2020. с 361–390. URL: https://doi.org/10.1016/b978-0-12-819714-1.00030-0

6. Zong Woo Geem, Joong Hoon Kim, Loganathan G. V. A New Heuristic Optimization Algorithm: Harmony Search. *SIMULATION*. 2001. Т. 76, № 2. С. 60–68. URL: https://doi.org/10.1177/003754970107600201.

7. Zhongda Tian and Chao Zhang (2018). An Improved Harmony Search Algorithm and Its Application in Function Optimization. Journal of Information Processing Systems, 14(5), с.1237-1253. URL: https://doi.org/10.3745/JIPS.04.0090.

8. A modified single and multi-objective bacteria foraging optimisation for the solution of quadratic assignment problem / S. Parvandeh та ін. *International*

Journal of Bio-Inspired Computation. 2021. Т. 17, № 1. С. 1. URL: https://doi.org/10.1504/ijbic.2021.113354.

9. Liu H., Abraham A., Zhang J. A Particle Swarm Approach to Quadratic Assignment Problems. *Advances in Soft Computing*. Berlin, Heidelberg. С. 213–222. URL: https://doi.org/10.1007/978-3-540-70706-6_20.

10. Karaboga D. Artificial bee colony algorithm. *Scholarpedia*. 2010. Т. 5, № 3. С. 6915. URL: https://doi.org/10.4249/scholarpedia.6915.

11. Genetic Algorithm: Review and Application / M. Kumar та ін. *SSRN Electronic Journal*. 2010. URL: https://doi.org/10.2139/ssrn.3529843.

12. Glover F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*. 1986. Т. 13, № 5. С. 533–549. URL: https://doi.org/10.1016/0305-0548(86)90048-1.

13. Glover F. Tabu Search–Part I. *ORSA Journal on Computing*. 1989. Т. 1, № 3. С. 190–206. URL: https://doi.org/10.1287/ijoc.1.3.190.

14. Knox J. Tabu search performance on the symmetric traveling salesman problem. *Computers & Operations Research*. 1994. Т. 21, № 8. С. 867–876. URL: https://doi.org/10.1016/0305-0548(94)90016-7.

15. Traveling-Salesman-Problem Algorithm Based on Simulated Annealing and Gene-Expression Programming / A.-H. Zhou та ін. *Information*. 2018. Т. 10, № 1. С. 7. URL: https://doi.org/10.3390/info10010007.

16. De Corte A., Sörensen K. An Iterated Local Search Algorithm for Multi-Period Water Distribution Network Design Optimization. *Water*. 2016. Т. 8, № 8. С. 359. URL: https://doi.org/10.3390/w8080359.

17. Automatic enhancement of coronary arteries using convolutional gray-level templates and path-based metaheuristics / M.-A. Gil-Rios та ін. *Recent Trends in Computational Intelligence Enabled Research*. 2021. С. 129–153. URL: https://doi.org/10.1016/b978-0-12-822844-9.00005-0.

18. Lourenço H. R., Martin O. C., Stützle T. Iterated Local Search. *Handbook of Metaheuristics*. Boston. С. 320–353. URL: https://doi.org/10.1007/0-306-48056-5_11.

19. Dorigo M. Ant colony optimization. *Scholarpedia*. 2007. Т. 2, № 3. С. 1461. URL: https://doi.org/10.4249/scholarpedia.1461.

20. Ranking programming languages by energy efficiency / R. Pereira та ін. *Science of Computer Programming*. 2021. Т. 205. С. 102609. URL: https://doi.org/10.1016/j.scico.2021.102609.

21. Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms / Jinghui Zhong та ін. *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, м. Vienna, Austria. URL: https://doi.org/10.1109/cimca.2005.1631619.

22. A Comparison of GA Crossover and Mutation Methods for the Traveling Salesman Problem / R. T. Bye та ін. *Advances in Intelligent Systems and Computing*. Singapore, 2020. С. 529–542. URL: https://doi.org/10.1007/978-981-15-6067-5_60.

23. Liu T. How significant is the initial population when using a genetic algorithm to solve the quadratic assignment problem?. *SIAM Undergraduate Research Online*. 2021. Т. 14. URL: https://doi.org/10.1137/21s1406192.

24. Sarmady, S. (2007). An investigation on genetic algorithm parameters. School of Computer Science, Universiti Sains Malaysia, 126.

25. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. Science, vol. 220. 1983. с. 671–680.

26. Stützle, T., & Dorigo, M. Local search and metaheuristics for the quadratic assignment problem. Technical Report. 2001. 20 с.

27. Burkard R. E., Karisch S., Rendl F. QAPLIB-A quadratic assignment problem library. *European Journal of Operational Research*. 1991. Т. 55, № 1. С. 115–119. URL: https://doi.org/10.1016/0377-2217(91)90197-4.

28. Ant Colony Optimization and Swarm Intelligence / ред.: M. Dorigo та ін. Berlin, Heidelberg : Springer Berlin Heidelberg, 2004. URL: https://doi.org/10.1007/b99492.

ДОДАТОК

Додаток А Код генетичного алгоритму

```
from population import generate_random_population
from selection import Selection, TournamentSelection, RouletteSelection
from mutation import Mutation, BasicMutation
from crossover import BasicCrossover, Crossover
from fitness_function import get_normalized_result_of_fitness_function_scores_list, compute_fitness_scores_list
import numpy as np
from draw_plot import draw_plot
from config import OPT
def run_genetic_algorithm(
    flows,
    distances,
    population_size,
    number_of_generation,
    is_draw
):
    number_of_facilities = len(flows)
    selection_strategy = Selection(selection_algorithm=TournamentSelection())
    mutation_strategy = Mutation(mutation_algorithm=BasicMutation())
    crossover_strategy = Crossover(crossover_algorithm=BasicCrossover())
    population = generate_random_population(number_of_facilities, population_size)
```

```
        generation_indicies = []
        average_results = []
        min_results = []
        max_results = []
        opt = []
        previous_max_chromosome = []
        fitness_value_arr = []
        def print_console_output():
            print("Epoch: \t\t\t\t{}\nMean fit.: \t\t\t{}\nMax score: \t\t\t{}\nMax chrom.: \t\t{}\n\n"
                  .format(epoch, average_fitness, max_fitness, max_chromosome))


        for epoch in range(number_of_generation):


            fitness_scores = compute_fitness_scores_list(population, distances, flows)
            fitness_scores_normalized = get_normalized_result_of_fitness_function_scores_list(fitness_scores)


            # While it's not normalized yet, max means "the worst", therefor "min" for us.
            max_fitness = np.min(fitness_scores)
            min_fitness = np.max(fitness_scores)
            average_fitness = np.mean(fitness_scores)
            max_results.append(max_fitness)
            min_results.append(min_fitness)
```

```
            average_results.append(average_fitness)
            generation_indicies.append(epoch)


            max_chromosome = population[np.argmin(fitness_scores)]
            max_chromosome = list(map(lambda value: value + 1, max_chromosome))
            selected_chromosomes = selection_strategy.select(population, fitness_scores_normalized)
            crossed_chromosomes = crossover_strategy.crossover(selected_chromosomes)
            mutated_chromosomes = mutation_strategy.mutate(crossed_chromosomes)
            #print_console_output()
            previous_max_chromosome = max_chromosome
            population = mutated_chromosomes
            opt.append(OPT)
        draw_plot(max_results, generation_indicies,opt,'GA', 'Generation', 'Fitness Value', 'Generation vs Fitness Value', is_draw)
        return max_chromosome, max_fitness
```