СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____

Кафедра _____інформаційних технологій та програмування_____

Пояснювальна записка

до магістерської дипломної роботи

_____магістр_____

(освітньо-кваліфікаційний рівень)

на тему: _Дослідження генетичного алгоритму із застосуванням локальної

_____оптимізації для вирішення задачі комівояжера_____

Виконав: студент групи_ІСТ-22дм_____

126 «Інформаційні системи та технології___

(шифр і назва спеціальності)

_____Пікун О.Д._____

(прізвище та ініціали)

Керівник ___Іванов В.Г._____

(прізвище та ініціали)

Рецензент_ __Меняйленко О.С._____

(прізвище та ініціали)

Київ – 2023 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет ___інформаційних технологій та електроніки_____
Кафедра _____інформаційних технологій та програмування_____
Освітньо-кваліфікаційний рівень __магістр_____
Спеціальність 126 «Інформаційні системи та технології»_____
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
_____ д.т.н., доц. Захожай О.І.
(підпис)
« ___ » _____2023 р.

ЗАВДАННЯ

на магістерську дипломну роботу студенту
Пікун Олександр Дмитрович
(прізвище, ім'я, по батькові)

1.Тема роботи: Дослідження генетичного алгоритму із застосуванням локальної оптимізації для вирішення задачі комівояжера
керівник роботи доцент, к.т.н. Іванов Віталій Геннадійович ,
(вчене звання, науковий ступінь, прізвище, ім'я, по батькові)
затверджені наказом університету від «_ __ »____ 2023року № 614/15.15-С___
2. Строк подання студентом роботи: 07 грудня 2023 р.
3. Вихідні дані до роботи: Матеріали науково-дослідної практики, науково-методична література; дані інтернет-мережі .
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ
4.2 Аналітичний огляд питання (огляд публічних джерел інформації)
4.3 Основна частина, в якої висвітлити методи, які будуть використовуватися для реалізації проекту.
4.4 Практична частина – огляд технологій, які використовуються під час реалізації проекту.
4.4 Висновки
4.5 Перелік використаних джерел

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____
_____

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 20 жовтня 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	Одержання завдання на виконання роботи	20.10.2023	
2.	Укладання і погодження з керівником плану і етапів виконання роботи	24.10.2023	
3.	Узагальнення даних літературних джерел	28.10.2023	
4.	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху виконання завдання	01.11.2023	
5.	Аналіз технічних засобів та існуючих систем	07.11.2023	
6.	Реалізація практичної частини завдання	24.11.2023	
7.	Укладання, оформлення та погодження пояснювальної записки з керівником	05.12.2023	
8.	Здача пояснювальної записки на кафедру	07.12.2023	
9.	Підготовка доповіді та презентації	09.12.2023	

Студент _____ Пікун О.Д.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Іванов В.Г.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дипломна работа: стор. 64с., Рисунок 10, табл. 13, джерел 9.

Актуальність: Завдання комівояжера (TSP), що є широко застосовуваною в різних галузях завданням комбінаторної оптимізації, має величезну складність, яка просто не описується в її простому описі, особливо при великій кількості міст. З метою подолання викликів, пов'язаних із розв'язанням задачі TSP, пропонується впровадити генетичний алгоритм у процес розв'язання. Генетичний алгоритм, що базується на принципах природного відбору та принципі "кращий виживає" в біологічному світі, є глобальним алгоритмом оптимізації, що демонструє самоорганізацію, адаптивність та здатність до самонавчання. В рамках даного дослідження пропонується новий метод оптимізації, що поєднує генетичний алгоритм та локальну оптимізацію, для вирішення задачі TSP. Проведення ретельного аналізу ефективності та порівняльного аналізу демонструє перевагу даного методу в порівнянні з традиційними підходами рішення.

Актуальність: Завдання комівояжера (TSP), що є широко застосовуваною в різних галузях завданням комбінаторної оптимізації, має величезну складність, яка просто не описується в її простому описі, особливо при великій кількості міст. З метою подолання викликів, пов'язаних із розв'язанням задачі TSP, пропонується впровадити генетичний алгоритм у процес розв'язання. Генетичний алгоритм, що базується на принципах природного відбору та принципі "кращий виживає" в біологічному світі, є глобальним алгоритмом оптимізації, що демонструє самоорганізацію, адаптивність та здатність до самонавчання. В рамках даного дослідження пропонується новий метод оптимізації, що поєднує генетичний алгоритм та локальну оптимізацію, для вирішення задачі TSP. Проведення ретельного аналізу ефективності та порівняльного аналізу демонструє перевагу даного методу в порівнянні з традиційними підходами рішення.

Об'єкт дослідження: Даної роботи полягає у вирішенні задачі комівояжера (TSP) з використанням генетичного алгоритму та локальної оптимізації**.**

Предмет дослідження: даного дослідження полягає у використанні генетичного алгоритму із застосуванням локальної оптимізації для вирішення задачі комівояжера (TSP) та проведення порівняльного аналізу з традиційними генетичними методами.

Методи дослідження: Теоретичний та експериментальний.

Результати: З проведеного дослідження було з'ясовано, що поєднання генетичного алгоритму з методами локальної оптимізації ефективно вирішує проблему комівояжера (TSP). У практичному застосуванні генетичний алгоритм забезпечує стабільне наближене оптимальне рішення. Слід зазначити, що ініціалізація та налаштування генетичних операторів є ключовими аспектами розробки генетичного алгоритму завдання TSP.

У порівнянні з традиційними алгоритмами, генетичний алгоритм більш гнучкий і може багаторазово змінювати параметри для більш прийнятних рішень.

Ці результати підтверджують ефективність та застосовність запропонованого генетичного алгоритму з методами локальної оптимізації для вирішення проблеми комівояжера. Ці висновки мають важливе значення для подальших досліджень у галузі проблеми комівояжера та пов'язаних областей.

ГЕНЕТИЧНИЙ АЛГОРИТМ, ЛОКАЛЬНА ОПТИМІЗАЦІЯ, ПЕРЕХРЕСТЯ, МУТАЦІЯ, ВИБІР.

ABSTRACT

Master's thesis: p. 64 p., Figure 10, table. 13, sources 9.

Relevance: The traveling salesman problem (TSP), which is a widely used combinatorial optimization problem in various industries, has a huge complexity that is simply not captured in its simple description, especially with a large number of cities. In order to overcome the challenges associated with solving the TSP problem, it is proposed to introduce a genetic algorithm into the solution process. Based on the principles of natural selection and the survival of the fittest in the biological world, the genetic algorithm is a global optimization algorithm that exhibits self-organization, adaptability, and self-learning. Within the framework of this study, a new optimization method combining genetic algorithm and local optimization is proposed for solving the TSP problem. Thorough performance analysis and benchmarking demonstrate the superiority of this method over traditional solution approaches.

Relevance: The traveling salesman problem (TSP), which is a widely used combinatorial optimization problem in various industries, has a huge complexity that is simply not captured in its simple description, especially with a large number of cities. In order to overcome the challenges associated with solving the TSP problem, it is proposed to introduce a genetic algorithm into the solution process. Based on the principles of natural selection and the survival of the fittest in the biological world, the genetic algorithm is a global optimization algorithm that exhibits self-organization, adaptability, and self-learning. Within the framework of this study, a new optimization method combining genetic algorithm and local optimization is proposed for solving the TSP problem. Thorough performance analysis and benchmarking demonstrate the superiority of this method over traditional solution approaches.

Object of research: This work consists in solving the traveling salesman problem (TSP) using genetic algorithm and local optimization.

The subject of the study: this study consists in the use of a genetic algorithm with the application of local optimization for solving the traveling salesman problem (TSP) and conducting a comparative analysis with traditional genetic methods.

Research methods: Theoretical and experimental.

Results: From the conducted study it was found that the combination of genetic algorithm with local optimization methods effectively solves the traveling salesman problem (TSP). In practical application, the genetic algorithm provides a stable approximate optimal solution. It should be noted that the initialization and tuning of genetic operators are key aspects of developing a genetic algorithm for the TSP task.

Compared to traditional algorithms, the genetic algorithm is more flexible and can repeatedly change the parameters for more acceptable solutions.

These results confirm the effectiveness and applicability of the proposed genetic algorithm with local optimization methods for solving the traveling salesman problem. In the paper, we discuss in detail the advantages of this method and its scope, and provide empirical analyzes and comparative results to support our conclusions. These findings are important for further research in the traveling salesman problem and related areas.

GENETIC ALGORITHM, RECURSION, LOCAL OPTIMIZATION, CROSSOVER, MUTATION, SELECTION.

ЗМІСТ

ВСТУП

Проблема комівояжера (TSP) відноситься до визначення оптимальної послідовності міст (найкращого маршруту) з набору можливих шляхів, що включають декілька міст. Це універсальне уявлення безлічі складних проблем, що виникають у різних галузях. У процесі вирішення таких завдань дослідники стикаються з труднощами щодо оптимальних рішень з використанням глобальних алгоритмів пошуку. Внаслідок цього для вирішення TSP застосовується генетичний алгоритм. Генетичний алгоритм - це орієнтований групу метод випадкового пошуку, заснований на теорії біологічної еволюції Дарвіна. Він імітує етапи біологічної еволюції та вводить в алгоритм такі поняття, як розмноження, схрещування, варіація, конкуренція та відбір. За допомогою рекомбінації набору можливих рішень рекомбінація покращуються здійсненні рішення, покращується траєкторія руху або тенденція здійснюваних рішень у багатовимірному просторі, і, нарешті, досягається оптимальне рішення.

Генетичний алгоритм – це механізм випадкового пошуку, заснований на теорії біологічної еволюції Дарвіна та орієнтований на групу. Алгоритм імітує етапи біологічної еволюції, такі як розмноження, схрещування, мутація, конкуренція та відбір. Рекомбінація набору можливих рішень дозволяє покращувати рішення, оптимізувати траєкторію руху чи тенденцію рішень у багатовимірному просторі і, зрештою, досягти оптимального рішення.

У цій статті основні ідеї та принципи оптимізації генетичного алгоритму застосовуються для побудови програми, що вирішує завдання TSP та знижує складність цього завдання. генетичного алгоритму для вирішення TS.

РОЗДІЛ 1

АНАЛІТИЧНИЙ ОГЛЯД

1.1 Біологічні засади генетичних алгоритмів

Основою генетичного алгоритму є принципи генетики, які у основі процесу еволюції біологічних організмів. Він запозичив ідеї "природного відбору" та "виживання найбільш пристосованих" з дарвінівської теорії еволюції, а також принципи молекулярної генетики Менделя, такі як "генетичний ген" та "гібридизація та мутація". У генетичному алгоритмі хороші індивіди розглядаються як "вижили", а погані індивіди відсіваються, потім шляхом схрещування та мутації кращих індивідів у популяції постійно створюються кращі індивіди, зрештою досягаючи мети оптимізації.

1.2 Розвиток та огляд генетичних алгоритмів

Походження генетичних алгоритмів можна простежити до початку 60-х років минулого століття, коли професор Джон Холланд (1 John Holland) з університету Мічигана першим запропонував ідею використання біологічних принципів, таких як схрещування та мутація генів, для вирішення завдань оптимізації в комп'ютерних системах. Він виявив, що ці принципи є важливими факторами, що забезпечують еволюцію нових видів у природі, у процесі дослідження складних систем.

З моменту виникнення генетичний алгоритм пройшов відносно стабільний період розвитку. Однак із початку 90-х років минулого століття генетичний алгоритм пережив період розквіту, ставши як у теоретичних дослідженнях, так і в прикладних дослідженнях дуже популярною темою. Особливо активно розвивається дослідження застосування генетичного алгоритму, тому що цей алгоритм є методом самоналаштування стохастичного пошуку, який не вимагає безперервності чи диференційованості об'єкта оптимізації і має високу стійкість і внутрішній

механізм паралельної обробки даних. Тому з поглибленням досліджень його сфера застосування постійно розширюється, а здатність генетичного алгоритму до оптимізації та навчання правил помітно зростає. Крім того, деякі нові теорії та методи також набули швидкого розвитку в прикладних дослідженнях, що, безумовно, додало генетичному алгоритму нову життєву силу. Він досяг чудових результатів у галузі машинного навчання, автоматичного управління, робототехніки, електроавтоматики, а також у галузі комп'ютерів та зв'язку.

РОЗДІЛ 2

ХАРАКТЕРИСТИКА ГЕНЕТИЧНОГО АЛГОРИТМУ

2.1 Основна ідея генетичного алгоритму

Основна ідея генетичного алгоритму полягає в тому, щоб почати з вирішення проблеми та закодувати деякі можливі рішення у вигляді індивідів (хромосом) популяції. Функція оцінки здатності індивідів адаптації до довкілля є цільової функцією проблеми. Для напряму навчання та визначення напряму пошуку використовуються генетичні оператори, які імітують гібридизацію, мутацію та реплікацію в генетиці. Використовується принцип природного відбору, щоб виробляти еволюцію популяції, яка виробляє популяції з вищими середніми показниками адаптації та кращими індивідами. Після кількох поколінь вибирається найкращий адаптований індивід, який є оптимальним або приблизно оптимальним вирішенням проблеми.

2.2 Порівняння генетичних алгоритмів коїться з іншими традиційними методами пошуку

Порівняння генетичних алгоритмів відбувається з іншими традиційними методами пошуку:

1. Глобальна здатність пошуку:

Генетичний алгоритм може шукати по всьому простору пошуку та не обмежений локальним простором пошуку, тому він має глобальну здатність пошуку.

2. Здатність до паралельних обчислень:

Оскільки кожен індивід (хромосома) в генетичному алгоритмі може обчислювати свою пристосованість незалежно, може бути добре використана здатність паралельних обчислень.

3. Адаптивність функції пристосованості:

Від функція пристосованості генетичного алгоритму не вимагає бути функцією, що диференціюється, вона може бути нелінійною і складною функцією, здатною адаптуватися до вирішення різних завдань.

4. Висока ефективність пошуку:

Генетичний алгоритм має високу ефективність пошуку, він може швидко знайти глобальний оптимум або наближений глобальний оптимум.

5. Немає попередніх передумов:

Генетичний алгоритм не вимагає будь-яких попередніх передумов щодо простору пошуку та може адаптуватися до оптимального вирішення задачі.

2.3 Принципи та проектування генетичних алгоритмів. Основні принципи генетичних алгоритмів

Генетичний алгоритм є евристичним оптимізаційним алгоритмом, який моделює принципи наслідування та еволюції генів у процесі біологічної еволюції. На відміну від традиційних алгоритмів пошуку, генетичний алгоритм починає процес пошуку із групи випадково згенерованих початкових рішень, що утворюють популяцію. Кожен індивід у популяції представляє вирішення проблеми у вигляді хромосоми, яка поступово еволюціонує у наступних ітераціях, які називаються поколіннями. Генетичний алгоритм здійснює операції кросовера, мутації та відбору для генерації наступного покоління хромосом, званих нащадками. Якість хромосом оцінюється за допомогою функції пристосованості. Виходячи з розміру пристосованості, вибирається певну кількість індивідів із попереднього та поточного покоління для формування наступного покоління популяції, яке продовжує еволюцію. Таким чином, після кількох поколінь алгоритм сходиться до кращої хромосоми, яка, ймовірно, є оптимальним або субоптимальним вирішенням проблеми.

2.3.1 Проектування генетичного алгоритму

Генетичний алгоритм включає наступні п'ять елементів:

1. кодування параметрів;
2. встановлення початкової популяції;
3. проектування функції пристосованості;
4. проектування генетичних операцій;
5. встановлення контрольних параметрів (переважно які стосуються розміру популяції та ймовірності використання генетичних операцій тощо.).

2.3.1.1 Генетичне кодування

Кодування це метод переведення допустимих розв'язків проблеми з її простору рішень у простір пошуку, який може бути оброблений генетичним алгоритмом. Кодування є головною проблемою, яку необхідно вирішити під час застосування генетичного алгоритму, а також є ключовим кроком у проектуванні генетичного алгоритму. Метод кодування визначає як форму розташування хромосом індивідуумів, але й визначає метод декодування під час перекладу генотипу індивідуума з простору пошуку фенотип рішення у просторі рішень. Метод кодування також впливає на методи роботи генетичних операторів, таких як кросинговер, мутація тощо.

Однією з труднощів при застосуванні генетичних алгоритмів є розробка ідеальної схеми кодування для конкретної проблеми. На даний момент не існує чіткої та повної теорії керівництва та критеріїв оцінки, які можуть допомогти нам розробити кодувальну схему. Тим не менш, De Jong [4] запропонував два оперативно сильні принципи кодування:

1. Принцип кодування осмислених будівельних блоків: слід використовувати кодувальні схеми, які можуть легко породжувати гени, що мають низький порядок та коротку довжину визначення, пов'язані з проблемою, яку ми вирішуємо.

2. Принцип кодування мінімального набору символів: слід використовувати схеми кодування, які можуть натурально описувати або представляти проблему з використанням мінімального набору символів.

Ці два принципи кодування можуть забезпечити деяке керівництво при розробці схеми кодування. Конкретно, для першого принципу кодувальна схема повинна забезпечувати, щоб гени хромосоми особини мали сенс, пов'язаний з проблемою, наприклад, для вирішення проблеми TSP можна використовувати послідовність для представлення шляху, або для вирішення проблеми мінімізації булевої функції використовувати список послідовності булевих змінних. Крім того, визначення кодувальної схеми також має бути якомога коротшим, щоб зменшити обчислювальну складність алгоритму.

Для другого принципу кодувальна схема повинна дозволяти природне подання або опис проблеми, наприклад, використання двійкового числа для представлення цілого числа або використання букв для представлення рядка. Крім того, набір символів у кодувальній схемі повинен бути якнайменше, щоб зменшити розмір простору пошуку та підвищити ефективність пошуку.

2.3.1.2 Встановлення початкової популяції

Генетичний алгоритм працює з населенням, що складається з безлічі особин. Одним із завдань після кодування є встановлення початкової популяції, на основі якої відбувається еволюція через покоління доти, доки не буде досягнута зупинна точка за певним критерієм. Таким чином, виходить останнє покоління особин, яке є вирішенням задачі.

При встановленні розміру популяції слід враховувати два фактори:

1. Налаштування початкової популяції:

(1) На основі вбудованих знань про простір оптимальних рішень встановлюється початкова популяція всередині діапазону розподілу рішень. Спочатку створюються випадкові особини, потім вибираються найкращі з них і додаються в початкову популяцію. Цей процес повторюється до того

часу, поки число особин у початковій популяції досягне заданого значення.

2. Як підтримувати розмір популяції кожному поколінні у процесі еволюції?

У процесі еволюції підтримання розміру популяції є ключовим завданням. Якщо розмір популяції занадто малий, це може призвести до збіжності алгоритму до підоптимального рішення або раннього насичення. Якщо розмір популяції занадто великий, це може призвести до значного збільшення обчислювальної складності або уповільнення швидкості збіжності алгоритму. Зазвичай розмір популяції повинен бути досить великим, щоб забезпечити повне покриття простору пошуку і уникнути застрягання в локальному оптимумі. У той же час, розмір популяції повинен бути якнайменше, щоб знизити обчислювальну складність і прискорити збіжність алгоритму.

Зазвичай розмір популяції може бути визначений на основі таких факторів:

(1) Складність проблеми: що складніша проблема, то більший розмір популяції потрібно повного охоплення простору пошуку;

(2) Ресурси обчислювальної потужності: що більше ресурсів доступно, то більший розмір популяції можна використовуватиме прискорення збіжності алгоритму;

Час виконання генетичного алгоритму: якщо час дозволяє, можна використовувати менший розмір популяції і збільшити час виконання генетичного алгоритму ширшого пошуку;

(3) Характер проблеми: деякі специфічні типи проблем можуть потребувати використання певного розміру популяції. Наприклад, для завдань безперервної оптимізації зазвичай використовуються більш малі розміри популяції, а для завдань комбінаторної оптимізації - більші.

2.3.1.3 Функція пристосованості

При вивченні спадковості та еволюції в природі біологи

використовують термін "фітнес" для вимірювання ступеня пристосованості певного виду до свого середовища проживання. У видів з більш високою пристосованістю до довкілля буде більше можливостей для розмноження, у той час як у видів з більш низькою пристосованістю до довкілля буде відносно менше можливостей для розмноження, навіть до того, як вони можуть поступово вимирати. Тобто ймовірність того, що особина з вищим фітнесом успадкує свої гени в наступне покоління, відносно вища, а ймовірність того, що особина з більш низьким фітнесом успадкує свої гени в наступне покоління, відносно менша. Функція, яка вимірює фітнес особи.

Розмір пристосованості особини визначає можливість її успадкування у наступному поколінні популяції. Генетичний алгоритм використовує лише значення цільової функції для отримання інформації про пошук на наступному етапі. Використання значення цільової функції проявляється через оцінку пристосованості особини.

Процес оцінки пристосованості особин включає наступні три кроку:

1. Декодування: для генетичних алгоритмів або інших еволюційних алгоритмів генотип особини зазвичай представляється у вигляді бінарного кодованого рядка, який необхідно перетворити на фенотип, тобто відповідні чисельні значення або конкретний спосіб реалізації.

2. Обчислення значення цільової функції: для декодованого фенотипу особини, відповідно до конкретних вимог завдання, може бути обчислено значення цільової функції. Цільова функція зазвичай включає в себе мету оптимізаційної задачі та обмеження.

3. Обчислення пристосованості: на основі значення цільової функції обчислюється пристосованість особи відповідно до певних правил перетворення. Різні завдання можуть використовувати різні методи обчислення пристосованості, такі як лінійне масштабування, експоненціальне масштабування тощо.

Масштабування пристосованості (fitness scaling) - це підхід, у якого пристосованість особин змінюється шляхом збільшення чи зменшення її

значень з допомогою певних перетворень. Існують три методи масштабування пристосованості особин:

Лінійне масштабування, формула якого виглядає так:

$$\mathrm{F}' = \mathrm{a}F + b \tag{2.1}$$

де F - вихідна пристосованість;

F'- нова пристосованість після масштабування;

a и b - коефіцієнти.

Масштабування з використанням ступеня, формула якого виглядає так:

$$F' = Fn \tag{2.2}$$

де n – показник ступеня залежить від розв'язуваного завдання.

Масштабування з використанням експонентів, формула якого виглядає наступним чином:

$$F' = exp(-\beta F) \tag{2.3}$$

де β – коефіцієнт визначає ступінь примусу до вибору.

Масштабування пристосованості особин – це техніка, що використовується в генетичних алгоритмах для коригування розподілу значень пристосованості, з метою ефективнішого дослідження простору пошуку та пошуку оптимальних рішень. Основна перевага масштабування пристосованості полягає в тому, що воно дозволяє відобразити значення пристосованості у відповідний діапазон значень, що полегшує пошук оптимальних рішень.

У процесі оцінки пристосованості обчислення пристосованості є важливою основою для вибору та еволюції особин, і зазвичай особини з більш високою пристосованістю більш імовірно будуть обрані та збережені у наступному поколінні. Тому точність та надійність оцінки пристосованості мають важливе значення для продуктивності та швидкості збіжності алгоритму.

2.3.1.4 Операції генетичних алгоритмів

Генетична операція – це операція, що моделює генетичне успадкування біологічних організмів. У генетичному алгоритмі після кодування початкової популяції завдання генетичної операції у тому, щоб виконати певні операції над особинами популяції залежно від своїх адаптації до довкілля (оцінка придатності), щоб реалізувати процесс еволюції « виживання найкращих « . З точки зору оптимізації пошуку генетичні операції можуть оптимізувати розв'язання задачі, покращуючи його з покоління в покоління та наближаючись до найкращого рішення. Операції генетичного алгоритму включають три основних генетичних оператора: вибір (selection); кросовер (crossover); мутація (Mutation). Ці три генетичні оператори мають такі характеристики:

Генетичні операції є випадковими операціями, тому правила міграції особин до оптимального рішення у популяції є випадковими;

1. Ефект генетичних операцій тісно пов'язаний з ймовірністю виконання цих трьох генетичних операторів, методом кодування, розміром популяції, початковою популяцією та функцією придатності;
2. Методи чи стратегії виконання трьох основних генетичних операторів безпосередньо пов'язані з методом кодування особин.

Оператор оператора вибору, також відомий як оператор відтворення, відіграє важливу роль генетичних алгоритмах, визначаючи, які особини з популяції будуть обрані передачі у наступне покоління. Особи з більш високим рівнем пристосованості мають більшу ймовірність бути обраними, у той час як особини з нижчим рівнем пристосованості мають меншу ймовірність. Операції вибору засновані на оцінках пристосованості особин і спрямовані на запобігання втраті генетичної інформації, покращення глобальної збіжності та підвищення обчислювальної ефективності. Три найбільш поширені методи вибору включають:

1. Вибір на основі рулетки: цей метод полягає в присвоєнні

ймовірностей вибору особин пропорційно їх рівню пристосованості, розділяючи рулетку на n секторів. Потім генерується n випадкових чисел від 0 до 1, що відповідають обертанням n рулетки. Особина, пов'язана з сектором, на якому зупиняється покажчик, вибирається.

Стратегія елітизму: протягом еволюційного процесса генетичного алгоритму нові особини створюються за допомогою генетичних операцій, таких як схрещування та мутація. Незважаючи на те, що ці операції можуть призвести до більш пристосованих осіб, вони також можуть випадково знищити найкращі особини в поточній популяції. Стратегія елітизму використовується для збереження найкращих особин від одного покоління до іншого.

2. Вибір на основі ранжирування: у цьому методі особини сортуються відповідно до їх рівня пристосованості зі спадання. Потім ймовірності вибору призначаються особинам на основі їхнього рангу в популяції.

Проектування оператора кросовера, ключову роль генетичному алгоритмі грає оператор кросинговера, який відповідає за рекомбінацію генетичного матеріалу двох батьківських особин до створення нової особини. Кросинговер значно підвищує можливості пошуку у генетичному алгоритмі. Оператор кросинговерара зазвичай включає наступні етапи:

1. Випадковий вибір пари особин із популяції для схрещування;
2. Випадковий вибір одного чи кількох цілих чисел k з діапазону [1, L-1], де L – довжина бітового рядка особини, як точки кросинговера;
3. Реалізація оператора кросинговера з ймовірністю Pc ($0 < Pc <= 1$): особини обмінюються частинами своїх структур у точках кросинговера, створюючи нову пару нащадків.

Основні оператори кросовера:

Найбільш поширеними операторами кросовера є: однокрапковий кросовер, багатокрапковий кросовер. Для різних способів кодування також

існують різні оператори кросовера, такі як часткове відображення кросовер, послідовний кросовер, циклічний кросовер та інші.

Нижче описуються два простих і поширених оператора кросинговера:

Одноточковий кронсинговерр, також називається простим кросинговер. Конкретний спосіб дії полягає в тому, що випадково задається точка кросинговера в генетичній послідовності. При виконанні кросинговера структури двох особин до або після цієї точки взаємно обмінюються, що призводить до створення двох нових особин.

Наприклад:

Відповідні особини:

Особина *а* 1001 | 111 ---------- 1001000 Нова особина А

Особина *б* 0011 | 000 ----------0011111 Нова особина Б

Тут точку кросинговера встановлено між четвертим і п'ятим геном. При кросинговері дві послідовності після цієї точки взаємно обмінюються. Таким чином, гени від першого до четвертого гена особини А і від п'ятого до сьомого гена особини Б утворюють нову особину. Також виходить нова особина В. Точка кросинговера задається випадковим чином, коли довжина хромосоми дорівнює n, можуть бути встановлені n-1 точок кросинговера, тому одноточковий кросинговер може реалізувати n-1 різних результатів кросинговера.

Дії при двоточковому кросинговері аналогічні одноточковому кросинговера, тільки тут випадково задаються дві точки кросонвера, і гени між ними змінюються місцями.

Наприклад:

Відповідні особини:

Особина *а* 10 | 110 | 11 ----------1001011 Нова особина А

Особина *б* 00 | 010 | 00 ----------0011000 Нова особина Б

Основні принципи проектування оператора консинговера наступні:

1. Кожен оператор кросинговера повинен задовольняти критерію

оцінки, тобто гарантувати, що чудові характеристики особин попереднього покоління успадковуватимуться на нових особинах наступного покоління;

2. Для забезпечення відповідності цьому критерію необхідно узгодити дизайн оператора кросинговера та кодування;

3. Для бінарного кодування, яке є основним у генетичних алгоритмах, всі оператори кросинговера включають два основні елементи:

(1) Випадковий вибір пар особин з популяції за допомогою операції вибору, а потім ухвалення рішення про необхідність виконання оператора кросинговера на основі заздалегідь заданої ймовірності;

(2) Встановлення точок кросинговера та обмін частинами структури (або генів) пар особин перед і після точок кросинговера.

Дизайн оператора мутації：

Мутаційний оператор – це операція в генетичному алгоритмі, що імітує явище мутації гена на одному з хромосом у біологічній еволюції. У генетичному алгоритмі мутаційний оператор реалізується шляхом випадкового інвертування двійкового символу аллельного гена з ймовірністю мутації Pm. Основні кроки мутаційної операції наступні:

1. Випадково вибираються генні локуси в діапазоні двійкових рядків всіх особин у популяції;

2. Значення генів цих локусів мутуються з ймовірністю мутації Pm.

Введення мутаційного оператора в генетичний алгоритм має дві основні цілі. З одного боку, може забезпечити локальний випадковий пошук, допомогти уникнути застрягання в локальному оптимумі. Коли генетичний алгоритм наближається до оптимальної області через кросовер, використання цієї можливості випадкового локального пошуку за допомогою мутаційного оператора може прискорити збіжність до оптимального рішення. З іншого боку, мутаційний оператор може відновити втрачену інформацію про алелі у процесі еволюції популяції, щоб зберегти індивідуальну різницю у популяції.

Розробка параметрів роботи:

Для генетичних алгоритмів необхідно вибрати наступні параметри виконання: довжину кодуючого рядка L, розмір популяції M, ймовірність кросинговера Pc, ймовірність мутації Pm, кінцеве покоління T та покоління G.

1. Довжина кодуючого рядка L: у разі бінарного кодування та кодування дійсних чисел довжина кодуючого рядка залежить від точності вирішення проблеми, зазвичай вона встановлюється відповідно до діапазону значень та вимог точності проблеми. Для символьних змінних та змінних із змінною довжиною кодування має бути налаштовано відповідно до конкретного методу кодування;

2. Розмір популяції M: вибір розміру популяції повинен враховувати швидкість обчислень та ефективність пошуку. Якщо розмір популяції занадто малий, то популяція матиме мало різноманітності, що може призвести до передчасної збіжності та зниження випадковості пошуку. Якщо розмір популяції дуже великий, це може вплинути на швидкість обчислень. Зазвичай рекомендується вибирати розмір популяції в діапазоні від 20 до 100;

3. Імовірність кросинговера Pc: операція кросинговера є одним з основних способів генерації нових особин у генетичних алгоритмах, адекватна ймовірність кросинговера може збільшити різноманітність популяції та стимулювати пошук. Зазвичай рекомендується вибирати значення в діапазоні від 0,4 до 0,99, конкретне значення має бути налаштовано відповідно до характеристик проблеми;

Імовірність мутації Pm: операція мутації може уникнути попадання генетичного алгоритму локальний оптимум і збільшити різноманітність популяції. Однак надто висока ймовірність мутації може зруйнувати добрі моделі популяції та знизити ефективність пошуку. Зазвичай рекомендується вибирати значення в діапазоні від 0,001 до 0,1, конкретне значення має бути налаштовано відповідно до характеристик пр;

4. Завершальна епоха T: пошук генетичного алгоритму вимагає

кількох ітерацій, завершальна епоха визначає часові рамки пошуку. Зазвичай рекомендується вибирати значення між 1000 і 2000, конкретне значення має бути налаштовано відповідно до особливостей конкретної проблеми;

5. Епоха G: епоха G означає відсоток замінюваних особин у кожному поколінні популяції, тобто у кожному поколінні замінюється M × G особин. Розумний вибір епохи G може збалансувати швидкість та ефективність пошуку. Зазвичай рекомендується вибирати значення між 0,2 та 0,8, конкретне значення має бути налаштовано відповідно до особливостей конкретної проблеми.

2.4 Кроки операції генетичного алгоритму

Генетичний алгоритм – це метод оптимізації, який імітує природний процесс еволюції на вирішення завдань. Його кроки включають:

1. Ініціалізація популяції: випадкова генерація початкової популяції відповідно до обмежень завдання. Кожен індивід є потенційним розв'язанням задачі.

Оцінка придатності: обчислення значення функції придатності кожного індивіда, що відбиває його здатність розв'язати задачу. Чим вище значення функції придатності, тим більше шансів, що індивід є найкращим рішенням.

2. Операція вибору: вибір кількох кращих індивідів у порядку зменшення значення функції придатності, які будуть використовуватися як «батьки» для генерації наступного покоління індивідів.

3. Операція схрещування: вибір двох індивідів-«батьків» із попереднього покоління та створення двох нових індивідів-«нащадків» шляхом комбінації генетичних матеріалів батьків. Це підвищує різноманітність популяції.

4. Операція мутації: зміна випадкового гена в геномі нового індивіда з деякою ймовірністю, щоб додатково підвищити різноманітність популяції.

5. Оновлення популяції: змішування нових «нащадків» з «батьками» для створення нової популяції.

6. Визначення критерію зупинки: якщо було досягнуто заздалегідь визначеного критерію зупинки (наприклад, максимальна кількість ітерацій або наближення до оптимального рішення), алгоритм завершує роботу і виводить остаточний результат. В іншому випадку алгоритм повертається до кроку 2.

Генетичний алгоритм – це операція типу популяції, що з кожним індивідуумом у популяції. Вибір, схрещування та мутація є трьома основними операторами генетичного алгоритму, які утворюють так звані генетичні операції, що дає генетичного алгоритму властивості, відсутні в інших традиційних методів.

РОЗДІЛ 3

ОПИС ПРОБЛЕМИ КОМІВОЯЖЕРА (TSP)

3.1 Математичний опис та модель завдання комівояжера (Traveling Salesman Problem, TSP)

TSP (Проблема комівояжера) може бути описана наступним чином: відомі відстані між n містами, один комівояжер починає свій маршрут з якогось міста, відвідуючи кожне місто рівно один раз і повертаючись до початкового міста. Який розклад слід скласти, щоб зменшити довжину його маршруту? Іншими словами, якщо є граф G = (V, E), де V - це безліч вершин, а E - безліч ребер, і D = (dij)

- матриця відстаней між вершинами i і j, то TSP полягає в тому, щоб знайти найкоротший цикл Гамільтона, який проходить через всі вершини рівно один раз.

Наступним чином:

Для цього безлічі міст V={v1, v2, …, vn}, кожна пара міст має відстань dij (або вартість, час, довжину шляху тощо). Потрібно знайти порядок відвідування міст T={t1, t2, …, tn} де ti∈V (i=1~n) і tn+1=tl (t1 - початкова точка, tl - кінцева точка), щоб мінімізувати сумарну довжину шляхи.

Математична модель може бути представлена таким чином:

Цільова функція:

$$\min_{T\in\Omega} \sum_{i=1}^{n} d_{t_i t_{i+1}} \qquad (4.1)$$

де Ω – це безліч всіх можливих циклів, що складаються з n міст, що не повторюються.

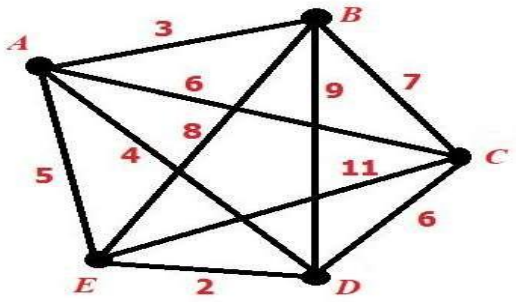

Рисунок 1 – Приклад постановки завдання комівояжера [3]

3.2 Обчислювальна складність завдання комівояжера (TSP)

Загальне формулювання завдання комбінаторної оптимізації: нехай задана функція f:S -> R, де S - кінцева множина. Для кожного x з S функція f(x) визначає єдине речове число, тоді завдання на мінімум (максимум) полягає в тому, щоб знайти x^' з S, таке для будь-якого x з S виконано f(x^') ≤ (≥) f(x). Один з основних алгоритмів для вирішення цієї задачі - для кожного x з S обчислити значення f(x), вибрати x^', таке, що для всіх x з S виконано f(x^') ≤ (≥) f(x).

Це метод повного перебору, який теоретично може вирішувати будь-які завдання комбінаторної оптимізації з кінцевим безліччю. Але для TSP з n містами може бути до (n-1)!/2 маршрутів і обчислення кожного маршруту вимагає підсумовування n відстаней, тому кількість обчислень буде пропорційно n!/2. Таблиця 1 показує час, необхідний обчислення TSP методом повного перебору на суперкомп'ютері CrayXT3 зі швидкістю обробки 105 мільярдів операцій на секунду.

Таблиця 1 – Обчислювальна складність TSP

Кількість міст N	Кількість циклів (n-1)!/2	Кількість додань n!/2	Час пошуку
5	12	60	6×10^{-12} Секунда
10	1.8144×10^{5}	1.8144×10^{6}	1.8144×10^{-7} Секунда
20	6.08×10^{16}	1.22×10^{18}	1.22×10^{5} Секунда
40	1.02×10^{46}	4.08×10^{47}	1.32×10^{27} Року
100	4.6×10^{155}	4.6×10^{157}	1.48×10^{136} Року
200	5×10^{371}	10^{374}	3.22×10^{356} Року

3.3 Теоретичне значення та прикладна цінність TSP

TSP - це класичне завдання комбінаторної оптимізації, яка має важливе теоретичне значення та широку практичну застосовність у галузі комп'ютерних наук та математики. TSP є одним із NP-складних завдань, тому алгоритми пошуку оптимального рішення можуть займати багато часу, проте багато наближених алгоритмів можуть бути використані для вирішення практичних завдань. Застосування TSP включає такі області, як логістика, планування транспортних маршрутів, виробничий сектор, проектування електричних схем, секвенування ДНК та інші. Розв'язання задачі TSP дозволяє підвищити ефективність роботи, знизити витрати та збільшити використання ресурсів.

3.4 Опис основних алгоритмів розв'язання задачі TSP

Розв'язання задачі TSP має багато алгоритмів. Ось короткий опис деяких основних алгоритмів:

1. Алгоритм перебору:

Це наївний метод, який перебирає всі можливі маршрути та вибирає найкоротший. Однак, оскільки завдання TSP є NP-важкою, цей алгоритм застосовується тільки до невеликих завдань.

2. Жадібний алгоритм:

Жадібний алгоритм будує маршрут, вибираючи найближче невідвідане місто. Цей алгоритм легкий у реалізації, але не гарантує оптимального рішення.

3. Алгоритм гілок та кордонів:

Цей алгоритм шукає всі можливі маршрути шляхом пошуку із розгалуженням та обмеженням кордону, і в процесі пошуку обрізає гілки для оптимізації ефективності. Цей алгоритм може дати оптимальне рішення, але у реальних додатках обчислювальна вартість висока.

4. Генетичний алгоритм:

Генетичний алгоритм використовує імітацію процесу природного відбору для пошуку найкоротшого маршруту. Цей алгоритм зазвичай використовує випадковий пошук для отримання кращих результатів і може застосовуватись до великим завданням.

3.5 Поширені методи кодування для вирішення задачі комівояжера (TSP)

Зазвичай для вирішення проблеми TSP використовуються два поширені методи кодування: матричне подання суміжності та подання шляху. Нижче ми коротко розглянемо ці два методи.

1. Нотація суміжної матриці (Adjacency Matrix Representation):

Використовується масив для позначення номерів усіх міст, а матриця використовується для зберігання відстаней між ними. Наприклад, якщо є n міст, то відповідна матриця матиме розмірність n x n, де i-я рядок і j-й стовпець позначають відстань між i-м та j-м містами. Цей метод кодування простий і зрозумілий, підходить для вирішення малих завдань TSP.

2. Нотація шляху (Path Representation):

Шлях закодований як послідовності, кожен елемент якої позначає номер відвіданого міста. Перший і останній елементи послідовності збігаються, позначаючи місто, з якого починається і який закінчується шлях. Наприклад, послідовність довжиною n+1 {1, 2, 3, ..., n, 1} означає шлях, який починається і закінчується в місті 1, і відвідуються всі міста. Цей метод кодування підходить для вирішення задач TSP великого розміру.

РОЗДІЛ 4

ЗАСТОСУВАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ ДО ПРОБЛЕМИ TSP

4.1 Схема кодування

Для n числа міст хромосома є перестановку цілих чисел від 1 до n, де кожне число позначає місто і порядок чисел у перестановці визначає порядок відвідування міст комівояжером. Наприклад, для 8 міст хромосома може виглядати як 8 → 1 → 2 → 4 → 3→ 6→ 7 → 5, що означає, що комівояжер починає в 8-му місті, потім відвідує 1-й, 2-й, 4-й і так далі, поки не повернеться до 5-го міста.

Наприклад:

Рисунок 2 – Приклад кодування міст

4.2 Створення початкової популяції

Для визначення розміру популяції на основі заданих параметрів, необхідно враховувати, який рівень різноманітності хочемо отримати в початковій популяції. Зазвичай, розмір популяції варіюється від кількох десятків до кількох сотень особин.

Для створення початкової популяції за допомогою випадкової генерації можна використовувати алгоритм, який генерує випадкові перестановки генів у кожному рядку геному, щоб отримати різні послідовності відвідування міст. Потім, щоб гарантувати, що кожне місто відвідується лише один раз у кожному рядку геному, можна перевірити, що кожне місто зустрічається лише один раз у кожній перестановці.

Для створення матриці популяції необхідно визначити розміри матриці S*T, де S - це розмір популяції, а T - це довжина геному кожного індивідуума. В даному випадку, довжина геному дорівнюватиме кількості

міст плюс один ген, який представляє загальну відстань, пройдену через ці міста.

Наприклад, початкової популяції розміру 10x9 для заданих параметрів популяції (розмір популяції = 10, кількість міст = 8)

Таблица 2 – Матриця кодування генів 10x9

Gen1	Gen2	Gen3	Gen4	Gen5	Gen6	Gen7	Gen8	Total
1	5	4	7	3	6	2	8	287
7	1	8	3	2	6	4	5	296
1	2	5	3	7	6	4	8	258
2	6	8	4	3	5	1	7	296
4	3	8	2	1	5	7	6	251
7	3	4	2	8	1	5	6	286
2	8	7	6	3	1	5	4	314
1	5	8	6	2	7	4	3	292
3	1	4	2	6	8	5	7	275
5	1	6	2	3	7	8	4	290

4.3 Функція пристосованості

У вирішенні завдання TSP можна використовувати функцію вартості або відстань як придатність для оцінки оптимальності результату. У матриці pop кожен рядок є можливим рішенням, тобто шлях, що починається з початкової точки, що проходить через усі міста і повертається в початкову точку, де останній елемент представляє сумарну відстань. Ця відстань використовується як придатність. Ми можемо визначити функцію для обчислення загальної відстані кожного рішення, наприклад:

$$Popi,-1 = \sum_{i=0}^{n} d\,(Popi,j,Popi,j+1) + d(Popi,N-1,Popi,0) \quad (4.1)$$

де $pop_{i,j}$ – елемент у і-му рядку і j-му стовпці матриці, що представляє j-е місто, відвідане і-м індивідуумом.

$d(pop_{i,j}, pop_{i,j+1})$ – відстань між містом j та містом j+1.

$d(pop_{\mathrm{i,N-1}},\ pop_{\mathrm{i,0}})$ – відстань між останнім відвіданим містом та початковою точкою, щоб закінчити цикл.

Для кожного рішення ми перебираємо всі міста, що його містяться, обчислюємо відстань від поточного міста до наступного міста і додаємо їх у загальну суму, отримуючи тим самим загальну відстань. Нарешті потрібно додати відстань від останнього міста до початкового пункту. Після обчислення загальної відстані кожного рішення ми використовуємо його як придатності рішення, тобто відстань стає придатністю рішення. Таким чином, рішення з більш короткою відстанню матимуть більш високу придатність, оскільки вони більш наближені до оптимального рішення.

4.4 Селекція (Відбір)

Операція вибору спрямовано вибір особин з високою придатністю з популяції як батьківських особин наступного покоління. Часто використовувані стратегії вибору включають вибір на основі рангу, вибір методом рулетки, вибір методом змагання і т.д.

Ці стратегії вибору дозволяють на основі значень пристосованості (fitness) відібрати з певною ймовірністю батьківські особини для наступного покоління, зберігаючи кращі особини та відкидаючи менш пристосовані, тим самим поступово підвищуючи пристосованість популяції.

4.4.1 Метод елітаризму (Elitism)

Метод рангового відбору (Rank Selection) - це стратегія вибору генетичному алгоритмі, заснована на ранжируванні особин за рівнем пристосованості. Спочатку кожен індивід у популяції ранжується порядку спадання його пристосованості, потім визначається можливість вибору кожного індивіда як батьківського наступного покоління, пропорційно його рангу, тобто. індивіди з вищим рангом мають велику можливість вибору. Перевагою рангового відбору є зменшення впливу відмінностей у

пристосованості до ймовірності вибору, що збільшує можливості вибору кожної особини. Однак ранговий відбір може призвести до нерівномірного розподілу ймовірностей вибору, що може зменшити швидкість збіжності алгоритму. Тому зазвичай до рангового відбору додаються інші модифікації, щоб збалансувати його ефекти і зберегти оптимальну швидкість і точність вибору.

4.4.2 Вибір методом рулетки (Roulette Wheel Selection)

Вибір методом рулетки (Roulette Wheel Selection) - це стратегія вибору генетичних алгоритмах, заснована на ймовірності вибору певного рішення, пропорційної його пристосованості у популяції. Більш пристосовані рішення мають більшу можливість бути обраними, ніж менш пристосовані рішення. Це досягається шляхом подання кожного рішення у вигляді сектора на колесі рулетки, розмір якого пропорційний його пристосованості. Потім колесо рулетки обертається, і рішення, вибране під час зупинки колеса, використовується для створення наступного покоління рішень.

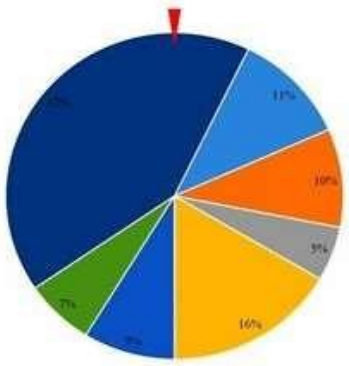

Рисунок 3 – Розподіл ймовірностей пристосованості особин на колесі рулетки.

4.4.3 Турнірний відбір（Tournament Selection）

При турнірної селекції всі особи популяції розбиваються на підгрупи з наступним вибором у кожному їх особини з найкращою пристосованістю. Підгрупи можуть мати довільний розмір, але найчастіше населення

поділяється на підгрупи по 2-3 особини у кожній. Турнірний метод придатний вирішення завдань як максимізації, і мінімізації функції. У турнірному методі допускається зміна розміру підгруп, на які підрозділяється населення (tournament size). На рисунку 4 представлена схема, яка ілюструє метод турнірної селекції для підгруп, що складаються з двох особин. Таку схему легко узагальнити на підгрупи більшого розміру.

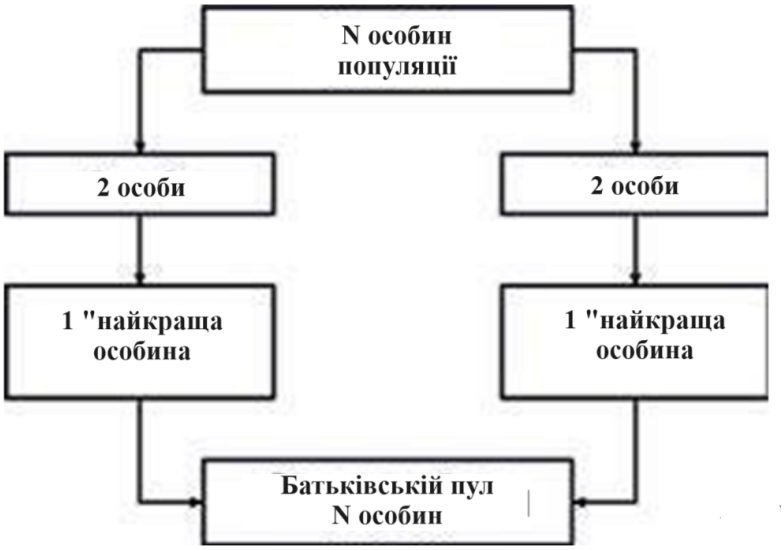

Рисунок 4 – Схема турнірної селекції для підгруп, що складаються із двох особин [3]

4.5 Кросовер (Схрещування)

Дизайн оператора кросовера безпосередньо впливає на швидкість збіжності популяції та досягнення оптимального рішення. У генетичному алгоритмі оператор кросовера виконує заміну та рекомбінацію пари батьківських особин із ймовірністю кросовера (Pc) для створення нових нащадків. У завданнях, пов'язаних з поданням шляхів, таких як завдання комівояжера, оператор кросовера повинен враховувати зв'язність та унікальність шляхів. Широко поширені такі оператори кросовера, що використовуються для подання шляхів: часткове відображення, впорядкований, циклічний та інші види.

4.5.1 Частково зіставлений кросовер (Partially-Mapped Crossover, PMX)

Partially – Mapped Crossover. Був запропонований Голдбергом та Лінгле. Після рівномірного вибору двох довільних точок розрізу батьківських хромосом, частини між точками розрізу одного з батьків (Батьки 1 та Батьки2) відображаються на рядок іншого з батьків, а гени, що залишилися, обмінюються. Розглянемо, наприклад, дві батьківські хромосоми (точки розрізу позначимо як «|»):

Батьки 1: 1 2 3 |4 5 6 | 7 8

Батьки 2: 3 7 5 |1 6 8 | 2 4

Відображення: 4 ↔1, 5 ↔ 6 , і 6 ↔ 8. Далі, підрядок 1-го батька копіюється 2-му нащадку (з дотриманням позицій генів), а підрядок 2-го батька копіюється 1-му нащадку.

Нащадок 1: * * * |1 6 8 | **

Нащадок 2: * * * |4 5 6 | **

Нащадок і, де і = 1, 2 заповнюється шляхом копіювання генів і-го батька. У разі, якщо місто вже присутнє в і-му нащадку, воно замінюється відповідно до відображення. Наприклад, перший елемент 1-го нащадка дорівнюватиме 1, як і 1-й елемент першого родителя. Однак елемент, значення якого дорівнює 1, вже міститься у нащадку. Отже, застосовуючи відображення (1 → 4) встановлюємо 1-м елементом 1-го нащадка ген, рівний 4 Другий, третій та сьомий елементи 1-го нащадка можна запозичити від першого родителя. Однак останнім елементом 1-го нащадка буде 8, яке вже є у ньому. Застосувавши відображення (8 → 6 → 5), отримаємо значення, що дорівнює 5 (рисунок 5а).

Нащадок 1: 4 2 3 |1 6 8 | 7 5

Для формування 2-го нащадка необхідно виконати аналогічні дії (див. рисунок 5б).

Нащадок 2: 3 7 8 |4 5 6 | 2 1

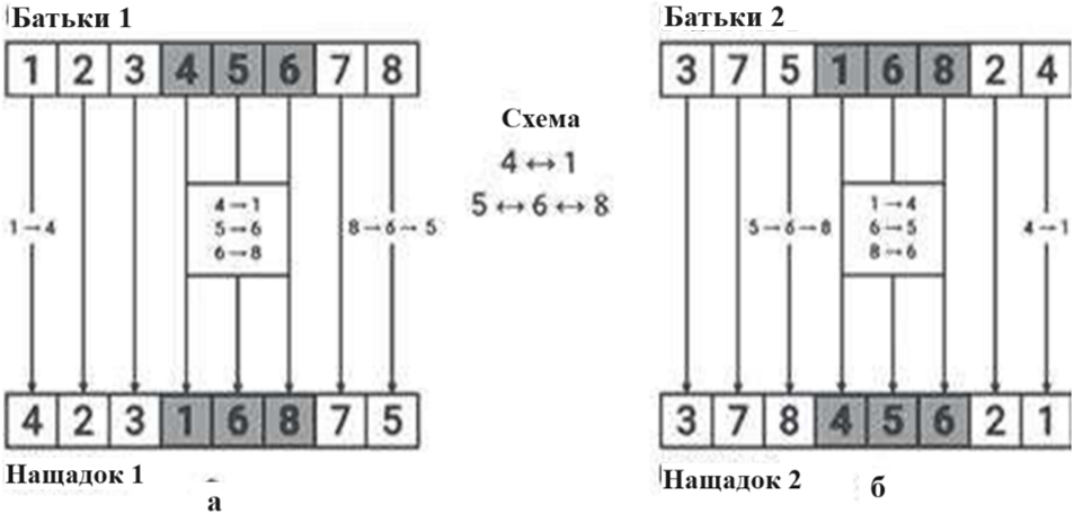

Рисунок 5 – Partially – Mapped Crossover [3]

4.5.2 Упорядкований кросовер (Order Crossover OX)

Ordered Crossover (OX) оператор є класичним алгоритмом кросинговеру генів, який застосовується до проблем, що мають обмеження на порядок, таким як завдання комівояжера (TSP).

Кроки OX-оператора наступні:

Випадковим чином вибираються два батьківські елементи для кросинговера, припустимо, що ці два батьківські елементи називаються Батьки 1 и Батьки 1.

1. Вибирається випадкова область кросинговера, і генетичний фрагмент усередині цієї області копіюється з Батьки 1 у відповідне положення першого нащадка.

2. Починаючи з першої позиції Батьки 2, додаються гени, які не були скопійовані в перший нащадок, в тому самому порядку, поки перший нащадок не буде заповнений.

3. Те саме робиться для другого нащадка.

Нижче наведено конкретний приклад:

Припустимо, что Батьки 1 и Батьки 2 мають такі значення:

Батьки 1: 1 2 3 4 5 6 7 8

Батьки 2: 4 6 8 2 1 3 5 7

Вибирається випадкова область кросинговера між 3-ю та 6-ю позицією Батьки1и Батьки 2:

Батьки 1: 1 2 3 | 4 5 6 | 7 8

Батьки 2: 4 6 8 | 2 1 3 | 5 7

Цей фрагмент копіюється у перший нащадок:

Нащадок 1: * * * | 4 5 6 | * *

Потім гени, що залишилися, і Батьки 2 вибираються по порядку і додаються в порожні місця:

Нащадок 1: 8 2 1 | * * * | 3 7

Те саме робиться для другого нащадка:

Нащадок 2: * * * | 2 1 3 | * *

Нащадок 2: 4 5 6 | * * * | 7 8

У результаті виходять два нащадки:

Нащадок 1: 8 2 1 4 5 6 3 7

Нащадок 2: 4 5 6 2 1 3 7 8

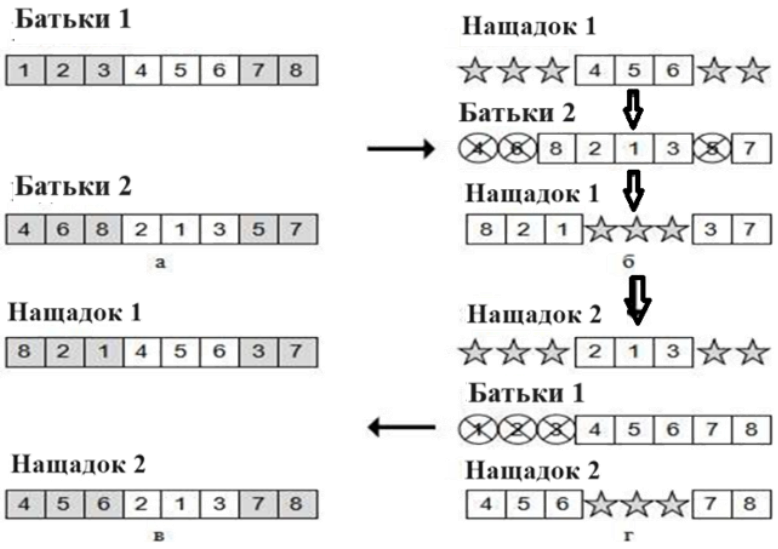

Рисунок 6 – Order Crossover

4.5.3 Циклічний кросовер (Cycle Crossover CX)

Cyclic Crossover. Оператор визначає ряд так званих циклів між двома батьківськими хромосомами. Потім, щоб сформувати першого нащадка,

значення першого циклу копіюється від першого батька, значення другого циклу від другого з батьків, значення третього циклу від першого з батьків і так далі. Приклад:

Батьки 1: 1 2 3 4 5 6 7 8

Батьки 2: 2 4 6 8 7 5 3 1

Цикл 1: Почнемо з першого значення у 1-го батька і опустимося на ту ж позицію у 2-го батька: 1 → 2. Потім шукаємо 2 у 1-го батька та знаходимо його на 2-й позиції, де опускаємося до 4: 1 → 2 → 4. Знову ж таки, ми шукаємо це значення у 1-го батька і знаходимо його на 4-й позиції і опускаємося до 8: 1→ 2 → 4 → 8. Шукаємо 8 у одного з батьків і знаходимо її на 8-й позиції і опускаємося до 1: 1 → 2 → 4 → 8 → 1. Цикл завершений (рисунок 7а).

Значення1-го цикла: 1 2 4 8

Батьки 1: 1 2 **3** 4 **5 6 7** 8

Батьки 2: 2 4 **6** 8 **7 5 3** 1

Цикл 2: 3 → 6 → 5→ 7 → 3 (дв. рисунок 3б).

Значення 2-го цикла: 3 5 6 7

Батьки 1: 1 2 **3** 4 **5 6 7** 8

Батьки 2: 2 4 **6** 8 **7 5 3** 1

Наповнення нащадків значеннями.

Копіювання 1-го циклу: Значення 1-го циклу з 1-го батька копіюються 1-му нащадку, значення від 2-го батька будуть скопійовані 2-му нащадку.

Копіювання 2-го циклу: Значення 2-го циклу з 1-го батька копіюються 2-му нащадку, значення від 2-го батька будуть скопійовані 1-му нащадку. Сформовано два нащадки:

Нащадок 1: 1 2 6 4 7 5 3 8

Нащадок 2: 2 4 3 8 5 6 7 1

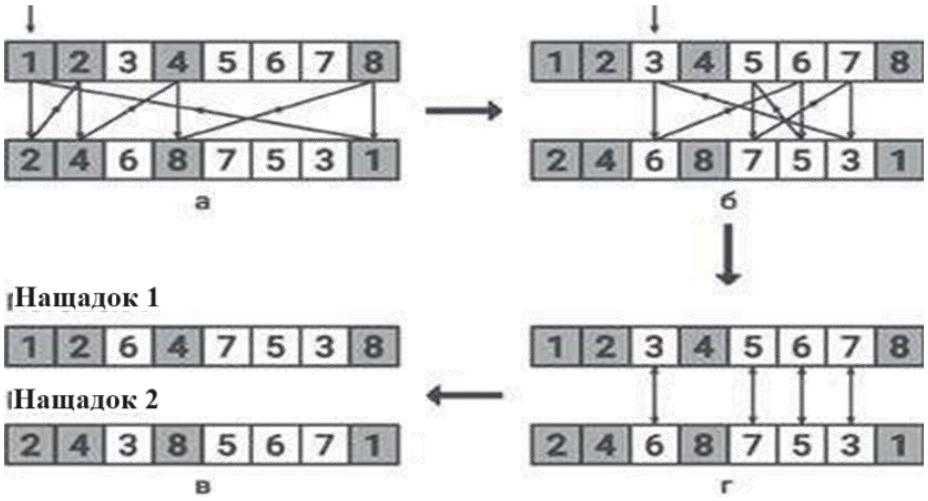

Рисунок 7 – Cycle Crossover[5]

4.6 Мутація

Після операції схрещування хромосоми зазнають мутаційного процесу (mutation). Операція мутації полягає у зміні значень генів деяких позиціях хромосоми з ймовірністю мутації Pm. Якщо після мутації нащадка значення цільової функції покращується, то використовується змінена хромосома нащадка, інакше використовується вихідна хромосома батька. Для проблеми комівояжер відомі наступні оператори мутації: інверсія, вставка, зсув, обмін і мутація на основі розбиття на сегменти. Ці оператори мутації, крім оператора інверсії, не враховують сусідство між ребрами і зберігають вихідну структуру маршруту. Це не дозволяє зберегти високу продуктивність маршруту, що знижує швидкість пошуку оптимального рішення.

Тут використовується метод інверсії послідовності, припустимо, що є батьківська хромосома X, яка містить послідовність міст (1, 3, 7, 4, 8, 5, 6, 2). Оператор мутації випадково вибирає дві точки розриву, наприклад, можна вибрати точки розриву 3 і 7, і інвертувати підрядок між цими двома точками, отримавши дочірню хромосому X ', тобто (1, 3, 7, 6, 5, 8, 4, 2).

Операція мутації зазвичай заснована на якійсь ймовірності мутації Pm, щоб випадково вирішити, чи будуть мутації. Конкретно, кожної генної позиції кожного хромосоми мутація виконуватиметься з ймовірністю Pm, і з

ймовірністю 1-Pm мутації нічого очікувати, і батьківська хромосома буде збережено як дочірня хромосома. У цьому прикладі ми припускаємо, що ймовірність мутації Pm дорівнює 0, 1, тобто з ймовірністю 0, 1 буде виконана мутація для X, інакше батьківська хромосома X буде збережена як дочірня хромосома.

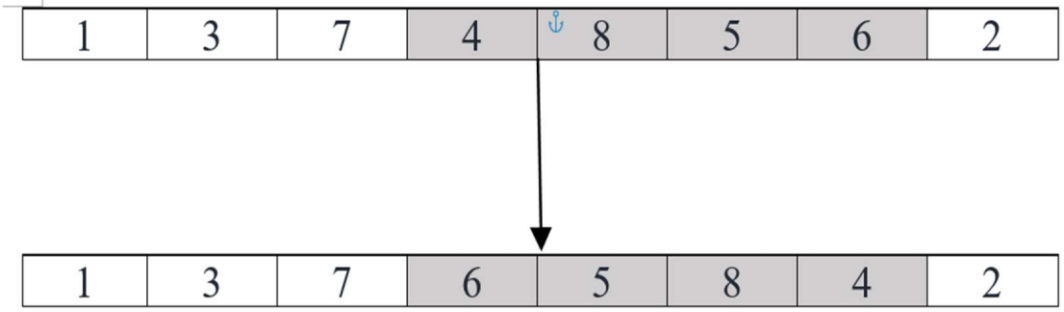

Рисунок 8 – Мутація

4.7 Завершення роботи алгоритму

Коли алгоритм виконує задане число ітерацій і зупиняється, він може вивести найкраще рішення, якщо в процесі ітерацій була здійснена оцінка та збереження кращого рішення.

Після виконання заданого числа ітерацій алгоритм може зупинитися і вивести найкраще рішення, яке було знайдено до цього моменту. Це дозволяє отримати наближене оптимальне рішення задачі, навіть якщо повне оптимальне рішення не було знайдено.

Що ефективність алгоритму оптимізації може залежати від різних факторів, таких як вибрані параметри алгоритму, природа завдання та доступні обчислювальні ресурси. Тому найкраще рішення, знайдене після заданої кількості ітерацій, може бути наближеним, але не обов'язково оптимальним рішенням.

4.8 Схема роботи генетичного алгоритму

Рисунок 9 – Алгоритмічна діаграма

РОЗДІЛ 5

МОДЕЛЮВАННЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Платформа та налаштування параметрів експерименту

1. Програмна платформа для проведення експериментів: ПК на базі операційної системи Windows та програмне забезпечення Matlab.

Основний діапазон налаштувань базових параметрів :

Є 4 параметри генетичного алгоритму, які необхідно встановити заздалегідь, і зазвичай вони встановлюються у наступному діапазоні：

(1) Розмір популяції:20~100

(2) Відсоток ймовірності схрещування:0.5~0.9

(3) Відсоток ймовірності мутацій:0.01~0.1

(4) Кількість поколінь:1000~2000

програмний код представлений у додатку А.

5.2 Результати експерименту

У програмі ініціалізації надано матрицю відстаней між 47 містами, а також виконано моделювання та аналіз результатів із використанням програмного забезпечення Matlab.

Спочатку використовується точний алгоритм гілок та кордонів для розрахунку оптимального рішення з найменшою відстанню та маршрутом для 47 міст як зразок.

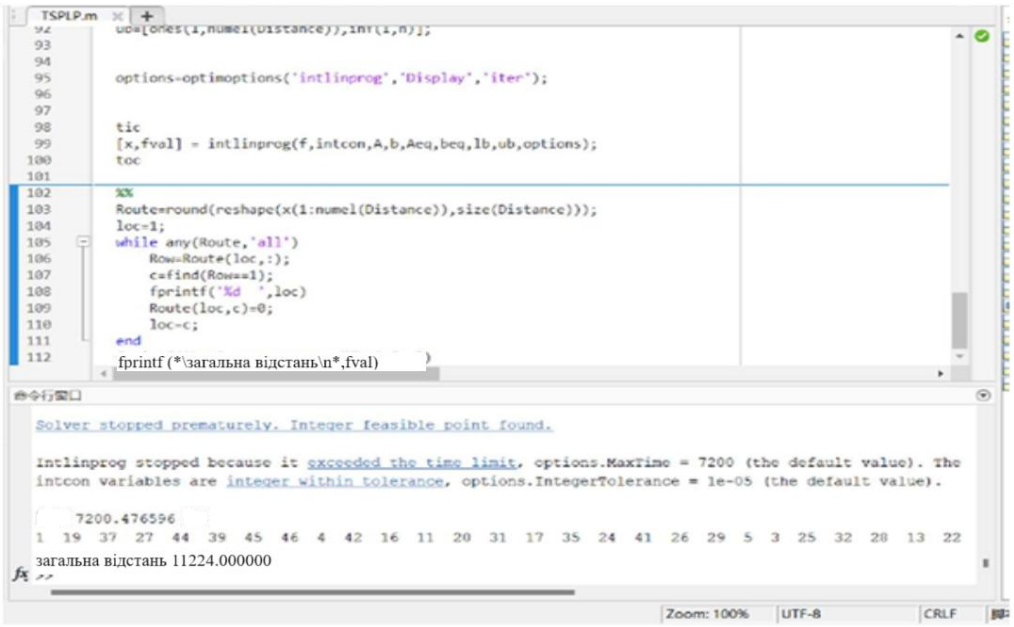

Рисунок 10 – результат алгоритму розгалуження та кордону точне рішення для 47 міст

1. довжина колії: 11224.0000
2. міський маршрут:

1->19->37->27->44->39->45->46->4->42->16->11->20->31->17->35->24->41->26->29->5->3->25->32->28->13->22->12->18->14->38->7->47->8->6->34->36->40->10->9->23->30->33->21->43->15->2->1

Потім застосовується генетичний алгоритм на вирішення завдання. Параметри популяції встановлені рівними 20, 40, 60, 80 та 100. Можливість кросовера PC = 0.5. Використовуються три різних оператори

кросовери: PMX, OX, CX. Імовірності мутації після кожного оператора кросовера встановлені рівними Pm=0.01, Pm=0.05 та Pm=0.1. Ітерації припиняються після 1000 поколінь для нагляду параметрів найменшої відстані.

Таблиця 3 – Order Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12684
20	47	1000	0.5	0.05	13428
20	47	1000	0.5	0.1	14186
40	47	1000	0.5	0.01	11580
40	47	1000	0.5	0.05	12020
40	47	1000	0.5	0.1	12076
60	47	1000	0.5	0.01	11372
60	47	1000	0.5	0.05	11612
60	47	1000	0.5	0.1	12315
80	47	1000	0.5	0.01	11937
80	47	1000	0.5	0.05	11616
80	47	1000	0.5	0.1	11303
100	47	1000	0.5	0.01	11352
100	47	1000	0.5	0.05	11269
100	47	1000	0.5	0.1	11595

Таблица 4 – Partially Mapped Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12156
20	47	1000	0.5	0.05	12775
20	47	1000	0.5	0.1	12665
40	47	1000	0.5	0.01	12436
40	47	1000	0.5	0.05	11722
40	47	1000	0.5	0.1	11993
60	47	1000	0.5	0.01	11337
60	47	1000	0.5	0.05	12061
60	47	1000	0.5	0.1	12044
80	47	1000	0.5	0.01	12366
80	47	1000	0.5	0.05	11972
80	47	1000	0.5	0.1	11556
100	47	1000	0.5	0.01	11921
100	47	1000	0.5	0.05	11627
100	47	1000	0.5	0.1	11485

Таблица 5 – Cycle Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12081
20	47	1000	0.5	0.05	12634
20	47	1000	0.5	0.1	12136
40	47	1000	0.5	0.01	12004
40	47	1000	0.5	0.05	12280
40	47	1000	0.5	0.1	11807
60	47	1000	0.5	0.01	11961
60	47	1000	0.5	0.05	12020
60	47	1000	0.5	0.1	12079
80	47	1000	0.5	0.01	12756
80	47	1000	0.5	0.05	11769
80	47	1000	0.5	0.1	11885
100	47	1000	0.5	0.01	12083
100	47	1000	0.5	0.05	12116
100	47	1000	0.5	0.1	11882

5.3 Локальна оптимізація та покращення генетичного алгоритму

Локальна оптимізація та покращення генетичного алгоритму мають кілька переваг та застосовуються для подальшого покращення якості та продуктивності рішення. Основні переваги включають:

Поліпшення локальних оптимальних рішень: Генетичний алгоритм та інші глобальні оптимізаційні алгоритми можуть застрягати у локальних оптимумах, не досягаючи глобального оптимуму. Локальна оптимізація дозволяє на основі локального оптимуму вносити коригування та покращення, щоб наблизитися до глобального оптимуму.

Підвищення якості рішення: Локальні оптимізаційні алгоритми зазвичай вносять невеликі зміни та коригування у поточне рішення з метою знаходження кращого рішення у локальній області. Це дозволяє отримувати більш точні рішення кожної ітерації і підвищує якість рішення.

Прискорення збіжності алгоритму: Локальна оптимізація швидко виправляє локальні недоліки поточного рішення, прискорюючи збіжність

алгоритму оптимального рішення. Локальна оптимізація скорочує зайвий простір пошуку та робить алгоритм ефективнішим.

Підвищення стійкості алгоритму Локальна оптимізація знижує залежність алгоритму від початкового рішення. Навіть якщо початкове рішення низької якості, після локальної оптимізації алгоритм поступово наближається до оптимального рішення завдяки коригуванням і поліпшенням.

Мета локальної оптимізації полягає у подальшому поліпшенні результатів розв'язання задачі. Хоча генетичний алгоритм дозволяє шукати та знаходити більш оптимальні рішення за допомогою операцій вибору, кросинговера та мутації, у деяких випадках він може не досягти глобального оптимуму або залишити місце для покращень.

Ідея локальної оптимізації полягає у подальшому покращенні знайденого рішення. За допомогою локального пошуку та покращень в обмеженій області можна спробувати додатково зменшити загальну відстань шляху, отримуючи більш оптимальне рішення.

У задачі комівояжера (TSP) локальна оптимізація може здійснюватися шляхом постійної зміни порядку чи позиції шляху. З використанням рекурсії або інших методів можна в кожній ітерації вносити малі коригування в поточне рішення, намагаючись знайти коротший шлях. Таким чином, можна додатково покращити продуктивність генетичного алгоритму і наблизитися до глобального оптимального рішення.

5.3.1 Кроки алгоритму рекурсивної локальної оптимізації

1. Визначення рекурсивної функції: Спочатку необхідно визначити рекурсивну функцію, яка викликатиметься кожної ітерації. Вхідні параметри функції повинні включати поточний індивід, матрицю відстаней та кількість рекурсивних викликів. Наприклад, можна визначити функцію local_optimization, що приймає параметри Chrom (поточний індивід), D (матриця відстаней) та n (кількість рекурсивних викликів).

2. Реалізація алгоритму локальної оптимізації: Всередині рекурсивної функції необхідно реалізувати алгоритм локальної оптимізації. Конкретний метод локальної оптимізації можна вибрати залежно від особливостей завдання. В даному випадку можна використовувати операцію звернення, яка змінює порядок вузлів колії шляхом звернення підпослідовності вузлів. Звернення дозволяє досліджувати різні перестановки колії та спробувати знайти більш оптимальне рішення.

3. Рекурсивний виклик: Усередині рекурсивної функції, коли кількість рекурсивних викликів менша за задане значення, можна продовжувати виконання локальної оптимізації. Для цього можна знову викликати функцію local_optimization, передавши оновленого індивіда, матрицю відстаней та значення n-1 як параметри. Це дозволить функції виконувати додаткові ітерації локальної оптимізації на кожному рекурсивному виклику.

4. Умова завершення рекурсії: У рекурсивній функції необхідно визначити умову завершення рекурсії. Це може бути досягнуто, коли кількість рекурсивних дзвінків досягає заданого значення або коли виконується певна умова зупинки. Наприклад, можна встановити значення n рівним 0 як умову завершення рекурсії.

5.3.2 Тестування рекурсивної локальної оптимізації

Программный код представлен в приложении Б.

На кожній ітерації основного циклу відбувається виклик функції local_optimization для рекурсивного поліпшення локального оптимального індивіда.

Функція local_optimization призначена для виконання локальної оптимізації заданого індивіда. Вона використовує допоміжний масив aux_array для збереження проміжних результатів. Кількість рекурсивних оптимізації визначається параметром 10 і може бути відрегульовано по необхідності.

У функції local_optimization застосовується операція інвертування (Reverse) зміни порядку вузлів в індивіді з метою знаходження оптимального рішення у локальній області. Операція інвертування перевертає підпослідовність вузлів у дорозі, змінюючи порядок між ними. Шляхом багаторазового застосування операції інвертування можна додатково оптимізувати шлях індивіда.

За допомогою багаторазового виклику функції local_optimization в основному циклі можна здійснювати послідовне локальне поліпшення, що дозволяє покращувати якість і продуктивність індивідів. Це сприяє наближенню генетичного алгоритму до глобального раціонального рішення.

Тестування експериментальних даних : 47 міст Параметри популяції встановлені рівними 20, 40, 60, 80 і 100. Ймовірність кросовера PC = 0.5. Використовуються три різні оператори кросовера: PMX, OX, CX. Імовірності мутації після кожного оператора кросовера встановлені рівними Pm=0.01, Pm=0.05 та Pm=0.1. Ітерації припиняються після 1000 поколінь для нагляду параметрів найменшої відстані.

Таблиця 6– Order Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11911
20	47	1000	0.5	0.05	12225
20	47	1000	0.5	0.1	11618
40	47	1000	0.5	0.01	11562
40	47	1000	0.5	0.05	11421
40	47	1000	0.5	0.1	11671
60	47	1000	0.5	0.01	11319
60	47	1000	0.5	0.05	11544
60	47	1000	0.5	0.1	11769
80	47	1000	0.5	0.01	11812
80	47	1000	0.5	0.05	11505
80	47	1000	0.5	0.1	11319
100	47	1000	0.5	0.01	11371
100	47	1000	0.5	0.05	11224
100	47	1000	0.5	0.1	11422

Таблица 7– Partially Mapped Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11714
20	47	1000	0.5	0.05	11904
20	47	1000	0.5	0.1	11951
40	47	1000	0.5	0.01	11455
40	47	1000	0.5	0.05	11516
40	47	1000	0.5	0.1	11913
60	47	1000	0.5	0.01	11650
60	47	1000	0.5	0.05	11555
60	47	1000	0.5	0.1	11590
80	47	1000	0.5	0.01	11578
80	47	1000	0.5	0.05	11926
80	47	1000	0.5	0.1	11377
100	47	1000	0.5	0.01	11463
100	47	1000	0.5	0.05	11590
100	47	1000	0.5	0.1	11483

Таблица 8 – Cycle Crossover (Результати розв'язання задачі TSP)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11866
20	47	1000	0.5	0.05	12413
20	47	1000	0.5	0.1	11790
40	47	1000	0.5	0.01	11455
40	47	1000	0.5	0.05	12137
40	47	1000	0.5	0.1	11731
60	47	1000	0.5	0.01	11711
60	47	1000	0.5	0.05	11533
60	47	1000	0.5	0.1	11862
80	47	1000	0.5	0.01	12159
80	47	1000	0.5	0.05	11427
80	47	1000	0.5	0.1	11607
100	47	1000	0.5	0.01	11751
100	47	1000	0.5	0.05	11362
100	47	1000	0.5	0.1	11559

5.4 Порівняння аналізу даних до та після локальної оптимізації генетичного алгоритму

Порівняння таблиці даних до та після локальної оптимізації з використанням генетичного алгоритму:

Таблиця 9 – Order Crossover (Результати розв'язання задачі TSP до локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12684
20	47	1000	0.5	0.05	13428
20	47	1000	0.5	0.1	14186
40	47	1000	0.5	0.01	11580
40	47	1000	0.5	0.05	12020
40	47	1000	0.5	0.1	12076
60	47	1000	0.5	0.01	11372
60	47	1000	0.5	0.05	11612

Окончание таблицы 9

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
80	47	1000	0.5	0.01	11937
80	47	1000	0.5	0.05	11616
80	47	1000	0.5	0.1	11303
100	47	1000	0.5	0.01	11352
100	47	1000	0.5	0.05	11269
100	47	1000	0.5	0.1	11595

Таблиця 10 – Order Crossover (Результати розв'язання задачі TSP після локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11911
20	47	1000	0.5	0.05	12225
20	47	1000	0.5	0.1	11618
40	47	1000	0.5	0.01	11562
40	47	1000	0.5	0.05	11421
40	47	1000	0.5	0.1	11671
60	47	1000	0.5	0.01	11319
60	47	1000	0.5	0.05	11544
60	47	1000	0.5	0.1	11769
80	47	1000	0.5	0.01	11812
80	47	1000	0.5	0.05	11505
80	47	1000	0.5	0.1	11319
100	47	1000	0.5	0.01	11371
100	47	1000	0.5	0.05	11224

Таблиця 11 – Partially Mapped Crossover (Результати розв'язання задачі TSP до локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12156
20	47	1000	0.5	0.05	12775
20	47	1000	0.5	0.1	12665

Окончание таблицы 11

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
40	47	1000	0.5	0.05	11722
40	47	1000	0.5	0.1	11993
60	47	1000	0.5	0.01	11337
60	47	1000	0.5	0.05	12061
60	47	1000	0.5	0.1	12044
80	47	1000	0.5	0.01	12366
80	47	1000	0.5	0.05	11972
80	47	1000	0.5	0.1	11556
100	47	1000	0.5	0.01	11921
100	47	1000	0.5	0.05	11627
100	47	1000	0.5	0.1	11485

Таблиця 12 – Partially Mapped Crossover (Результати розв'язання задачі TSP після локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11714
20	47	1000	0.5	0.05	11904
20	47	1000	0.5	0.1	11951
40	47	1000	0.5	0.01	11455
40	47	1000	0.5	0.05	11516
40	47	1000	0.5	0.1	11913
60	47	1000	0.5	0.01	11650
60	47	1000	0.5	0.05	11555
60	47	1000	0.5	0.1	11590
80	47	1000	0.5	0.01	11578
80	47	1000	0.5	0.05	11926
80	47	1000	0.5	0.1	11377
100	47	1000	0.5	0.01	11463
100	47	1000	0.5	0.05	11590

Таблиця 13 – Cycle Crossover (Результати розв'язання задачі TSP до локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	12081
20	47	1000	0.5	0.05	12634
20	47	1000	0.5	0.1	12136
40	47	1000	0.5	0.01	12004
40	47	1000	0.5	0.05	12280
40	47	1000	0.5	0.1	11807
60	47	1000	0.5	0.01	11961
60	47	1000	0.5	0.05	12020
60	47	1000	0.5	0.1	12079
80	47	1000	0.5	0.01	12756
80	47	1000	0.5	0.05	11769
80	47	1000	0.5	0.1	11885
100	47	1000	0.5	0.01	12083
100	47	1000	0.5	0.05	12116

Таблиця 14 – Cycle Crossover (Результати розв'язання задачі TSP після локальної оптимізації)

Розмір популяцій	47 Міст	Ітерації	Можливість схрещування	Відсоток мутацій	Довжина колії
20	47	1000	0.5	0.01	11866
20	47	1000	0.5	0.05	12413
20	47	1000	0.5	0.1	11790
40	47	1000	0.5	0.01	11455
40	47	1000	0.5	0.05	12137
40	47	1000	0.5	0.1	11731
60	47	1000	0.5	0.01	11711
60	47	1000	0.5	0.05	11533
60	47	1000	0.5	0.1	11862
80	47	1000	0.5	0.01	12159
80	47	1000	0.5	0.05	11427
80	47	1000	0.5	0.1	11607
100	47	1000	0.5	0.01	11751
100	47	1000	0.5	0.05	11362

Найкращий оптимальний результат: розмір популяції 100, кількість ітерацій 1000, ймовірність мутації 0, 1%, оператор кросовера OX, довжина шляху 11269.

Найгірший оптимальний результат: розмір популяції 20, кількість ітерацій 1000, ймовірність мутації 0, 1%, оператор кросовера OX, довжина колії 14186.

Генетичний алгоритм з ітеративною локальною оптимізацією:

Найкращий оптимальний результат: розмір популяції 100, кількість ітерацій 1000, ймовірність мутації 0, 1%, оператор кросовера OX, довжина шляху 11224.

Найгірший оптимальний результат: розмір популяції 20, кількість ітерацій 1000, ймовірність мутації 0,05%, оператор кросовера OX, довжина колії 12225.

Для об'єктивної оцінки ефективності генетичного алгоритму та його порівняння з точним алгоритмом ми використовували відносне відхилення для порівняння довжин шляхів, отриманих двома методами.

Відносне відхилення (%) між результатами генетичного алгоритму та точним алгоритмом обчислюється за формулою:

(Довжина шляху генетичного алгоритму - Довжина шляху точного алгоритму|

/ Довжина шляху точного алгоритму) * 100

Для кращого оптимального результату стандартного генетичного алгоритму:

Відносне відхилення = (| 11269 - 11224 | / 11224) * 100 ≈ 0, 40%C

Аналогічно, для найгіршого оптимального результату стандартного генетичного алгоритму:

Відносне відхилення = (| 14186 - 11224 | / 11224) * 100 ≈ 26, 43%

Для кращого оптимального результату генетичного алгоритму з ітеративною оптимізацією локальної:

Відносне відхилення = (| 11224 - 11224 | / 11224) * 100 = 0%

Аналогічно, для найгіршого оптимального результату генетичного

алгоритму з ітеративною локальною оптимізацією:

Відносне відхилення = (| 12225 - 11224 | / 11224) * 100 ≈ 8, 92%

Таким чином, відносне відхилення для кращого оптимального результату генетичного алгоритму з ітеративною локальною оптимізацією становить 0%, а для найгіршого оптимального результату - 8, 92%.

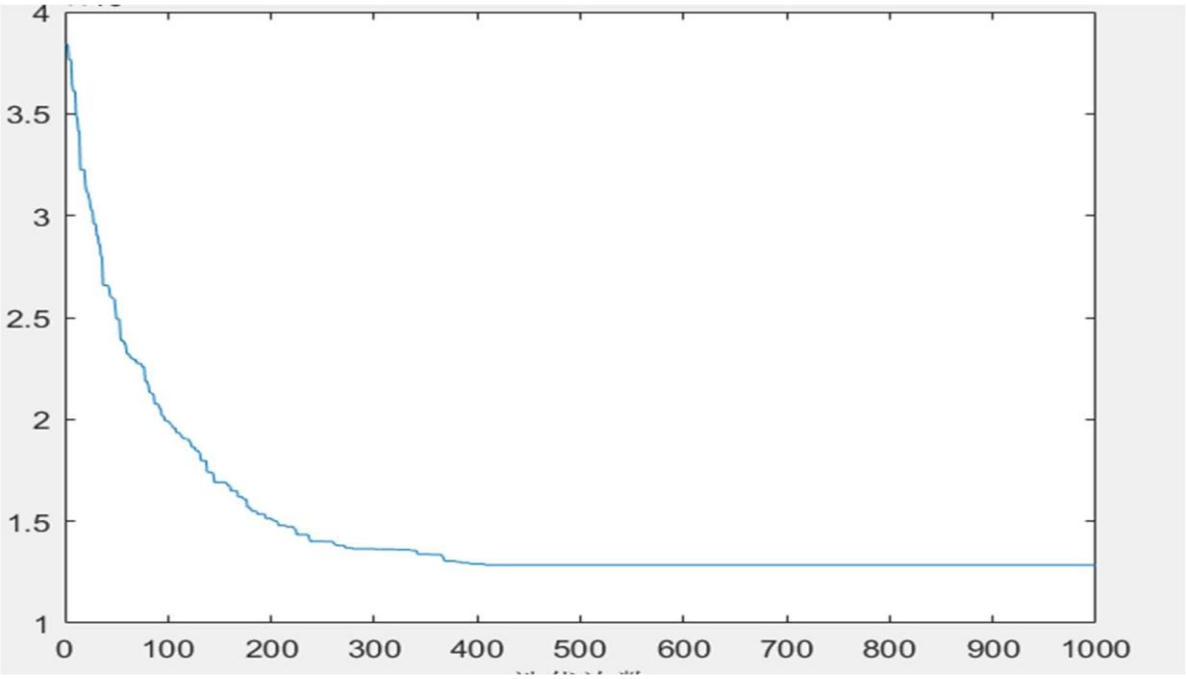

Рисунок 11 – Стандартна крива збіжності ітерацій у генетичному алгоритмі

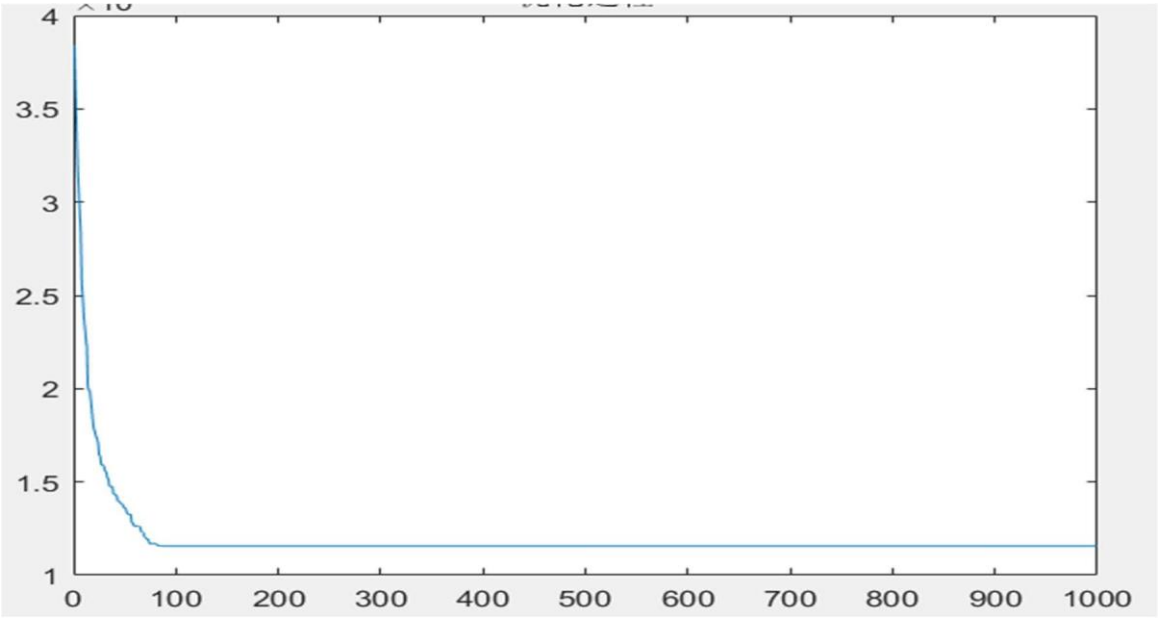

Рисунок 12 – Рекурсивна крива збіжності ітерацій при локальній оптимізації.

З даних видно, що з розв'язанні завдання комівояжера (TSP) генетичний алгоритм з ітеративної локальної оптимізацією значно покращує результати проти стандартним генетичним алгоритмом. Найкращий оптимальний результат генетичного алгоритму з ітеративною локальною оптимізацією повністю збігається з оптимальним рішенням точного алгоритму, відхилення становить 0,00%. З графіка збіжності ітерацій також видно, що алгоритм з ітеративною локальною оптимізацією має більш стабільну збіжність та кращу збіжність.

ВИСНОВОК

У цьому дослідженні було запропоновано метод вирішення проблеми комівояжера, що поєднує генетичний алгоритм із локальною оптимізацією. Шляхом аналізу даних та порівняння результатів традиційного генетичного алгоритму та генетичного алгоритму з локальною оптимізацією були отримані такі висновки:

Генетичний алгоритм з локальною оптимізацією виявляє більш високу продуктивність під час вирішення проблеми комівояжера. Впровадження локальної оптимізації у процесі ітерацій генетичного алгоритму дозволяє покращити рішення та наблизити його до глобального оптимуму. У порівнянні з традиційним генетичним алгоритмом, генетичний алгоритм з локальною оптимізацією виявляє кращу збіжність та якість рішення.

Впровадження локальної оптимізації ефективно підвищує можливості генетичного алгоритму локального пошуку. Локальна оптимізація дозволяє шукати покращення в околиці поточного рішення та додатково оптимізувати локальні частини рішення. Таке поєднання глобального пошуку генетичного алгоритму та локального пошуку локальної оптимізації дозволяє ефективніше знаходити оптимальні рішення.

Результати експериментів показують, що генетичний алгоритм з локальною оптимізацією демонструє стабільнішу збіжність, близьку до точного рішення. Час роботи цього алгоритму також

суттєво скорочується в порівнянні з традиційним алгоритмом.

На закінчення, впровадження локальної оптимізації в генетичний алгоритм дозволяє поліпшити його продуктивність під час вирішення проблеми комівояжера. Це дослідження надає цінні вказівки на вирішення інших складних завдань і підтверджує ефективність поєднання генетичного алгоритму з локальною оптимізацією. У майбутньому дослідження можуть бути спрямовані на вивчення різних методів локальної оптимізації та їх застосування для вирішення інших задач оптимізації з метою подальшого покращення

продуктивності та ефективності алгоритму

ПЕРЕЛІК ПОСИЛАНЬ

1. Holland J H. Genetic algorithms[J]. Scientific american, 1992, 267(1): 66-73.

2. TSPLIB [Электронный ресурс]. – Режим доступа: https:// wwwproxy.iwr.uni-heidelberg.de/groups/comopt/software/ TSPLIB95/.

3. Гардейчик, С. М. Порівняльний аналіз операторів кросовера PMX, CX та OX на прикладі розв'язання задачі комівояжера / С. М. Гардейчик // Вести БДПУ. Серія 3. Фізіка. Математика. Інформатика. Біологія. Географія – 2019. – № 3(101). – С. 94-101.

4. KA D J. An analysis of the behavior of a class of genetic adaptive systems[J]. Doctoral Dissertation University Microfilms 76-9381, 1975.

5. Umbarkar A. J., Sheth P. D. Crossover Operators In Genetic Algorithms: A Review // Ictact journal on soft computing. - 2015. - Vol. 6, № 1. - P. 1083-1092.

6. Wu, J., Liu, F., & Li, Q. MATLAB Genetic Algorithm Optimization Example using the 'ga' Function. Mechanical Engineering and Automation, 2017(2), 61-63.

7. Jiang, R. Application of Improved Genetic Algorithm in TSP Problem. Software Guide, 15(12), 127-129.

8. Zong, D., & Wang, K. Genetic Algorithm with Hybrid Local Search for Solving the Traveling Salesman Problem. Computer Application and Software, 32(3), 266-270+305.

9. Wu, J., Liu, F., & Li, Q. Optimization Example of Genetic Algorithm Function 'ga' in MATLAB. Mechanical Engineering and Automation, 2017(2), 61-63

ДОДАТОК А

Matlab програмний код

```
%% Вхідні дані.
Data=importdata(Incdata.txt');X=Data(:,
2:3);
%% Налаштування параметрів генетичного алгоритму
NIND=100; %популяція
MAXGEN=1000; % Кількість ітерацій Pc=0.5; %
Схрещування ймовірність Pm=0.01; % Мутація
ймовірність GGAP=0.9;
D=importdata('"matrix.txt'); % Вважайте матрицю відстаней, де відстань від
вузла до самого себе одно 0
N=size(D, 1);
%% Створення початкової популяції Chrom =
InitPop (NIND, N); pause(0.0001)
% Вихід випадкового рішення маршруту та
загальної відстані disp('Випадкове значення в
початковій популяції :') OutputPath(Chrom(1,
:));
Rlength = PathLength (D, Chrom (1, :));
disp([' Загальна відстань.  : ',
num2str(Rlength)]);
disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~')
%% оптимізація gen=1; % Лічильник. figure;
hold on;box on xlim([0, MAXGEN])
title('Процес оптимізації') xlabel('ітерація');
ylabel('Довжина шляху');
ObjV = PathLength (D, Chrom); % Розрахунок
курсу. preObjV=min(ObjV);
```

```
while gen<=MAXGEN
%Розрахунок адаптація
ObjV = PathLength (D, Chrom); %Розрахунок
довжини шляху line([gen-1, gen], [preObjV,
min(ObjV)]); pause(0.0001)
preObjV=min(ObjV);
FitnV = Fitness (ObjV);
% відбір
SelCh = Select (Chrom, FitnV, GGAP);
%PMX Схрещування SelCh = Recombin
(SelCh, Pc);
%мутація SelCh = Mutate (SelCh, Pm);
% Операція реверс SelCh = Reverse (SelCh, D);
%Повторно вводиться нова популяція
нащадків Chrom = Reins (Chrom, SelCh, ObjV);
%Ітерація плюс Один gen=gen+1;
end
```

ДОДАТОК Б

Код тестової програми на MATLAB :

% Очищення робочого простору tic

clear clc

%% Ввід данних

Data = importdata(Incdata.txt'); % Це потрібно лише для побудови графіка, можна видалити, якщо не потрібно. Використовуйте натомість координати вузлів 1-47 з попередніх рядків. X = Data(:, 2:3);

%% Параметри генетичного алгоритму NIND = 100;% Размер популяции

MAXGEN = 1000; % Кількість ітерацій Pc = 0.5; % Імовірність схрещування

Pm = 0.1; % Імовірність мутації GGAP = 0.9; % Розрив поколінь

D = importdata('"matrix.txt'); % Завантаження матриці відстаней, де відстань від вузла до самого себе дорівнює 0

N = size(D, 1);

%% Ініціалізація популяції load('abc.mat', 'Chrom')

%% Виведення випадкового маршруту та загальної відстані disp ('Випадковий маршрут у початковій популяції:')

OutputPath(Chrom(1, :));

Rlength = PathLength(D, Chrom(1, :)); disp([Загальна відстань: ', num2str(Rlength)]);

disp('~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

%% Оптимізація

gen = 1; % Лічильник ітерацій figure;

hold on; box on;

xlim([0, MAXGEN]);

title(Процес оптимізації'); xlabel(Кількість ітерацій'); ylabel('Довжина маршруту');

ObjV = PathLength(D, Chrom); % Обчислення довжини маршруту .preObjV = min(ObjV);

while gen <= MAXGEN
~~~~~~~~~~~~~~~~~~~~~~~~~~~~

```
% Обчислення пристосованості
ObjV = PathLength(D, Chrom);      % Обчислення довжини маршруту line([gen-1, gen],  [preObjV, min(ObjV)]);
pause(0.0001); preObjV = min(ObjV); FitnV = Fitness(ObjV);
% Вибір
SelCh = Select(Chrom, FitnV, GGAP);
% Схрещування з використанням PMX SelCh = Recombin(SelCh,  Pc);
% Мутація
SelCh = Mutate(SelCh, Pm);
% Операція звернення SelCh = Reverse(SelCh, D);
% Переселення нової популяції нащадків Chrom = Reins(Chrom, SelCh, ObjV);
% Збільшення лічильника ітерацій gen = gen + 1;
% Виклик функції рекурсивної локальної оптимізації для кращої особи
aux_array = zeros(size(Chrom)); % Створення допоміжного масиву Chrom = local_optimization(Chrom, D, 10); % Рекурсивна оптимізація 10 разів
end
%% Побудова графіка зміни загальної відстані для кращої особи на кожній ітерації
figure;
ObjV = PathLength(D, Chrom); [minObjV, minInd] = min(ObjV); DrawPath(Chrom(minInd(1),   :), X); p = OutputPath(Chrom(minInd(1), :));
disp(['Обща відстань: ', num2str(ObjV(minInd(1)))]); toc
```