Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Вирішення задачі комівояжера за допомогою генетичного алгоритму

(тема)

Виконав:

студент II курсу, групи СПм-22-2

Онипченко А.О.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Іващенко Г.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Онипченку Артему Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Вирішення задачі комівояжера за допомогою генетичного алгоритму

затверджена наказом по університету від " 06 " листопада 2023 р. № 1299 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 15 січня 2024 р.

3. Вхідні дані до роботи

1) документація мови програмування Java;
2) алгоритми вирішення комбінаторних задач на графах;
3) середовище розробки IntelliJ IDEA Community Edition 2023.2.5;
4) середовище розробки SceneBuilder.

4. Перелік питань, що потрібно опрацювати у роботі

1) огляд поширених варіантів вирішення задачі комівояжера
2) вибір та обґрунтування методики та засобів дослідження;
3) програмна реалізація генетичних алгоритмів;
4) проведення експериментальних досліджень;
5) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд варіантів вирішення задачі комівояжера	07.11.23-13.11.23	
2	Вибір та обґрунтування методики дослідження	14.11.23-20.11.23	
3	Вибір інструментальних засобів	21.11.23-23.11.23	
4	Розробка моделі генетичного алгоритму	24.11.23-06.12.23	
5	Проведення експериментів	07.12.23-23.12.23	
6	Оформлення матеріалів кваліфікаційної роботи	26.12.23-02.01.24	
7	Подання кваліфікаційної роботи керівникові та	03.01.24-06.01.24	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	09.01.24-12.01.24	
	вання		

Дата видачі завдання 06 листопада 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Іващенко Г.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 80 с., 18 рис., 5 табл., 2 дод., 26 джерел.

ЗАДАЧА КОМІВОЯЖЕРА, ГЕНЕТИЧНІ АЛГОРИТМИ, МАРШРУТ, МУТАЦІЯ, ГРАФ, ЖАДІБНИЙ АЛГОРИТМ, КРОСОВЕР, ІНТЕРФЕЙС.

Метою кваліфікаційної роботи є дослідження та розробка генетичних алгоритмів для вирішення задачі комівояжера.

У ході виконання кваліфікаційної роботи було проведено аналіз можливості вирішення задачі комівояжера шляхом використання різних видів кросоверів генетичного алгоритму. Проведено аналіз існуючих підходів з виявленням їх переваг та недоліків. Реалізовано та досліджено алгоритми для вирішення задачі комівожера. Досліджено використання жадібного алгоритму у поєднанні з генетичним алгоритмом для покращення шуканого рішення. Реалізовано програмну систему з графічним інтерфейсом користувача для тестування та дослідження впливу різних параметрів та налаштувань алгоритмів на ефективність роботи.

ABSTRACT

Master’s thesis: 80 pages, 18 figures, 5 tables, 2 appendices, 26 sources.

TRADEMAN'S PROBLEM, GENETIC ALGORITHMS, ROUTE, MUTATION, GRAPH, GREEDY ALGORITHM, CROSSOVER, INTERFACE.

The purpose of the qualification work is research and development of genetic algorithms for solving the traveling salesman problem.

In the course of the qualification work, an analysis of the possibility of solving the traveling salesman problem by using various types of crossovers of the genetic algorithm was carried out. An analysis of the existing approaches was carried out with the identification of their advantages and disadvantages. Algorithms for solving the salesman's problem have been implemented and studied. The use of a greedy algorithm combined with a genetic algorithm to improve the sought solution is investigated. A software system with a graphical user interface has been implemented for testing and researching the impact of various parameters and algorithm settings on work efficiency.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА – генетичний алгоритм

ГІ – графічний інтерфейс

ЖА – жадібний алгоритм

ЗК – задача комівояжера

CHC – Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation

MVC – model, view, controller

NP – non-deterministic polynomial

TSP – travelling salesman problem

ВСТУП

На сьогоднішній день одна з найпоширеніших задач теорії графів, задача комівояжера (ЗК), залишається актуальною через її широке застосування в сферах логістики, організації виробничих процесів, вирішення телекомунікаційних проблем, при плануванні подорожей та багатьох інших областях. Головна мета задачі полягає в знаходженні найвигіднішого маршруту, який проходить через всі вузли у графі по одному разу та повертається у вихідну точку.

Задача комівояжера є класичним прикладом NP-повної задачі, через це вирішення її за допомогою традиційних точних алгоритмів, таких як повний лексичний перебір [1], метод гілок та меж [2], є трудомістким завданням. До класу складності NP відносяться задачі, які можна вирішити за допомогою недетермінованих алгоритмів. Ці алгоритми передбачають обробку декількох варіантів результатів, обираючи варіант, який найбільше задовольняє певним критеріям. Вибір може здійснюватися «випадковим» чином. ЗК є найбільш вивченою NP-повною задачею комбінаторної оптимізації, і існує велика кількість методів, які застосовувалися для її вирішення [3].

Знаходження найкоротшого маршруту за прийнятний проміжок часу ускладняється зі збільшенням обсягу наборів даних. У багатьох практичних застосуваннях достатньо знайти прийнятне, але не обов’язково найкраще рішення, за менший час. У зв'язку з цим, широкого розповсюдження набули різноманітні евристичні підходи, такі як генетичні та мурашині алгоритми, які не гарантують знаходження найкращого рішення, проте забезпечують прийнятну точність результату за обмежений час [4].

Актуальність роботи полягає у тому, що, незважаючи на існуючі підходи до вирішення задачі комівояжера, є необхідність у подальшій реалізації та удосконаленні цих підходів для покращення пошуку рішення та зниження витрат у практичних сценаріях використання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис задачі комівояжера

Задача комівояжера є однією з класичних задач комбінаторної оптимізації, яка була вперше сформульована в середині 20-го століття і з того часу стала однією з найвідоміших і детально вивчених задач в галузі комп'ютерної науки та математики. Рішення задачі комівояжера полягає у пошуку найкоротшого маршруту, який проходить через заданий набір міст, при цьому кожне місто відвідується рівно один раз і закінчується маршрут у початковій точці. Задача комівояжера пов'язана з задачею про пошук мінімального кістяка в графі [5].

Задача комівояжера здобула популярність та почала широко використовуватись у наукових дослідженнях у галузі оптимізації та алгоритмів, має широкий спектр практичних застосувань в таких галузях, як логістика, транспортування, маршрутизація та багатьох інші областях [6], у яких є актуальною проблема пошуку найкоротшого маршруту. Не зважаючи на те, що задача комівояжера відноситься до класу NP-повних задач, на сьогоднішній день існує велика кількість різноманітних підходів, які надають прийнятну точність рішення за прийнятний час. Задача має кілька варіантів та модифікацій залежно від додаткових обмежень та умов [7].

1.1.1 Класична ЗК

В класичній задачі комівояжера розглядається сценарій, в якому торговець має обов'язково відвідати кожну точку і повернутися до стартового пункту, з метою зменшення загальної довжини маршруту. У цій задачі мінімізується відстань, яку необхідно подолати для виконання всіх візитів до точок [5].

1.1.2 Симетрична ЗК

Симетрична задача комівояжера представляє собою одну із варіацій класичної задачі комівояжера [7]. В цьому випадку відстані між парами точок однакові, незалежно від напрямку маршруту. Тобто, відстань від точки А до точки В дорівнює відстані від точки В до точки А. Ця симетрія спрощує обчислення та розв'язання задачі, оскільки не потрібно враховувати напрямок руху між точками. Головна мета задачі залишається незмінною, проте симетричність відстаней дозволяє використовувати більш ефективні методи пошуку найкращого маршруту.

1.1.3 Асиметрична ЗК

Головна відмінність асиметричної та симетричної задачі комівояжера полягає у тому, що в асиметричній відстані між точками в різних напрямках маршруту можуть бути різними [7]. Це означає, що відстань від точки А до точки В може відрізнятися від відстані між цими точками у зворотному напрямку. В асиметричній задачі комівояжера також потрібно відвідати кожну точку маршруту один раз і повернутися до його початку, але завдання ускладняється нерівністю відстаней між парами точок. Вирішення асиметричної задачі комівояжера вимагає визначення найвигіднішого маршруту, який зазвичай враховує відстані і вартості між точками, а також інші фактори, які впливають на знаходження найкращого маршруту.

1.1.4 Метрична ЗК

Метрична задача комівояжера – це модель задачі комівояжера, в якій відстані між точками мають властивість метричності. Це означає, що відстані між містами виконують умови метрики, тобто вони є додатними, симетричними та задовольняють нерівність трикутника.

Не-метричною може бути задача, наприклад, коли потрібно знайти оптимальний маршрут для мінімізації часу подорожі при урахуванні різних варіантів транспорту в різних напрямах. У такій ситуації може статися, що обхідний маршрут літаком виявиться коротшим, ніж прямий маршрут на автомобілі.

1.1.5 Задача декількох комівояжерів

Задача декількох комівояжерів (Multiple Traveling Salesman Problem) – варіація класичної ЗК, в якій за умовою не один, а декілька торговців. У цьому варіанті багато торговців починають з одного міста та повинні відвідати кожне місто. Завдання полягає в тому, щоб розподілити міста між торговцями так, щоб мінімізувати загальну відстань або час подорожі. Кожен торговець має свій власний набір міст для відвідування, і їм можуть бути задані різні критерії або обмеження, такі як максимальна відстань подорожі або обмежена кількість міст для відвідування. Даний варіант задачі застосовується у тих випадках, коли декілька агентів повинні ефективно відвідати певні точки на мапі, дотримуючись певних обмежень та завдань.

1.1.6 Залежна від часу ЗК

Задача комівояжера, що залежить від часу, представляє собою варіацію класичної ЗК, в якій враховуються змінні часові параметри на шляху між містами. Відстані або вартості подорожі між містами можуть змінюватися в залежності від часу доби або інших факторів. Ця задача враховує те, що шляхи між містами можуть бути різними в різний час і дозволяє знайти оптимальний маршрут з урахуванням часових обмежень. ЗК, що залежить від часу, застосовується в сферах, де важливо планувати маршрути з урахуванням змінних умов, таких як транспортування з урахуванням трафіку, доставка за різними графіками тощо.

1.1.7 Стохастична ЗК

У стохастичній версії задачі комівояжера вартість подорожі або відстань між точками не є фіксованою константою, а є випадковою величиною. Це означає, що вартість подорожі між двома точками може змінюватися з ітерації на ітерацію. Отже, мета полягає в тому, щоб знайти оптимальний маршрут, враховуючи цю випадковість.

Для вирішення стохастичної ЗК використовуються різні методи, які спрямовані на знаходження оптимального маршруту або наближеного рішення, враховуючи випадковий характер цих відстаней. Одним з таких методів є ітеративне покращення, яке полягає у багаторазових повторних обчисленнях та оцінках, кожен раз з урахуванням випадкових вартостей відстаней між точками.

Стохастична задача комівояжера поширена в тих ситуаціях, де вартість подорожі може змінюватися під час проходження. Наприклад, вартість поїздки на таксі може бути абсолютно різною для однакової відстані, тому що на неї впливають такі фактори, як завантаженість трафіку, попит.

1.1.8 ЗК з обмеженнями

Задача комівояжера з обмеженнями є розширеним варіантом класичної ЗК, в якій крім пошуку найкоротшого маршруту, що проходить через всі точки і повертається до початкової точки, додаються додаткові обмеження або умови [8]. Поширеними є обмеження на кількість відвідуваних точок, пріоритети відвідування певних точок, обмеження в часі на відвідування кожної окремої точки або взагалі максимальний час, за який комівояжер має завершити подорож. Інший варіант поширених обмежені – габаритні, що передбачає присутність у задачі певних параметрів, таких як вага або об’єм, і необхідно знайти маршрут, який буде проходити через всі задані точки, але при цьому будуть дотримані обмеження щодо ваги чи об’єму вантажу.

Також в задачі комівояжера можуть бути обмежені і інші параметри, які впливають на ефективність побудови маршруту. Введення додаткових умов та обмежень значно ускладнює знаходження розв’язку задачі. Для її вирішення можна використовувати різні оптимізаційні методи, такі як генетичні алгоритми, методи лінійного програмування, а також різноманітні гібридні підходи, які поєднують в собі різні методи оптимізації для вирішення задачі комівояжера з обмеженнями [9].

1.1.9 ЗК з кластерами

В даній модифікації класичної ЗК обираються кластери, тобто, групи точок. Мета задачі полягає у тому, щоб знайти найкоротший шлях, пройшовши яким комівояжер відвідає кожен кластер хоча б один раз і повернеться в початкову точку [10].

1.2 Існуючі методи вирішення задачі комівояжера

Для вирішення ЗК використовувалась велика кількість різноманітних підходів. Вони розподіляються за методами вирішення (точні та наближені методи) та за типами алгоритмів (евристичні та метаевристичні).

Точні методи, такі як повний лексичний перебір або метод гілок та меж гарантовано знаходять найкраще рішення [11], проте вони є досить ресурсозатратними для великих наборів даних.

Наближені методи – методи, які не гарантують знаходження найкращого рішення, проте надають прийнятне рішення за менший час. Сюди відносяться метод найближчого сусіда, генетичні алгоритми тощо.

Евристичні алгоритми – це алгоритми розв'язання задач, які можуть знайти прийнятне рішення серед можливих, але не можуть гарантувати оптимальний результат [12]. Хоч ці алгоритми є неточними та не гарантують знаходження найкращого рішення, проте забезпечують прийнятну точність

результату за обмежений час. Іноді евристичні алгоритми можуть знаходити точне найкраще рішення, але вони все одно залишаються евристичними.

Метаевристичні алгоритми – це загальні фреймворки для розв'язання різних задач оптимізації і пошуку [13]. Їх відмінність від евристичних алгоритмів полягає у тому, що метаевристичні алгоритми є загальними та потужними фреймворками для розв'язання різних видів оптимізаційних задач, тоді як евристичні алгоритми зазвичай спрямовані на рохв’язання конкретні задачі і можуть використовувати лише прості правила та евристики. Так як прості точні алгоритми є повільними та ресурсозатратними для великих наборів даних, то більш ефективними є різноманітні евристичні підходи. До цих підходів належать генетичні алгоритми, метод вставки, метод найближчого сусіда, імітація відпалу, мурашиний алгоритм та багато інших.

1.2.1 Генетичні алгоритми

Генетичні алгоритми – адаптивні методи пошуку, що використовуються для вирішення завдань оптимізації та пошуку. В їх основі лежать ідеї біологічної еволюції та природного відбору і використовуються для знаходження найкращих рішень у задачах, де існує велика кількість можливих варіантів [14].

Генетичні алгоритми базуються на принципі постійного вдосконалення результату шляхом зміни послідовних поколінь множин можливих варіантів рішення. Під час кожного кроку проводяться схрещування, мутації та відбір, що призводить до підвищення середньої якості рішень в популяції з кожним новим поколінням. Цей процес триває доти, поки не досягнуто задовільного рішення, або пройдено певну кількість ітерацій.

Переваги генетичних алгоритмів:

- універсальність та гнучкість;
- можливість враховувати багато різних рішень одночасно.

Недоліки:

- відносно висока обчислювальна вартість. Для великих задач ГА можуть бути обчислювально витратними, оскільки вони проводять обробку великої кількості рішень в популяції;

- випадковість рішення. Оскільки пошук відбувається випадковим чином, то для досягнення більшої точності може знадобитись багаторазовий запуск алгоритму.

1.2.2 Метод вставки

Метод вставки – це один із способів сортування масиву або списку даних. Його застосування для вирішення ЗК розпочинається з пустого маршруту і послідовно додаються міста так, щоб мінімізувати загальну довжину шляху. Треба обрати будь-яку початкову точку, яка буде початковим рішенням. Далі додається кожна точка, яка ще не включена в маршрут, вираховується вартість маршруту і обирається точка, яка призводить до мінімального збільшення вартості маршруту. Дана процедура повторюється до ти пір, поки всі міста не будуть включені в маршрут [23].

Переваги методу:

- простота реалізації;
- досить ефективний для невеликих наборів даних.

Недоліки методу:

- не гарантує знаходження оптимального рішення
- потребує додаткових оптимізацій.

1.2.3 Метод найближчого сусіда

Метод найближчого сусіда часто називають жадібним алгоритмом, принцип роботи якого полягає у виборі на кожному кроці точки, яка на даному етапі є найкращим варіантом, не зважаючи на можливі наслідки [25].

Переваги жадібного алгоритму:

- простота реалізації;
- швидкість роботи.

Недоліки:

- не завжди знаходять найкраще рішення;
- не є універсальним, через що підходить не для всіх задач.

1.2.4 Метод імітації відпалу

Метод імітації відпалу – це стохастичний оптимізаційний алгоритм. Основна ідея алгоритму імітації відпалу полягає в тому, щоб знайти глобальний оптимум, рухаючись в просторі можливих рішень, при цьому дозволяючи з певною ймовірністю приймати гірші рішення для уникнення потрапляння в локальні оптимуми [15]. Алгоритм отримав свою назву від аналогії з процесом відпалу металу. Цей алгоритм дозволяє вибирати гірші рішення на ранніх етапах і допомагає уникнути застрягання в локальних оптимумах.

Переваги алгоритму:

- гнучкість;
- здатність до виходу з локальних оптимумів.

Недоліки:

- випадковість;
- швидкість роботи (може бути повільним для складних задач);
- необхідність налаштування параметрів.

1.2.5 Мурашиний алгоритм

Мурашиний алгоритм – це метаевристичний алгоритм, який заснований на спостереженнях за поведінкою мурах при пошуку найкоротших шляхів до джерела їжі [24]. Кожна мураха залишає сліди

феромонів на своєму шляху, які інші мурахи можуть вловити і використовувати. Ці феромони слугують для інших мурах як позначки і вказують маршрути, які вже були пройдені. Мурахи віддають перевагу шляхам з більшою кількістю феромонів. При виборі свого маршруту кожна мураха керується ймовірностями, які враховують кількість феромонів на різних шляхах та відстань до найближчого міста, яке ще не було відвідано.

Переваги алгоритму:

- може адаптуватися до змінних умов;
- підходить для різних задач (задача комівояжера, задачі маршрутизації та інші).

Недоліки:

- високі обчислювальні витрати;
- залежність від початкових умов;
- необхідність підбору параметрів.

1.3 Постановка задачі

Задача комівояжера формально визначається таким чином.

Дана множина місць (міст) для відвідування. Вартість подорожі (або відстань) між кожною парою міст. Ця інформація зазвичай представляється у вигляді матриці відстаней D, $D[i][j]$, де $D[i][j]$ – вартість подорожі з міста i в місто j [16].

Задача полягає в знаходженні такої перестановки міст, яка б мінімізувала загальну вартість подорожі і включала б у себе всі міста.

Місто 1 (початкове місто) завжди залишається незмінним і є останнім місцем в маршруті. Метрика, яку треба мінімізувати, називається довжиною маршруту.

Цільова функція – мінімізувати суму відстаней по всім маршрутам, які відвідують всі міста один раз і завершуються в початковому місті:

$$\text{minimize} \sum D[i][j] * x[i][j] \tag{1.1}$$

Обмеження такі. Кожне місто повинно бути відвідано рівно один раз: $\sum x[i][j] = 1$ для всіх i. Кожне місто може бути вибрано як наступне після іншого міста тільки один раз: $\sum x[i][j] = 1$ для всіх j. Необхідно уникнути циклів: $\sum x[i][i] = 0$ для всіх i та забезпечити правильну послідовність міст: $\sum u[i] - u[j] + n * x[i][j] \le n - 1$ для всіх i, j, де $u[i]$ – величина зв'язана з послідовністю міст.

Метою роботи є аналіз використання різновидів генетичних алгоритмів для вирішення задачі комівояжера. Генетичні алгоритми є актуальними для вирішення задачі комівояжера завдяки їх здатностям до глобального пошуку, паралелізації, гнучкості і здатності працювати з великими задачами. Вони можуть бути ефективним інструментом для пошуку найкращих розв'язків у великому просторі можливих маршрутів задачі комівояжера.

З метою проведення аналізу ефективності обраних варіантів генетичного алгоритму реалізується програмний застосунок [17] на основі платформи JavaFX на мові програмування Java. Для покращення швидкості пошуку рішення використовується поєднання генетичного алгоритму з іншими методами пошуку найкоротшого шляху.

У ході дослідження проводиться аналіз отриманих результатів та визначення можливостей використання реалізованих генетичних алгоритмів для покращення пошуку ефективного рішення та зниження витрат у практичних сценаріях.

2 ВИКОРИСТОВУВАНІ МЕТОДИ

2.1 Генетичні алгоритми

Генетичні алгоритми – адаптивні методи пошуку, що використовуються для вирішення завдань оптимізації та пошуку [18]. В їх основі лежать ідеї біологічної еволюції та природного відбору і використовуються для знаходження найкращих рішень у задачах, де існує велика кількість можливих варіантів.

Генетичні алгоритми виникли як результат спостережень і спроб моделювання природних процесів, які мають місце у живій природі, таких як еволюція та пов'язана з нею селекція (природний відбір) в популяціях живих істот (рисунок 2.1).

Рисунок 2.1 – Модель ГА

Основну ідею генетичних алгоритмів висловив американський вчений Джон Холланд у кінці 1960-х – початку 1970-х років. Спостереження за процесами природної еволюції вказувало на те, що ці процеси можуть бути використані для вирішення оптимізаційних задач. Ідея полягала у створенні алгоритму, який міг би вирішувати складні задачі, використовуючи операції аналогічні до процесу еволюції. Тому було розпочато розробку алгоритмів, які працювали з послідовностями двійкових цифр, що отримали назву хромосом. Ці алгоритми моделювали процеси еволюції в поколіннях таких хромосом і включали в себе механізми селекції та репродукції, подібні до тих, які спостерігаються в природі (рисунок 2.2).

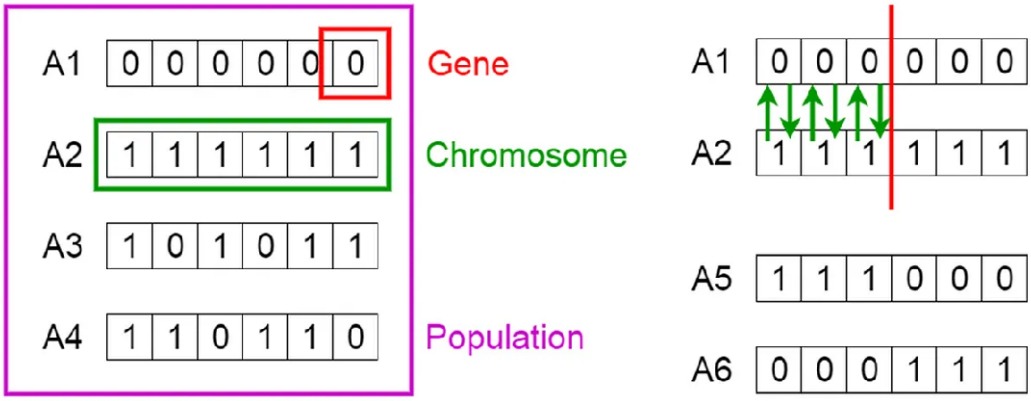

Рисунок 2.2 – Гени, хромосоми, популяція, схрещування, мутація

2.1.1 Поняття природного відбору

Етап природного відбору розпочинається з вибору найбільш адаптованих особин у популяції. Ці особини розмножуються, передаючи свої генетичні особливості наступному поколінню. Якщо батьки виявляються більш адаптованими, їхні нащадки отримують покращені характеристики та мають більше шансів на виживання. Цей процес повторюється, і поступово формується покоління, в якому переважають найбільш адаптовані особини. У генетичному алгоритмі розглядаються п'ять фаз:

- початкова популяція;
- фітнес-функція;
- видбір;
- кросовер;
- мутація.

2.1.2 Початкова популяція

Процес починається з створення набору особин, що називається популяцією. Кожна особина – це рішення задачі, яка вирішується. Особина характеризується набором параметрів – генів. Гени поєднуються в ланцюжок, утворюючи хромосому.

У генетичному алгоритмі набір генів індивідума представляється рядком, в термінах алфавіту (рисунок 2.3). Зазвичай використовуються двійкові значення (рядок із 1 і 0). Це називається кодуванням генів у хромосомі.

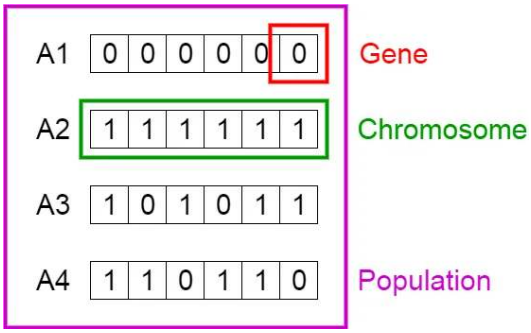

Рисунок 2.3 – Популяція, хромосоми, гени

2.1.3 Фітнес-функція

Функція пристосованості визначає, наскільки пристосований є індивід. Пристосованість показує здатність особини конкурувати з іншими особинами. Фітнес-функція оцінює пристосованість кожної особини.

Імовірність того, що особина буде обрана для розмноження, залежить від показника її пристосованості.

2.1.4 Відбір

Етап відбору полягає в тому, щоб відібрати найбільш пристосованих особин та дозволити їм передати свої гени наступному поколінню.

Дві пари особин (батьків) обираються з урахуванням їх показників пристосованості. Особи з високими показниками мають більше шансів бути відібраними для створення нового покоління.

2.1.5 Кросовер

Кросовер – найважливіша фаза генетичного алгоритму. Для кожної пари батьків, які підлягають спаровуванню, точка кросовера вибирається випадково серед генів. Наприклад, припустимо, що точка перетину дорівнює 3, як показано нижче (рисунок 2.4).

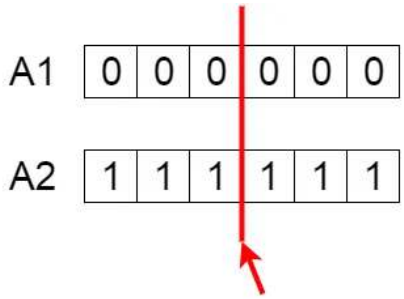

Рисунок 2.4 – Точка схрещування хромосом

Нащадок створюється шляхом обміну генів батьків між собою до досягнення точки схрещування (рисунок 2.5).

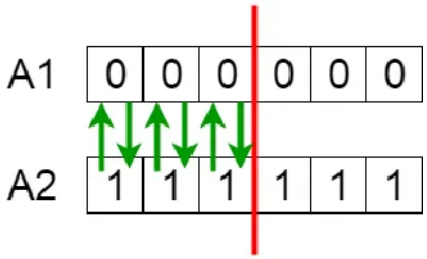

Рисунок 2.5 – Обмін генами між батьками

Нове потомство (рисунок 2.6) додається до популяції.

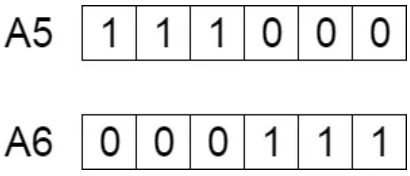

Рисунок 2.6 – Нове потомство

2.1.6 Мутація

У деяких нових нащадків, що утворилися, деякі їх гени можуть піддаватися мутації з малою випадковою ймовірністю. Це означає, що деякі біти в рядку біт можна інвертувати (рисунок 2.7).

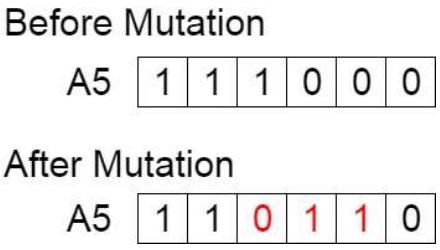

Рисунок 2.7 – Нащадки до та після мутації

Мутація використовується для підтримки різноманітності у популяції і запобігання передчасної збіжності.

2.1.7 Припинення дії

Популяція має фіксовану чисельність. У міру формування нових поколінь особини з найменшою пристосованістю вмирають, звільняючи місце для нового потомства. Послідовність фаз повторюється, щоб у кожному новому поколінні народжувалися особини, які краще, ніж попереднє покоління.

Для припинення роботи генетичного алгоритму можуть бути використані такі критерії зупинки: досягнення допустимого для користувача результату, вичерпання кількості ітерацій роботи, вказаної користувачем, та настання збіжності роботи алгоритму до певного значення. У цьому випадку алгоритм завершує роботу, якщо популяція не виробляє потомство, яке істотно відрізняється від попереднього покоління і має переваги відносно нього. Це свідчить про те, що генетичний алгоритм надав набір розв'язків вирішуваної проблеми і подальша робота не має сенсу.

2.2 Моделі генетичних алгоритмів

Існують різні моделі генетичного алгоритму (класичний, простий, гібридний, CHC генетичний алгоритм тощо).

Моделі генетичних алгоритмів відрізняються за стратегіями відбору та формування нового покоління особин, операторами мутації та кросоверу, обраним способом кодуванням [19].

2.2.1 CHC-алгоритм

Алгоритм CHC (Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation) – це еволюційний алгоритм, який використовується для розв'язання оптимізаційних завдань [20]. Алгоритм Cross generational elitist selection має декілька основних характеристик:

- для формування нового покоління вибираються N найкращих індивідів серед батьків і дітей без повторень. Це допомагає зберегти кращі рішення у популяції;

- для схрещування випадково вибираються пари батьків, але з урахуванням обмежень. Забороняється вибір пар батьків з малою хеммінговою відстанню або відстанями між крайніми відмінними бітами;

- для схрещування використовується одноточковий кросовер HUX, в результаті якого нащадок успадковує половину бітів від кожного з батьків;

- розмір популяції обмежений, зазвичай складає приблизно 50 індивідів. Ця малорозмірність популяції обґрунтовується використанням одноточкового кросовера.

Якщо популяція стає надто схожою або стагнує у процесі еволюції, CHC використовує «cataclysmic mutation». Це означає, що всі індивіди, крім найпристосованішого, піддаються змінам в своєму генетичному матеріалі.

Після «cataclysmic mutation» генетичний алгоритм перезапускається і продовжує роботу, використовуючи лише кросовер для поліпшення розв'язків.

CHC-алгоритм використовується для оптимізації завдань, де необхідно знайти найкращий набір параметрів або рішення в просторі пошуку.

2.2.2 Genitor

Цей алгоритм був запропонований в середині 1990-х і є одним із популярних підходів до оптимізації. Властивості, якими Genitor-подібні алгоритми відрізняються від класичного генетичного алгоритму:

- в Genitor-алгоритмі батьки вибираються з поточного покоління без врахування їхньої придатності і вони на кожному кроці створюють лише одного нащадка;

- цей нащадок замінює одну з найгірших особин у популяції, а не батьків;

- у Genitor-алгоритмі відбір батьків зазвичай проводиться без врахування їхньої придатності, і він базується на випадковому виборі індивідів з поточного покоління.

В цілому, Genitor-алгоритм відрізняється від класичного генетичного алгоритму своїм підходом до вибору батьків та формування нащадків.

2.2.3 Гібридні алгоритми

Суть гібридних алгоритмів полягає в поєднанні генетичних алгоритмів з іншими алгоритмами пошуку, які підходять для вирішення даної задачі. Вони комбінують переваги різних методів і можуть бути ефективними для вирішення різних проблем, включаючи оптимізацію, класифікацію, рекомендації, обробку даних та інші [21]. У кожному поколінні відбувається оптимізація нащадків з використанням певного методу, після чого відбувається стандартне виконання генетичного алгоритму. Цей вид еволюції відомий як ламаркізм, де особина здатна навчатися, а потім зберігати отримані навички у своєму генотипі, щоби передати їх потомству.

На практиці гібридні алгоритми часто виявляються успішними. Це пов'язано з тим, що зазвичай існує висока ймовірність того, що якась особина потрапить до сфери глобального максимуму і після оптимізації стане рішенням завдання.

2.2.4 Паралельні генетичні алгоритми

Паралельні генетичні алгоритми використовують паралельні обчислення для вирішення задач, передбачають використання значних обчислювальних ресурсів: множини процесорів, обчислювальних вузлів або обчислювальних кластерів для розподіленого обчислення.

Використання засобів паралельних та розподілених обчислень вимагає модифікацій звичайного генетичного алгоритму, але це дозволяє вирішувати

задачі швидше та ефективніше, розподіляючи обчислення між багатьма обчислювальними одиницями.

2.2.5 Острівна модель

Острівна модель генетичних алгоритмів використовує розділення популяції на окремі групи або «острови», кожен з яких має власну популяцію. Еволюція цих груп проводиться незалежно одна від одної. Кожен острів може еволюціонувати в різних напрямках, що впливає на різноманітність рішень. Острівна модель генетичних алгоритмів використовує паралельні обчислення, завдяки чому частина простору рішень обробляється одночасно, що позитивно впливає на швидкість процесу оптимізації [22].

2.2.6 Комірчасті генетичні алгоритми

Комірчасті генетичні алгоритми – модель генетичних алгоритмів, в якій для організації популяції та процесу еволюції використовується сітка з комірками [20]. Популяція організовується у вигляді сітки або решітки, де кожна комірка представляє одного індивіда. Всередині кожної з цих комірок проводиться локальна еволюція, де індивіди розмножуються, мутують і конкурують друг з одним. Іноді індивіди можуть переміщатися між сусідніми комірками і обмінюватись генами.

Кожна комірка може розвиватися незалежно від інших, що сприяє збереженню різноманітності в популяції.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Загальна структура проекту

Для дослідження та тестування розроблених варіантів генетичних алгоритмів для вирішення задачі комівояжера створена програмна система на мові програмування Java з використанням платформи JavaFX та патерну проектування MVC (рисунок 3.1). Використання цього патерну дозволяє розділити структуру застосунка на три компоненти: Model, View і Controller.

Model містить дані та логіку, пов’язану з обробкою точок і відстаней між ними. Цей компонент відповідає за зберігання та обробку необхідної інформації, а також розрахунки, пов'язані з розв'язанням задачі комівояжера.

Controller відповідає за обробку введених даних від користувача, взаємодію з Model для отримання необхідної інформації та відправлення відповідних команд для оновлення інтерфейсу.

View, у патерні MVC, відповідає за відображення даних користувачеві та оновлення інтерфейсу. Цей компонент взаємодіє з Model для отримання необхідних даних і з Controller для отримання вказівок щодо оновлення інтерфейсу. Забезпечуючи візуальне представлення інформації, View грає ключову роль у взаємодії користувача з програмою та представленні результатів обчислень.

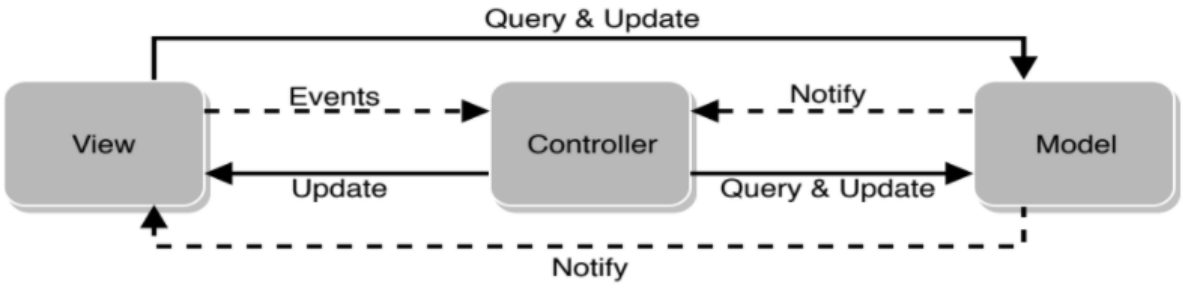

Рисунок 3.1 – Структура MVC

Структура проекту складається з головного класу TSPGeneticAlgorithmApp, класу моделі TSPGeneticAlgorithmModel, контролеру Controller та файлу design.fxml (рисунок 3.2).

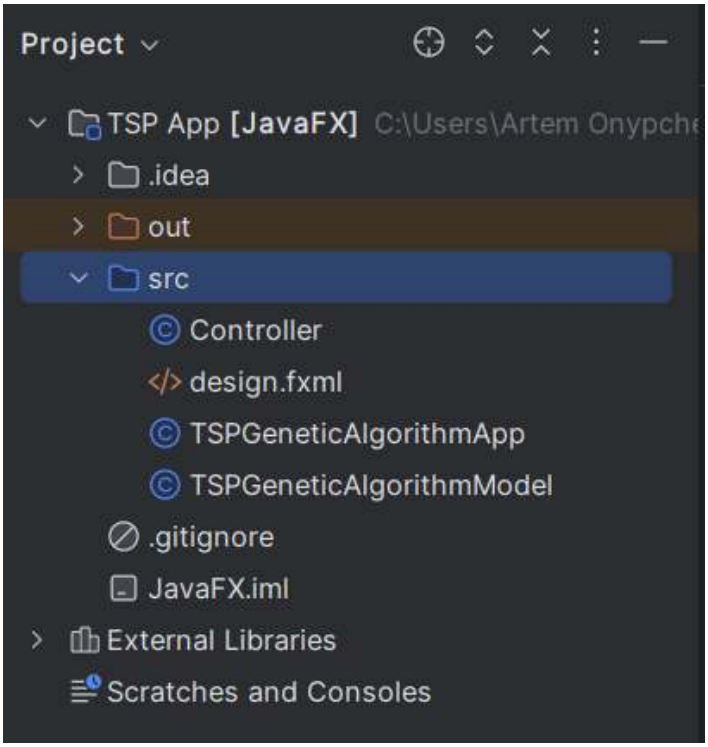

Рисунок 3.2 – Структура model

Файл design.fxml визначає структуру та вигляд головного вікна застосунку за допомогою компонентів JavaFX. В даному файлі вказується контролер, який відповідає за обробку подій для взаємодії з графічним інтерфейсом. Інтерфейс розробленого застосунку складається з таких компонентів, як AnchorPane, Canvas, Pane, Text,TextField, Button, RadioButton та CheckBo:

- AnchorPane – це головний контейнер, в якому розміщуються інші елементи і який забезпечую прив’язку до вікна застосунку;
- Canvas – контейнер, який використовується для відображення графічних елементів, таких як точки та маршрут між ними;
- Pane слугує для розміщення елементів управління системою;

- Text відображає текстові написи інтерфейсу;
- Button – кнопки, які забезпечують виклик функцій генерації точок та вирішення задачі комівояжера;
- RadioButton компонент використовується для вибору варіанту кросовера;
- CheckBox забезпечує використання жадібного алгоритму у поєднанні з генетичним алгоритмом.

Для розробки головного вікна використовувалась програма SceneBuilder (рисунок 3.3). Це графічний інструмент для побудови та редагування графічного інтерфейсу за допомогою технології JavaFX. Він надає візуальний інтерфейс, що дозволяє швидко та зручно створювати графічний інтерфейс без написання коду вручну.

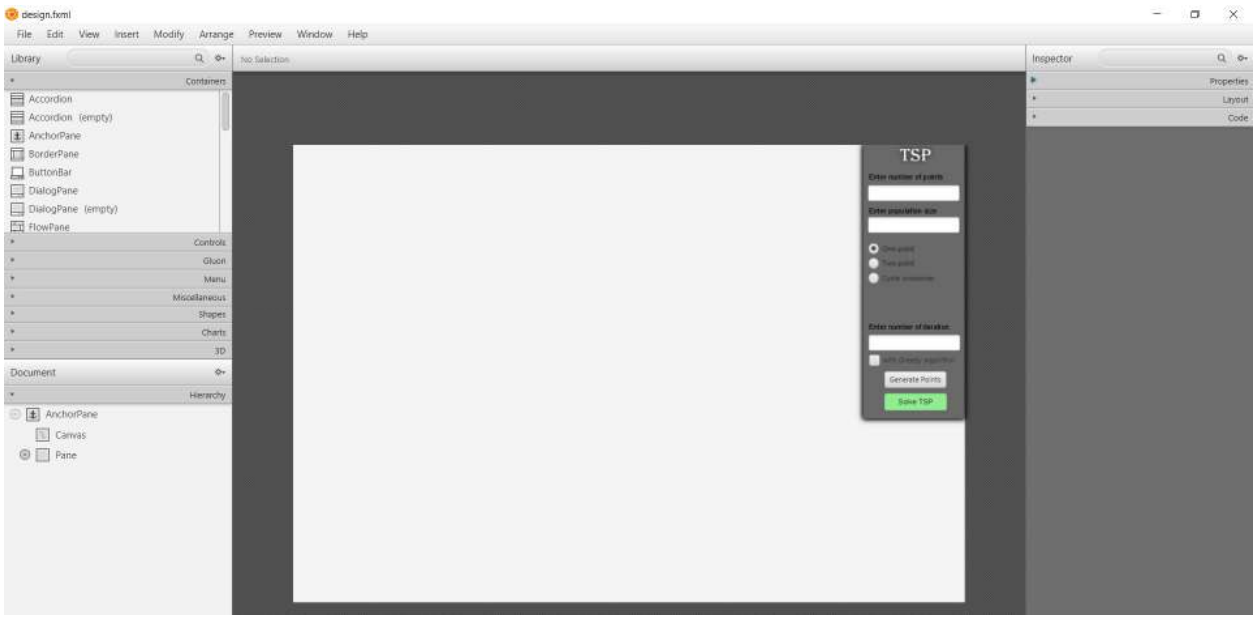

Рисунок 3.3 – Графічний інтерфейс SceneBuilder

3.2 Опис головного класу

Основним класом реалізованого JavaFX застосунку є клас TSPGeneticAlgorithmApp (лістинг 3.1). Він складається з методу main, який є

точкою входу для виконання програми та викликає метод launch, який запускає JavaFX застосунок, та методу start. Метод start використовує об’єкт FXMLLoader для завантаження графічного інтерфейсу з файлу design.fxml, а із завантаженого інтерфейсу отримує батьківський контейнер, на основі якого встановлюється сцена та відображається вікно програми.

Лістинг 3.1 – Клас TSPGeneticAlgorithmApp

```
public class TSPGeneticAlgorithmApp extends Application
{
        @Override
    public void start(Stage primaryStage) throws Exception
    {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("design.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args)
    {
        launch(args);
    }
}
```

3.3 Реалізація алгоритмів

3.3.1 Опис класу генетичного алгоритму

Логіка, пов’язана з генетичним алгоритмом для вирішення задачі комівояжера, міститься в класі TSPGeneticAlgorithmModel. Також в класі TSPGeneticAlgorithmModel знаходиться певна частина структур даних, які використовуються під час вирішення задачі.

Клас TSPGeneticAlgorithmModel містить наступні поля:

- список точок, які представляють координати на площині;
- список особин, які представляють можливі шляхи комівояжера;

- радіокнопки, які слугують для вибору варіанту кросовера;
- флаг, який вказує на використання жадібного алгоритму;
- розмір популяції;
- ймовірність мутації;
- кількість елітних осіб;
- кількість особин для жадібного алгоритму.

За ініціалізацію полів відповідає конструктор, який приймає список точок, радіокнопки та флаг для застосування жадібного алгоритму.

Метод initializePopulation ініціалізує початкову популяцію випадковим чином. За еволюцію генетичного алгоритму відповідає метод evolve. Даний метод на основі вибору користувача за допомогою радіокнопок визначає, який тип кросовера використовувати.

Далі запускається цикл для еволюції генетичного алгоритму і зберігаються найкращі особини з попереднього покоління в нову популяцію. Якщо обрано чекбокс useGreedyAlgorithm, то використовується жадібний алгоритм для покращення шляху для частини популяції. Для решти популяції застосовуються операції кросовера та мутації для створення нових особин.

3.3.2 Жадібний алгоритм

Реалізація жадібного алгоритму знаходиться в методі greedyAlgorithm (лістинг 3.2). На початку роботи методу створюються список remainingPoints для збереження залишкових точок та список greedyPath для побудови покращеного шляху. Зі списку remainingPoints випадковим чином обирається початкова точка та додається в greedyPath. Далі відбувається жадібний вибір, доки є залишкові точки. Від поточної точки обчислюється найменша відстань до всіх точок, що залишились, і обирається відповідна точка, що має мінімальну відстань. Ця точка додається до шляху і в результаті виконання методу повертається покращений шлях.

Лістинг 3.2 – Жадібний алгоритм

```
public List<Point2D> greedyAlgorithm(List<Point2D> initialPath){
    List<Point2D> remainingPoints = new
ArrayList<>(initialPath);
    List<Point2D> greedyPath = new ArrayList<>();
    Point2D currentPoint = remainingPoints.remove((int)
(Math.random() * remainingPoints.size()));
    greedyPath.add(currentPoint);
    while (!remainingPoints.isEmpty()) {
        double minDistance = Double.MAX_VALUE;
        Point2D nextPoint = null;
        for (Point2D point : remainingPoints) {
            double distance =
TSPUtils.calculateDistance(currentPoint, point);
            if (distance < minDistance) {
                minDistance = distance;
                nextPoint = point;
            }
        }
        remainingPoints.remove(nextPoint);
        greedyPath.add(nextPoint);
        currentPoint = nextPoint;
    }
    return greedyPath;}
```

3.3.3 Методи генетичного алгоритму

Для вибору батьківської особини для подальших операцій генетичного алгоритму в класі TSPGeneticAlgorithmModel реалізовано метод selectParent. За допомогою функції Math.random генерується випадкове число від 0.0 до 1.0, множиться на розмір популяції і результат конвертується в ціле число. В результаті повертається вибрана випадкова особина з поточної популяції.

Метод crossover реалізує одноточковий кросовер (лістинг 3.3). Випадково обираються точки (початок і кінець) для обміну шляхами між батьками. Новий шлях для нащадка спочатку ініціалізується нульовим значенням, після чого до нового шляху копіюється частина шляху між точками обміну від першого батька, а решта шляху заповнюється точками з другого батька. Точки, які вже входять до шляху, ігноруються.

Лістинг 3.3 – Метод crossover

```
private List<Point2D> crossover(Individual parent1, Individual
parent2) {
    int start = (int) (Math.random() * points.size());
    int end = (int) (Math.random() * points.size());
    if (start > end) {
        int temp = start;
        start = end;
        end = temp;
    }
    List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
    for (int i = start; i <= end; i++) {
        childPath.set(i, parent1.getPath().get(i));
    }
    int currentIndex = 0;
    for (Point2D point : parent2.getPath()) {
        if (!childPath.contains(point)) {
            while (childPath.get(currentIndex) != null) {
                currentIndex = (currentIndex + 1) %
points.size();
            }
            childPath.set(currentIndex, point);
            currentIndex = (currentIndex + 1) % points.size();
}}
    return childPath;}
```

Метод twoPointCrossover відповідає за здійснення операцій двоточкового кросоверу між батьківськими особинами для створення нової особини (лістинг 3.4). Випадковим чином обираються дві точки для обміну генетичною інформацією між батьками. Фрагмент шляху між цими двома точками копіюється від одного з батьків в нащадка, а решта шляху заповнюється точками з іншого батька, ігноруючи ті, які вже присутні в новому шляху.

Лістинг 3.4 – Метод twoPointCrossover

```
private List<Point2D> twoPointCrossover(Individual parent1,
Individual parent2) {
    int point1 = (int) (Math.random() * points.size());
    int point2 = (int) (Math.random() * points.size());
    int start = Math.min(point1, point2);
```

```
    int end = Math.max(point1, point2);
    List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
    for (int i = start; i <= end; i++) {
        childPath.set(i, parent1.getPath().get(i));  }
    int currentIndex = 0;
    for (Point2D point : parent2.getPath()) {
        if (!childPath.contains(point)) {
            while (childPath.get(currentIndex) != null) {
                currentIndex = (currentIndex + 1) %
points.size();  }
            childPath.set(currentIndex, point);
            currentIndex = (currentIndex + 1) % points.size();}}
    return childPath;}
```

Метод cycleCrossover реалізує циклічний варіант кросовера (лістинг 3.5), який полягає в тому, щоб створити нащадка, обираючи цикли з одного з батьків та заповнюючи решту шляху точками з іншого батька. Випадковим чином обирається вихідна точка, після чого створюється цикл, копіюючи генетичну інформацію з першого батька до нащадка. Ці точки запам’ятовуються і цикл перемикається на іншого батька та заповнюється решта точок, які ще не включені в цикл. Дії повторюються, доки цикл не завершиться.

Лістинг 3.5 – Метод cycleCrossover

```
private List<Point2D> cycleCrossover(Individual parent1,
Individual parent2) {
    List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
    boolean[] visited = new boolean[points.size()];
    int cycleStart = (int) (Math.random() * points.size());
    int index = cycleStart;
    do {
        childPath.set(index, parent1.getPath().get(index));
        visited[index] = true;
        index =
parent2.getPath().indexOf(parent1.getPath().get(index));
    } while (index != cycleStart);
    for (int i = 0; i < points.size(); i++) {
        if (!visited[i]) {
            childPath.set(i, parent2.getPath().get(i));
        }}
    return childPath;}
```

Метод mutate відповідає за мутацію шляху в генетичному алгоритмі (лістинг 3.6). Цей метод дозволяє випадковим чином здійснювати мутацію маршруту, що допомагає розширити простір пошуку та покращити ефективність генетичного алгоритму.

Лістинг 3.6 – Метод mutate

```
private void mutate(List<Point2D> path) {
    if (Math.random() < mutationRate) {
        int index1 = (int) (Math.random() * path.size());
        int index2 = (int) (Math.random() * path.size());
        Collections.swap(path, index1, index2);}}
```

Для створення та представлення окремого індивіда, тобто рішення у генетичному алгоритмі для задачі комівояжера, створено клас Individual (лістинг 3.7). Кожен індивід є маршрутом, який проходить через всі точки заданого набору. Клас Individual містить такі поля та методи:

- список точок, які представляють маршрут;
- значення функції пристосованості;
- функція для розрахунку пристосованості calculateFitness;
- метод інтерфейсу Comparable, який порівнює індивідів за значення функції пристосованості.

Лістинг 3.7 – Фрагмент класу Individual

```
private static class Individual implements
Comparable<Individual> {
    private List<Point2D> path;
    private double fitness = -1;
    public Individual(List<Point2D> path) {…}
    public List<Point2D> getPath() {…}
    public double getFitness() {…}
    private double calculateFitness() {…}
    @Override
    public int compareTo(Individual o) {…}
}
```

Клас TSPUtils відповідає за допоміжні функції, які використовуються при рішенні задачі комівояжера (лістинг 3.8). Він містить два методи – calculateDistance та calculateRouteDistance. Ці методи розраховують відстань між двома точками та протяжність всього маршруту відповідно.

Лістинг 3.8 – Фрагмент класу TSPUtils

```
public static class TSPUtils {
private static double calculateDistance(Point2D point1, Point2D
point2){...}
public static double calculateRouteDistance(List<Point2D.Double>
route) {...}}
```

3.4 Реалізація контролера

Клас Controller є контролером у патерні проектування MVC. Він відповідає за взаємодію користувача з системою та обробку подій в результаті цієї взаємодії. Головними завданнями класу є налаштування графічних елементів інтерфейсу, отримання та обробка введених даних від користувача. Контролер реагує на події, які відбуваються в інтерфейсі та викликає відповідні методи, запускаючи роботу генетичного алгоритму. Після отримання результатів розв’язання задачі він оновлює інтерфейс та відмальовує рішення.

Контролер містить наступні поля та методи:

- список точок та список їх порядкових номерів;
- флаг використання жадібного алгоритму;
- визначення елементів графічного інтерфейсу;
- метод ініціалізації контролера, налаштування обробників подій та графічних елементів;
- метод, який встановлює точку по кліку мишею;
- методи генерації випадкових точок та малювання даних точок на у вікні застосунку;

- метод, який забезпечує виклик генетичного алгоритму для вирішення задачі комівояжера та виведення результатів;
- метод, який відмальовує маршрут на графічному полотні.

Метод initialize використовується для ініціалізації об’єкта контролера та налаштування початкових параметрів інтерфейсу (лістинг 3.9). Даний метод викликається автоматично при завантаженні файлу design.fxml в головному класі програми.

Лістинг 3.9 – Фрагмент методу initialize

```
@FXML
void initialize() {
    points = new ArrayList<>();
    canvas.setWidth(633.0);
    canvas.setHeight(450.0);
    double maxWidth = 633.0;
    double maxHeight = 450.0;
    canvas.widthProperty().addListener((observable, oldValue,
newValue) -> {…});
    canvas.heightProperty().addListener((observable, oldValue,
newValue) -> {…});
    canvas.setOnMouseClicked(this::handleCanvasMouseClicked);
    ToggleGroup toggleGroup = new ToggleGroup();
    onePointRadioButton.setToggleGroup(toggleGroup);
    twoPointRadioButton.setToggleGroup(toggleGroup);
    cycleCrossoverRadioButton.setToggleGroup(toggleGroup);
    generateButton.setOnAction(event -> {…});
    solveButton.setOnAction(event -> solveTSP());
    greedyAlgorithm.selectedProperty().addListener((observable,
oldValue, newValue) -> {…});}
```

За обробку кліків миші на графічному полотні відповідає метод handleCanvasMouseClicked. Він отримує координати кліку миші, створює нову точку з цими координатами та викликає метод, який відмальовує точку на полотні.

В реалізованому застосунку є можливість встановлення точок по кліку миші, а також віипадкова генерація точок. Для цього передбачено поле введення кількості точок та кнопка Generate Points (рисунок 3.4).

Рисунок 3.4 – Поле введення кількості точок та кнопка генерації

За генерацію точок випадковим чином відповідає метод generateRandomPoints, який викликається під час натискання кнопки генерації точок. Генерація координат в межах розмірів графічного полотна для кожної точки здійснюється за допомогою функції Math.random. Після створення точка додається до списку points, а в список pointNumber потрапляє її порядковий номер.

На основі списку згенерованих точок викликається метод drawPoints, який в циклі отримує координати точок та відмальовує їх на графічному полотні за допомогою функції fillOval класу GraphicsContext (лістинг 3.10).

Лістинг 3.10 – Фрагмент методу drawPoints

```
for (Point2D.Double point : points) {
    double x = point.getX();
    double y = point.getY();
    double radius = 1;
    gc.setFill(Color.BLACK);
    gc.fillOval(x - radius, y - radius, 2 * radius, 2 * radius);
    gc.setFill(Color.RED);
    gc.fillText(Integer.toString(pointNumbers.get(index)), x +
5, y - 5);
    index++;}
```

Після виведення точок на полотно додається розмір популяції, обирається варіант кросовера, можливість поєднання генетичного і жадібного алгоритму, додається кількість ітерацій у циклі генетичного алгоритму та натискається кнопка Solve TSP (рисунок 3.5).

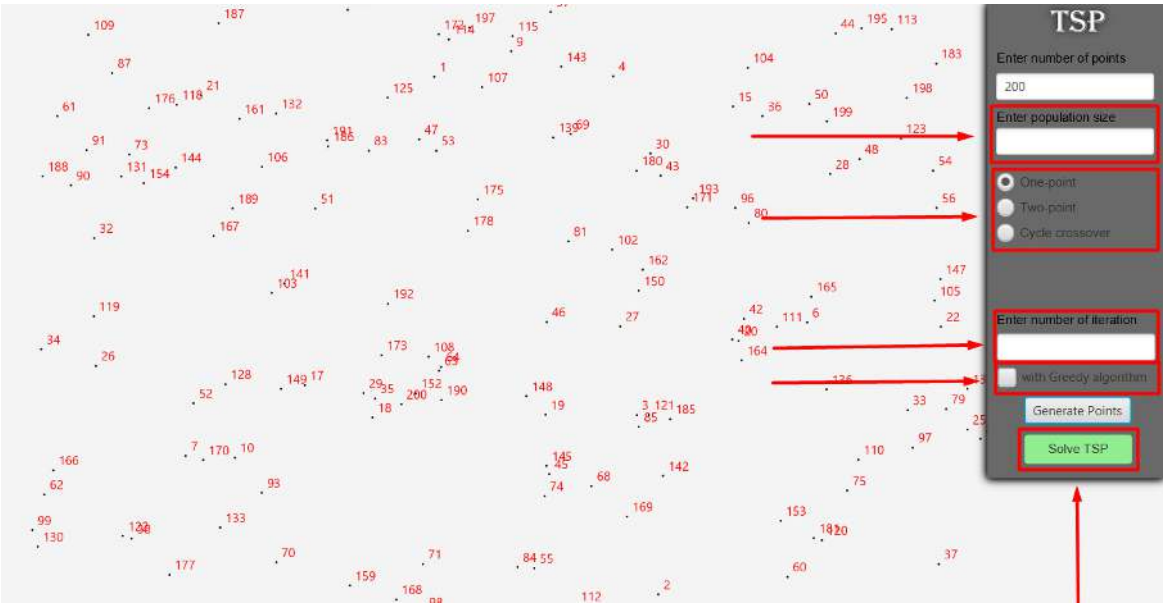

Рисунок 3.5 – Відображення точок та налаштування параметрів

Натискання кнопки породжує подію, яка викликає метод solveTSP. Даний метод відповідає за розв’язання задачі комівояжера з використанням генетичного алгоритму. Він викликає метод evolve, який запускає генетичний алгоритм для пошуку найкращого рішення. За допомогою System.currentTimeLillis визначається точний час виконання алгоритму. В результаті завершення процесу вирішення задачі у консоль виводяться тип кросовера, координати точок найкращого рішення, індекси цих точок, час вирішення та відстань маршруту. Ці дані використовуються для аналізу результатів дослідження. В результаті роботи методу викликається метод drawPath, який в якості параметра приймає список точок, який є найкращим рішенням (лістинг 3.11).

Лістинг 3.11 – Метод drawPath

```
private void drawPath(List<Point2D.Double> path) {
        GraphicsContext gc = canvas.getGraphicsContext2D();
        if (path == null || path.size() < 2) {
            System.out.println("Invalid path. Unable to draw.");
            return;
        }
        drawPoints(gc);
        for (int i = 0; i < path.size() - 1; i++) {
            Point2D.Double p1 = path.get(i);
            Point2D.Double p2 = path.get(i + 1);
            gc.strokeLine(p1.getX(), p1.getY(), p2.getX(),
p2.getY());
        }
        Point2D.Double firstPoint = path.get(0);
        Point2D.Double lastPoint = path.get(path.size() - 1);
        gc.strokeLine(lastPoint.getX(), lastPoint.getY(),
firstPoint.getX(), firstPoint.getY());
    }
```

Метод drawPath використовує функцію strokeLine класу GraphicsContext та відмальовує маршрут між точками в рамкам графічного полотна.

Для тестування розробленої системи у графічному інтерфейсі наявні всі необхідні елементи для зміни параметрів алгоритмів. Представлена можливість мануального та автоматизованого додавання точок для дослідження роботи алгоритмів на графах різної розмірності. Для дослідження впливу розміру популяції та варіанту кросовера наявні поле вводу та радіокнопки відповідно. Можливість дослідження впливу кількості ітерацій на ефективність генетичного алгоритму можна дослідити змініючи значення в текстовому полі. Для дослідження поєднання генетичного та жадібного алгоритму наявний чекбокс «with Greedy algorithm».

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Параметри та критерії ефективності алгоритму

Для початку експериментальних досліджень необхідно задати початкові параметри для роботи генетичного алгоритму.

Генетичний алгоритм має наступні параметри:

- розмір популяції;
- ймовірність мутації;
- кількість елітних особин.

Також треба встановити розмір підмножини популяції, яка буде використовуватись у жадібному алгоритмі. Налаштування даних параметрів є важливим етапом, необхідним для досягнення успішної роботи алгоритму. Вони напряму впливають на розмаїття рішень та на обчислювальні витрати.

У рамках проведення експериментів були використані різноманітні значення параметрів та умов:

- графи, які складаються з 200, 350 та 500 вершин, сгенеровані випадковим чином;
- розмір популяції для проведення дослідження був 100, 300 та 500 особин;
- застосовувалась низька (0,2), середня (0,5) та абсолютна (1,0) ймовірність мутації;
- застосовано 3 варіанти кросоверу: одноточковий, двоточковий та циклічний. Також був досліджений варіант поєднання кожного типу з жадібним алгоритмом.

Оцінкою ефективності роботи алгоритмів є час пошуку оптимального рішення та загальна відстань маршруту. Загальна відстань маршруту розраховується як сума відстаней між двома точками та відстань між першою і останньою точкою.

4.2 Дослідження роботи генетичного алгоритму

Параметри роботи генетичного алгоритму для проведення досліджень були встановлені наступним чином:

- розмір популяції – 100 особин;
- ймовірність мутації – 0,5;
- кількість елітних особин – 5.

Перший варіант графу складається з 200 вершин. Досліджувався вплив розміру популяції та варіанту кросовера на якість рішення (таблиця 4.1).

Таблиця 4.1 – Порівняння результатів застосування ГА для різних розмірів популяцій на графі розмірністю 200 вершин

Кросовер	Розмір популяції	Ймовірність мутації	Критерії ефективності	
			Час, мс	Відстань, мм
Одноточковий	100	0,5	5523	54963,599
	300		15685	57813,237
	500		26149	61200,292
Двоточковий	100	0,5	5288	52485,074
	300		16101	56871,918
	500		25390	60513,156
Циклічний	100	0,5	402	40375,547
	300		2004	40879,164
	500		4651	40214,635

Проаналізувавши отримані результати роботи генетичного алгоритму можна помітити, що для одного розміру популяції одноточковий та двоточковий кросовери дають майже однакову якість результату. Час знаходження оптимального рішення не відрізняється більше, ніж на 0,8 секунди, проте довжина маршруту для одноточкового кросовера більша за довжину маршруту для двоточкового кросовера в середньому на 2,4%. В

масштабах дослідження така відмінність довжин є незначною, проте вона може досить сильно впливати на інші показники (наприклад, вартість маршруту) в задачах більшого масштабу.

Найкращі результати були отримані при використанні циклічного кросовера. Час пошуку найближчого до найкращого рішення перевершує показники двох інших варіантів кросовера в середньому в 10 разів, що дорівнює приблизно 13,4 секунди. А значення довжини маршруту менше за значення попередніх кросоверів в середньому на 17,5 тисяч міліметрів. Слід зазначити, що для всіх значень кількості популяцій довжина маршруту була приблизно однаковою і відрізнялась в середньому всього на 1,1%.

Проаналізувавши показники можна помітити, що збільшення кількості популяцій призводить відповідно до збільшення часу пошуку найкращого рішення (рисунок 4.1) та довжини маршруту (рисунок 4.2). Переважно це пов’язано з тим, що з більшою кількістю особин у популяції зростає і різноманітність та кількість варіантів маршрутів. Так як генетичні алгоритми не гарантують знаходження 100% найкращого рішення, то відповідно зростає і вірогідність того, що при додаванні нових особин до популяції, деякі з них можуть представляти менш оптимальні шляхи, що може впливати на середню довжину маршруту.

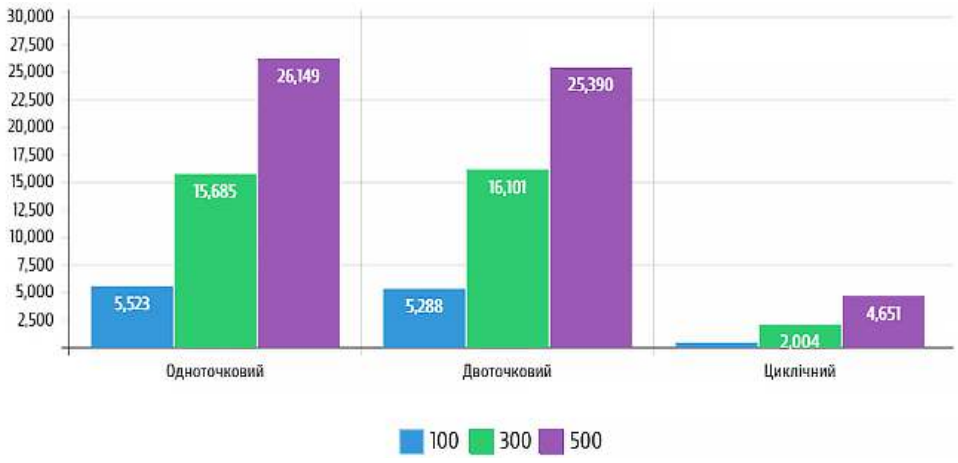

Рисунок 4.1 – Діаграма оцінок часу знаходження найкращих рішень для різних варіантів кросовера та розмірів популяцій

Рисунок 4.2 – Діаграма оцінок довжини маршруту найкращих рішень для різних варіантів кросовера та розмірів популяцій

Другий варіант графу складався з 350 вершин. На даному наборі вершин можна зробити аналогічні висновки, як і з попереднім набором. Різниця між результатами одноточкового та двоточкового кросоверів дуже незначна, проте циклічний кросовер показує значно кращі результати в порівнянні з ними. В той час як середня різниця у часі знаходження рішення між одноточковим та двоточковим кросовером в середньому складає 2,4 секунди, то циклічний кросовер в середньому знаходить рішення на 14,8, 41,8 та 66,7 секунд швидше для кожного розміру популяції відповідно. Довжина маршруту, знайденого в результаті роботи циклічного кросовера, на 47% менша за результати інших варіантів.

Третій варіант графу налічує 500 вершин (таблиця 4.2). Зі збільшенням розмірості графа різниця в результатах одноточкового та двоточкового кросовера стає більш помітною і час знаходження рішення вже відрізняється на 3,6-9,1 секунди в залежності від розміру популяції (рисунок 4.3). Проте різниця в довжині так і залишається близько 2%. Аналогічно до попередніх тестів найкращі результати надає генетичний алгоритм з використанням циклічного кросовера. Зі збільшенням кількості вершин даний кросовер демонструє значно менший час знаходження найкращого рішення.

Таблиця 4.2 – Порівняння результатів застосування ГА для різних розмірів популяцій на графі розмірністю 500 вершин

Кросовер	Розмір популяції	Ймовірність мутації	Критерії ефективності	
			Час, мс	Відстань, мм
Одноточковий	100	0,5	29927	162308,458
	300		90532	172849,864
	500		151177	172032,384
Двоточковий	100	0,5	26278	160266,751
	300		83168	170959,659
	500		142044	167380,062
Циклічний	100	0,5	1526	120579,91
	300		7983	119590,380
	500		21793	122848,955

На даному наборі різниця в часі становить 28,4, 82,5 та 129,4 секунди відповідно на розмірів популяції. Довжина знайденого маршруту стабільно залишається меншою за інші в середньому на 40% (рисунок 4.4).

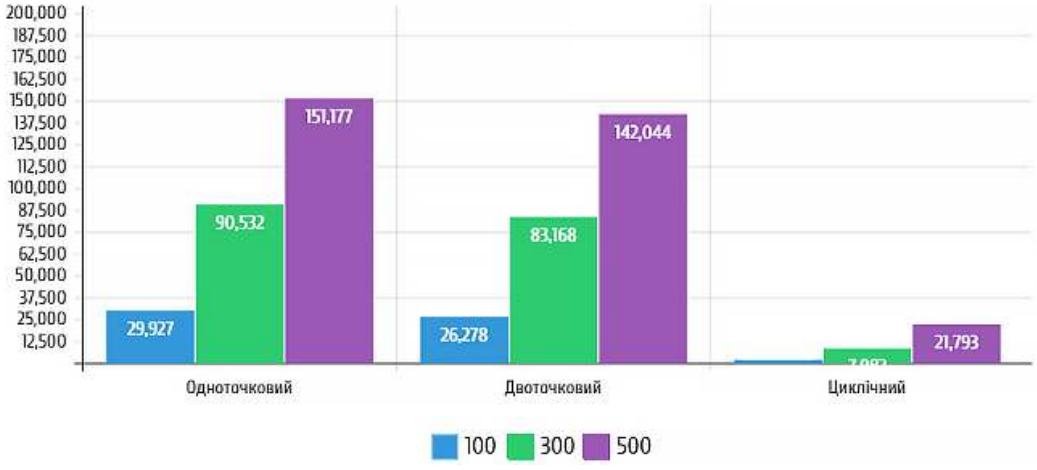

Рисунок 4.3 – Діаграма оцінок часу знаходження найкращих рішень для різних варіантів кросовера та розмірів популяцій на графі з 500 вершин

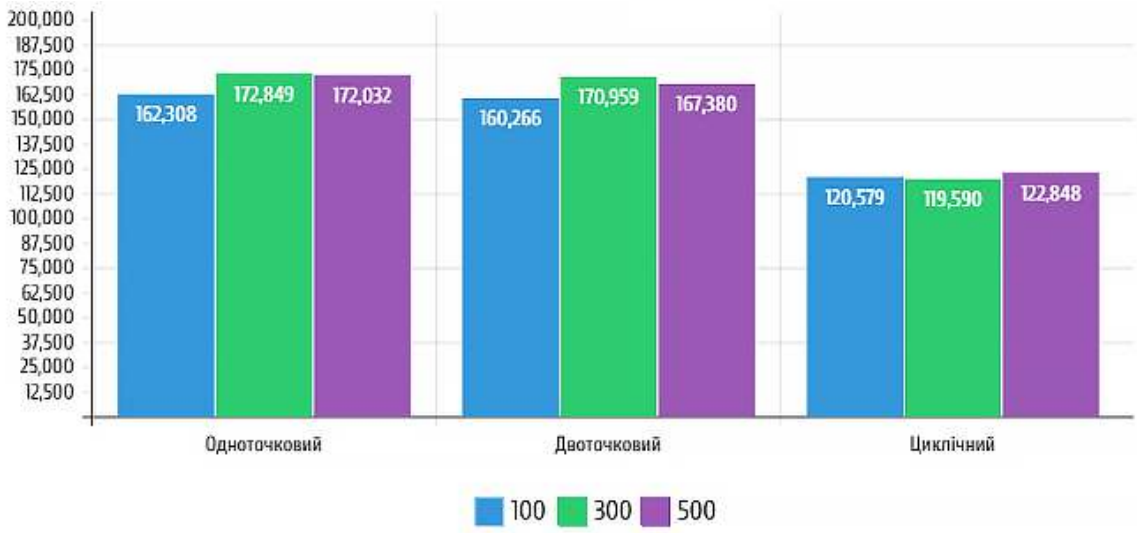

Рисунок 4.4 – Діаграма оцінок довжини маршруту найкращих рішень для різних варіантів кросовера та розмірів популяцій на графі з 500 вершин

4.3 Дослідження роботи ГА у поєднанні із жадібним алгоритмом

В даному дослідженні проаналізовано поєднання генетичного та жадібного алгоритму для покращення шуканого рішення (таблиця 4.3). Жадібний алгоритм використовується для покращення частини популяції в кожному поколінні ГА. Такий підхід поєднує в собі глобальний пошук ГА та локальний пошук жадібного алгоритму. Дослідження проведено з урахуванням попередніх наборів даних.

З результатів роботи розглянутих алгоритмів можна побачити, що використання жадібного алгоритму при побудові початкової популяції значно зменшує довжину шуканого маршруту, проте збільшує час його пошуку. В більшості протестованих випадків незалежно від розміру популяції та варіанту кросовера отримано однакову довжину маршруту. Це свідчить про те, що застосування жадібного алгоритму для одного набору даних дає в результаті одне й те саме рішення і різні варіації кросоверу майже не впливають на подальше покращення маршруту.

Таблиця 4.3 – Порівняння результатів застосування ГА у поєднанні із ЖА для різних розмірів популяцій на графі розмірністю 200 вершин

Кросовер	Розмір популяції	Ймовірність мутації	Критерії ефективності	
			Час, мс	Відстань, мм
Одноточковий, ЖА	100	0,5	5976	10932,214
	300		16317	10952,404
	500		26873	10932,214
Двоточковий, ЖА	100	0,5	5344	10873,465
	300		15097	10932,214
	500		24641	10932,214
Циклічний, ЖА	100	0,5	4325	10932,214
	300		12004	10932,214
	500		20052	10932,214

Найменший час пошуку рішення досягається при використанні циклічного кросовера (рисунок 4.5).

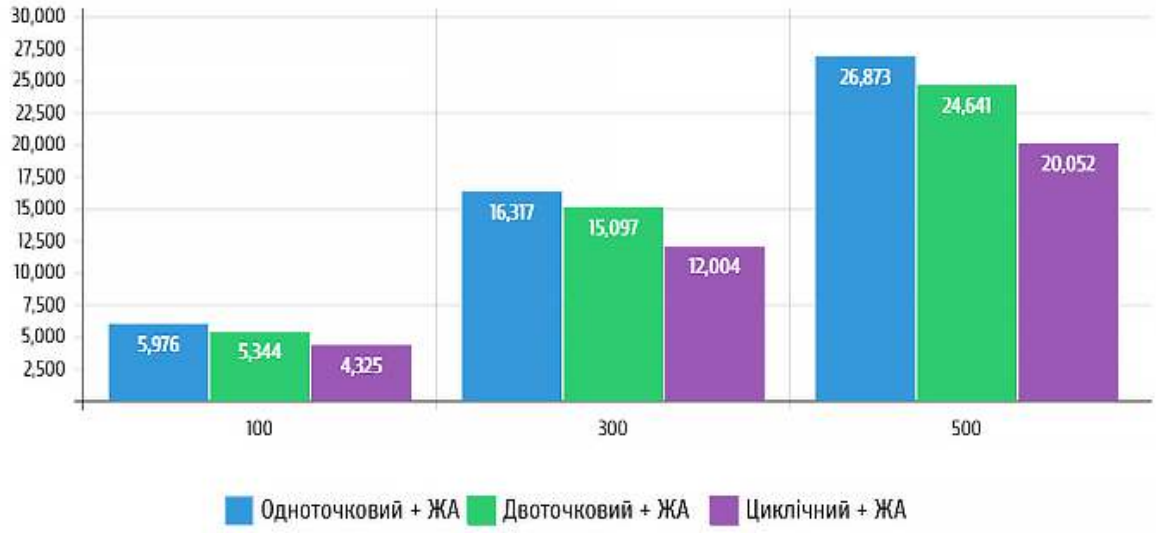

Рисунок 4.5 – Діаграма оцінок довжини маршруту найкращих рішень для різних варіантів кросовера та розмірів популяцій на графі з 500 вершин

4.4 Дослідження впливу кількості ітерацій

Робота генетичного алгоритму обмежується кількістю ітерацій циклу, які в багатьох випадках напряму впливають на ефективність роботи алгоритму. Досліджено вплив кількості ітерацій на ефективність ГА безпосередньо для розробленої системи.

Таблиця 4.4 – Порівняння результатів застосування різної кількості ітерацій для генетичного алгоритму

Кількість ітерацій	Критерії ефективності	Кількість вершин			Кросовер
		200	350	500	
1000	Час, мс	5215	13408	26250	Одноточковий
	Відстань, мм	57045,35	105831,87	166201,42	
2000	Час, мс	10317	26320	52234	
	Відстань, мм	51194,34	99921,00	154004,78	
3000	Час, мс	14563	39941	77934	
	Відстань, мм	49409,04	93979,93	142190,74	
1000	Час, мс	5849	15555	30258	Двоточковий
	Відстань, мм	55727,39	108066,97	159896,70	
2000	Час, мс	10724	30322	60244	
	Відстань, мм	49517,56	100988,87	148998,36	
3000	Час, мс	16024	45425	90089	
	Відстань, мм	46560,12	92835,76	145545,44	
1000	Час, мс	506	885	1396	Циклічний
	Відстань, мм	35572,20	77300,29	121276,02	
2000	Час, мс	793	1379	2024	
	Відстань, мм	32227,14	63419,73	104682,73	
3000	Час, мс	1067	1762	2701	
	Відстань, мм	31735,39	58541,99	89461,49	

Зі збільшенням кількості ітерацій значно покращується показник довжини маршруту. Це пояснюється тим, що алгоритму надається більше часу для знаходження та обробки кращих рішень.

Графік залежності відстаней від кількості ітерацій (рисунок 4.6) демонструє покращення рішення, проте після 2000 ітерацій покращення значно послаблюється. Це свідчить про те, що важливо знаходити баланс між часом та якістю рішень, особливо при роботі з обчислювально витратними задачами.

Рисунок 4.6 – Діаграма залежності відстаней знайдених рішень від кількості ітерацій у циклі генетичного алгоритму

Проведено загальний аналіз результатів дослідження вирішення задачі комівояжера за допомогою генетичних алгоритмів. Найбільшу якість рішення отримано під час використання циклічного кросовера для генетичного алгоритму.

Поєднання генетичного та жадібного алгоритму значно покращую показник довжини маршруту, проте негативно впливає на час знаходження найкращого рішення.

4.5 Дослідження впливу вірогідності мутації

В генетичних алгоритмах вірогідність мутації представляє собою вірогідність того, що кожен конкретний ген в хромосомі буде мутувати при створенні нового гену. Даний параметр допомагає зберегти різноманітність в популяції. Досліджено вплив вірогідності мутації на роботу ГА (таблиця 4.5).

Таблиця 4.5 – Порівняння використання різної вірогідності мутаціїї для генетичного алгоритму

Вірогідність мутації	Критерії ефективності	Кількість вершин			Кросовер
		200	350	500	
0,02	Час, мс	5364	14433	14266	Одноточковий
	Відстань, мм	52199,1	106692,2	110722,8	
0,5	Час, мс	5072	14133	13689	
	Відстань, мм	57453,1	106651,4	106388,5	
1,0	Час, мс	4983	14229	14324	
	Відстань, мм	61070,8	112733,8	108467,1	
0,02	Час, мс	14679	15095	15523	Двоточковий
	Відстань, мм	108525,5	108206,1	109635,5	
0,5	Час, мс	14138	14022	13974	
	Відстань, мм	107567,6	109081,5	107900,9	
1.0	Час, мс	14324	14233	14232	
	Відстань, мм	108178,0	109833,0	106528,1	
0,02	Час, мс	933	927	864	Циклічний
	Відстань, мм	98168,4	97547,4	100473,5	
0,5	Час, мс	801	798	812	
	Відстань, мм	77479,6	74787,9	76391,5	
1,0	Час, мс	957	886	924	
	Відстань, мм	79872,0	76857,6	76959,1	

З отриманих результатів можна побачити, що для отноточкового кросовера для менших наборів вершин (200 та 350) збільшення ймовірності мутації ситуативно зменшує час роботи алгоритму, проте значно погіршує якість знайденого рішення. Довжина знайденого маршруту при значенні вірогідності 0,2 менша на 16% за довжину маршруту при значенні 1,0. На більшому наборі вершин кращі результати отримано при середньому значенні ймовірності мутациі 0,5.

Двоточковий кросовер на найменшому наборі даних дозволяє отримати найкращий результати при середньому значені ймовірності мутації 0,5. Зі збільшенням кількості вершин даний варіант кросовера надає найкоротший маршрут при максимальному значенні ймовірності мутації 1,0.

Найзначніший вплив даного параметру при використанні циклічного кросовера. Найменшу довжину маршруту надає алгоритм при значенні ймовірності мутації 0,5. Різниця довжин при значенні мутації 0,02 та 0,5 в середньому складає 26%. В контексті практичних сценаріїв даний показник значно збільшить складність та вартість маршруту, тому необхідно зберігати баланс між різноманітністю рішень та їх якістю і підбирати значення окремо під конкретні задачі.

ВИСНОВКИ

В ході кваліфікаційної роботи було проведено аналіз можливості вирішення задачі комівояжера шляхом використання різних видів кросоверів генетичного алгоритму таких, як одночковий, двоточковий та циклічний кросовер. Було досліджено використання жадібного алгоритму у поєднанні з генетичним алгоритмом для покращення шуканого рішення. Виявлені переваги та недоліки існуючих підходів для вирішення ЗК.

Для проведення аналізу ефективності обраних варіантів кросоверів генетичного алгоритму було реалізовано програмний застосунок на основі платформи JavaFX на мові програмування Java.

Досліджено ефективність кожного типу кросовера на графах різних розмірностей та для різних розмірів популяції. Проведено аналіз впливу кількості ітерацій на якість знайденого рішення [26].

Для покращення результатів пошуку рішення використане поєднання генетичного алгоритму з іншими методами пошуку найкоротшого шляху.

Проведено аналіз отриманих результатів та визначення можливостей використання реалізованих генетичних алгоритмів для покращення пошуку ефективного рішення та зниження витрат у практичних сценаріях.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Громкович, Ю. Алгоритмизация труднорешаемых задач. Часть II [Текст] / Ю. Громкович, Б. Ф. Мельников. – 2014. – 612 с.

2. Mitchell, J. E. Branch-and-Cut Algorithms for Combinatorial Optimization Problems [Текст] / J. E. Mitchell // Mathematical Sciences Rensselaer Polytechnic Institute. – Troy. – 1999. – P. 19.

3. Ермоленко, С. В. Исследования решения задачи коммивояжера с помощью островной модели [Текст] / С. В. Ермоленко, В. Г. Кобак. – 2016. – №3. – 7 с.

4. Sathya, N. A review of the Optimization Algorithms on Traveling Salesman Problem [Текст] / N. Sathya, A. Muthukumaravel // Indian Journal of Science and Technology. – 2015. – Т. 29, №8. – С. 1–4.

5. Моторнюк, А. І. Задача комівояжера [Текст] / А. І. Моторнюк, А. М. Лаврів // Збірник наукових праць IX Всеукраїнської науково-практичної конференції курсантів та студентів. – 2022. – С. 76–77.

6. Applegate, D. The Traveling Salesman Problem: A Computational Study [Текст] / D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook. – Princeton University Press, 2006. – 562 p.

7. Gutin, G. The Traveling Salesman Problem and Its Variations [Текст] / G. Gutin, A. P. Punnen. – Kluwer Academic Publishers, 2002. – 40 p.

8. Goldberg, D. E. Alleles, loci and the traveling salesman problem [Текст] / D. E. Goldberg, R. Lingle // Proceedings of the First International Conference on Genetic Algorithms. – 1985. – P. 154–159.

9. John, E. Introduction to Automata Theory, Languages and Computation [Текст] / E. John, Rajeev Motwani, D. Jeffrey, A. Ullman. – 2001. – 550 p.

10. Приклад узагальненої задачі комівояжера. ЗК з кластерами [Електронний ресурс] – Режим доступу : www/ URL: https://dou.ua/forums/topic/39699/ – 20.10.2023 р. – Загол. з екрану.

11. Громкович, Ю. Алгоритмизация труднорешаемых задач. Часть I. Простые примеры и простые эвристики [Текст] / Ю. Громкович, Б. Ф. Мельников. – 2014. – 360 с.

12. Korte, B. Combinatorial Optimization: Theory and Algorithms [Текст] / B. Korte, J. Vygen. – Algorithms and Combinatorics, 2018. – 455 p.

13. Van Breedam, A. A Tour of Recent Developments in Metaheuristics for the Traveling Salesman Problem [Текст] / A. Van Breedam, P. Vansteenwegen, W. Souffriau // European Journal of Operational Research. – 2018. – P. 401–412.

14. Генетичні алгоритми. Ключові поняття та методи реалізації [Електронний ресурс] – Режим доступу : www/ URL: http://www.znannya.org/?view=ga_general. – 20.10.2023 р. – Загол. з екрану.

15. Osman, I. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem [Текст] / I. Osman // Annals of Operations Research. – 1993. – P. 421–451.

16. Володина, Е. В. Практическое применение алгоритма решения задачи коммивояжера [Текст] / Е. В. Володина, Е. А. Студентова // Инженерный вестник Дона. – 2015. – Т. 2, №2. – С. 1–13.

17. Жуковський, В. В. Основи проектування систем штучного інтелекту та розпізнавання образів [Текст] / В. В. Жуковський. – Методичні вказівки, 2016. – 53 с.

18. Mitchell, M. An Introduction to Genetic Algorithms [Текст] / M. Mitchell. – MIT Press, 1996. – 221 p.

19. Олійник, А. О. Розв’язання задачі комівояжера на основі еволюційного моделювання [Текст] / А. О. Олійник, Є. М. Федорченко, О. О. Степаненко, М. С. Рудь // Реєстрація, зберігання і обробка даних. – 2019. – Т. 21, №3. – С. 21–41.

20. Естественный отбор в мире битов. АИ, ГА, Нейронные сети и Генетические алгоритмы. Виды генетических алгоритмов [Електронний ресурс] – Режим доступу : www/ URL: http://algolist.ru/ai/ga/index.php. – 20.10.2023 р. – Загол. з екрану.

21. Ho, W. A hybrid genetic algorithm for the multi-depot vehicle routing problem [Текст] / W. Ho, G. T. Ho, P. Ji, H. C. Lau // Engineering Applications of Artificial Intelligence. – 2008. – Vol. 21, № 4. – P. 548–557.

22. Гудман, С. Введение в разработку и анализ алгоритмов: учебное пособие [Текст] / С. Гудман, С. Хидетниеми. – 1981. – 368 с.

23. Базилевич, Р. Дослідження ефективності існуючих алгоритмів для розв’язання задачі комівояжера [Текст] / Р. Базилевич, Р. Кутельмах // Вісник національного університету «Львівська політехніка». – 2009. – С. 235–244.

24. Алгоритм мурахи [Електронний ресурс] – Режим доступу : www/ URL: http://dn.khnu.km.ua/dn/k_default.aspx?M=k1113&T=14&lng=1&st=0. – 20.10.2023 р. – Загол. з екрану.

25. Жадные алгоритмы [Електронний ресурс] – Режим доступу : www/ URL: https://habr.com/ru/articles/120343/ – 25.10.2023 р. – Загол. з екрану.

26. Іващенко, Г. С. Вирішення задачі комівояжера за допомогою генетичних алгоритмів [Текст] / Г. С. Іващенко, А. О. Онипченко // Проблеми інформатизації. – 2023. – Т. 3. – С. 79.