Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму»

Виконав: студент 2-го курсу, групи 3КН-22м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Гнаповський О.І.
(прізвище та ініціали)

Керівник: к.т.н., ст.в. каф. КН
Петришин С.І
(прізвище та ініціали)
«07» 12 2023 р.

Опонент: к.т.н., доцент каф. АІІТ
Барабан М.В.
(прізвище та ініціали)
«07» 12 2023 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«08» 12 2023 р.

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.

29.08 **2023 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гнаповський Олег Ігорович
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму
керівник роботи к.т.н., ст.в. кафедри КН Петришин С.І затверджені наказом вищого навчального закладу від "29" 08 2023 року №247

2. Строк подання студентом роботи 13. 11. 2023 року

3. Вихідні дані до роботи:
1) максимальна кількість міст - 500); 2) можливість рандомізації задача наявна; 3) застосування при вирішенні задачі 3 технологій паралелізму 4) об'єктно-орієнтована мова програмування, ОС Windows

4. Зміст текстової частини:
Вступ, аналіз предметної області задачі комівояжера, проєктування інформаційної технології розв'язання задачі комівояжера, реалізація інформаційної технології розв'язання задачі комівояжера з генетичним алгоритмом, економічна частина, висновки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
Алгоритм роботи програмного забезпечення, структура десктопного додатку, UML діаграми, робочі вікна програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Петришин С.І., ст. в,. каф. КН	29.08.23	10.11.23
4	Кавецький В. В., к. е. н., доц. каф. ЕПВМ	16.10.23	20.10.23

7. Дата видачі завдання 29.08. 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз предметної області задачі комівояжера	01.09.23 - 07.09.23	
2	Розробка алгоритму роботи задачі комівояжера	08.09.23 - 24.09.23	
3	Реалізація програмного забезпечення задачі комівояжера	25.09.23 - 15.10.23	
4	Підготовка економічної частини	16.10.23 - 20.10.23	
5	Апробація та/або впровадження результатів дослідження	21.10.23 - 05.11.23	
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	06.11.23 - 10.11.23	

Студент _____ (підпис) Гнаповський О.І.

Керівник роботи _____ (підпис) Петришин С.І.

АНОТАЦІЯ

УДК 519.85

Гнаповський О.І. Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму. Магістерська кваліфікаційна робота зі спеціальності 122 – Комп’ютерні науки, освітня програма – Системи штучного інтелекту. Вінниця: ВНТУ, 2023. 104 с.

На укр. мові. Бібліогр.: 20 назв; рис.: 14; табл. 13.

У даній магістерській роботі досліджується інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму. Робота складається з трьох основних розділів. У першому розділі проводиться аналіз предметної області задачі комівояжера, обгрунтовується сфера її застосування та проводиться аналіз програм-аналогів. Другий розділ присвячений проектуванню системи, що реалізовує задачу комівояжера, включаючи проектування алгоритму роботи системи генетичного алгоритму, UML-діаграм та структурної схеми. У третьому розділі описується реалізація задачі комівояжера з генетичним алгоритмом, включаючи обґрунтування вибору мови програмування та середовища розробки, а також результати тестування розробленого додатку. Робота дозволяє зробити висновки про ефективність розв'язання задачі комівояжера із застосуванням генетичного алгоритму та може бути корисною для фахівців у галузі оптимізації маршрутів.

Ключові слова: задача комівояжера, генетичний алгоритм, мутація.

ABSTRACT

Hnapovskyi O.I. Application of genetic algorithms to solve the traveling salesman problem. Master's thesis on specialty 122 - Computer science, educational program - Artificial intelligence systems. Vinnytsia: VNTU, 2023. 104 with.

In Ukrainian speech Bibliography: 20 names; Fig.: 14; table 13.

This master's thesis examines the application of genetic algorithms to solve the salesman's problem. The work consists of three main sections. In the first section, an analysis of the subject area of the traveling salesman task is carried out, the scope of its application is substantiated, and an analysis of similar programs is carried out. The second section is devoted to the design of a system that implements the traveling salesman's task, including the design of the work algorithm of the genetic algorithm system, UML diagrams and a structural diagram. The third chapter describes the implementation of the traveling salesman problem with a genetic algorithm, including the justification of the choice of programming language and development environment, as well as the results of testing the developed application. The work allows us to draw conclusions about the effectiveness of the application of genetic algorithms to solve the traveling salesman problem and may be useful for specialists in the field of route optimization.

Keywords: salesman's problem, genetic algorithm, mutation.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Задача подорожі торговця, відома також як задача комівояжера, належить до одних з найбільш ключових і поширених задач комбінаторної оптимізації. Ця проблема має велике значення у різних сферах, таких як логістика, транспорт, електроніка та інші галузі, де виникає необхідність визначити оптимальний шлях для пройдення через набір пунктів.

Людство стикається з цією задачею у багатьох аспектах своєї діяльності. Наприклад, у логістиці і транспорті, ефективне планування маршрутів торговців може суттєво зекономити час і ресурси, забезпечуючи оптимальне розташування та послідовність відвідування різних місць. У сфері електроніки і виробництва, задача подорожі торговця може виникнути при оптимізації маршрутів обслуговування обладнання чи збирання деталей.

Важливість цієї задачі полягає в тому, що визначення найкоротшого шляху через декілька пунктів дозволяє мінімізувати витрати часу, енергії і ресурсів, що стає вирішальним фактором для підприємств та організацій. Застосування алгоритмів та методів, таких як генетичні алгоритми, для розв'язання цієї задачі може допомогти в оптимізації процесів і покращенні загальної продуктивності в різних галузях.

Але з ростом кількості міст точні методи стають непрактичними для вирішення цієї задачі. Це особливо відчутно у реальних сценаріях. Тому дослідження методів оптимізації для цієї задачі залишається актуальним в сучасній науці.

Один із потенційних методів — генетичні алгоритми. Вони ґрунтуються на принципах природного відбору та генетики, і можуть бути застосовані для розв'язання задач оптимізації в різних сферах, включаючи

подорож торговця. Проте, наразі було проведено обмежене число досліджень з використанням генетичних алгоритмів для цієї задачі.

Результати таких досліджень можуть послужити основою для подальшого розвитку методів оптимізації, які використовують генетичні алгоритми. Вони можуть бути використані для вирішення реальних проблем, пов'язаних із оптимізацією маршрутів.

Зв’язок роботи з науковими програмами, планами, темами

Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету та плану наукової та навчально-методичної роботи кафедри.

Мета та завдання дослідження. Метою дослідження є підвищення ефективності побудови оптимального маршруту для розв'язання задачі комівояжера. Для досягнення цієї мети необхідно вирішити наступні завдання:

- провести аналіз галузі застосування задачі комівояжера;
- проаналізувати переваги та недоліки програм аналогів;
- розробити інформаційну технологію розв'язання задачі комівояжера із застосуванням генетичного алгоритму;
- розробити алгоритм оптимізації розв'язання задачі комівояжера;
- здійснити програмну реалізацію інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму;
- здійснити аналіз функціонування створеного програмого забезпечення;
- здійснити економічне обгрунтування доцільності розробки нової інформаційної технології.

Об’єкт дослідження – процес розв'язання задачі комівояжера.

Предметом дослідження є програмні засоби для розв'язання задачі комівояжера.

Методи дослідження. У даній роботі використані наступні методи дослідження: аналіз предметної області задачі комівояжера, проектування системи

на основі генетичних алгоритмів, програмування та тестування розробленого додатку.

Наукова новизна отриманих результатів. Удосконалено інформаційну технологію розв'язання задачі комівояжера, що відрізняється від існуючих удосконаленою моделлю формування маршруту, що забезпечило підвищення ефективності побудови оптимального маршруту для розв'язання задачі комівояжера.

Практичне значення одержаних результатів полягає в тому, що:

- розроблено алгоритм побудови маршруту комівояжера з використанням генетичного підходу;
- здійснено програмну реалізацію інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується коректністю постановки завдання, коректністю використання математичного апарату методів дослідження, експериментальними дослідженнями тестування програмної реалізації інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму.

Особистий внесок здобувача. Усі наукові результати, наведені у магістерській кваліфікаційній роботі, отримані автором самостійно. У роботах, опублікованих у співавторстві, автору належать такі результати: [1] – описано особливості побудови маршруту комівояжера.

Апробація результатів роботи. Результати роботи були апробовані на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» (м. Вінниця, Україна, 2023 р.) [1].

Публікації. За результатами магістерської кваліфікаційної роботи опубліковано тезу доповіді на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАДАЧІ КОМІВОЯЖЕРА

1.1 Обґрунтування сфери застосування задачі комівояжера

Задача комівояжера (TSP) належить до числа відомих NP-складних завдань, що означає, що немає точних алгоритмів для знаходження оптимального маршруту за поліноміальний час [1]. Один з методів для отримання точного розв'язку ТСП - це метод вичерпного перебору та оцінки. Цей підхід включає у себе генерацію всіх можливих турів та їх оцінку згідно з тривалістю кожного туру. Тур з найменшою довжиною вважається оптимальним [2].

В сучасності багато компаній і установ стикаються з задачами TSP. Наприклад, компанії з доставки зіткнені з проблемами, коли маршрути до місць призначення не є оптимальними. У сфері доставки час - це репутація компанії. Тому для досягнення цілей потрібна система, яка забезпечить оптимальний маршрут, щоб час доставки був мінімальним. Тут виникає потреба у програмі, яка може оптимізувати рішення TSP, і генетичний алгоритм може забезпечити швидке отримання оптимальних рішень.

Генетичний алгоритм (GA) - це метод пошуку у сфері інформатики, який застосовується для задач оптимізації, таких як TSP. В останні роки GA став популярним підходом, оскільки він демонструє більшу ефективність у пошуку оптимальних або найближчих до них рішень [5, 6]. GA базується на популяції, де кращі рішення підтримуються для отримання оптимальних результатів за допомогою генетичних операцій [3]. Однак GA має свої обмеження: коли зійде на локальний оптимум, результат може бути не оптимальним. Одним із способів подолання цього є гібридизація з локальним пошуком, що покращує продуктивність GA, забезпечуючи оптимальніші рішення в задачах оптимізації, таких як TSP [7, 9].

Наведемо декілька прикладів застосування задачі комівояжера на практиці.

Пряме використання задачі комівояжера (TSP) знаходиться в області свердлення друкованих плат (PCB). При об’єднанні провідників на різних шарах

або розташуванні контактів інтегральних схем, необхідно створювати отвори у платі різного розміру. Процес свердління різноманітних отворів потребує зміни обладнання та переміщення головки верстата, що забирає багато часу. Тому, оптимальним рішенням є свердлення отворів одного діаметру послідовно, змінюючи свердло лише після завершення групи отворів одного розміру.

Такий процес свердління можна порівняти з серією задач комівояжера, де кожен діаметр отвору є "містом", а початкове положення - це верстат, а всі потрібні отвори - ці міста. Відстань між містами визначається часом переміщення головки верстата. Основна мета - мінімізувати час руху головки верстата для оптимізації процесу свердління.

Також, для забезпечення рівномірного потоку газу через турбіни використовуються вузли сопло-направляючої лопатки. Правильне розташування цих лопаток впливає на ефективність турбіни (зменшення вібрації, підвищення рівномірності потоку, зменшення витрати палива). Проблему оптимального розміщення лопаток можна сформулювати як задачу комівояжера з урахуванням специфічних вимог щодо цільової функції.

Аналіз структури кристалів, що використовує методику задачі комівояжера (TSP), має важливе практичне застосування у рентгенівській дифракції. В процесі дослідження структури кристалічного матеріалу використовується рентгенівський дифрактометр для отримання інформації про кристалічну структуру. Однак позиціонування для таких експериментів вимагає значних зусиль, оскільки потрібно реалізувати значну кількість позицій для отримання даних. Переміщення чотирьох двигунів для кожного експерименту та точне розрахування часу між позиціями є критичними. Оптимальне позиціонування стає ключовим аспектом для мінімізації часу проведення експерименту, вирішуючи проблему комівояжера в контексті дифракції.

У іншому сценарії, пов’язаному з’єднанням компонентів на платі комп’ютера, виникає схожа проблема оптимізації маршруту, але вже у контексті з’єднання модулів на платі. У цьому випадку потрібно підключити певний набір контактів, де кожен контакт повинен мати не більше двох проводів. Це призводить

до пошуку найкоротшого гамільтонового шляху з невизначеними початковою та кінцевою точками, аналогічно задачі комівояжера з невизначеними вершинами. Подібна ситуація має місце і в проводці тестової шини для випробування плати, де потрібно з'єднати всі модулі через плату визначеним способом. Ці задачі також можна сформулювати як проблему Гамільтонового шляху з нелінійними відстанями та заданими точками початку та кінця.

Проблема, пов'язана з обробкою матеріалів на складі, може бути порівняна з задачею комівояжера (TSP) в контексті комплектації замовлень. Припустимо, що на складі є замовлення на певний набір товарів, розташованих у різних місцях. Транспортні засоби повинні зібрати ці предмети для доставки клієнту. У цьому випадку місця зберігання товарів можна розглядати як вузли графа, а відстань між ними визначається часом переміщення транспортного засобу. Оптимізація маршруту для мінімізації часу доставки стає аналогічною до пошуку найкоротшого маршруту у задачі комівояжера.

Іншим прикладом є ситуація, де на поштових скриньках у місті n потрібно проводити спорожнення протягом обмеженого часу, скажімо, за 1 годину щодня. Вирішення цієї проблеми полягає в пошуку оптимальної кількості вантажівок для збирання та вивезення вантажівки з мінімальним часом, використовуючи цю кількість вантажівок. Аналогічно, в ситуації, де постачальник повинен задовольнити запити n клієнтів, задача полягає в розподілі товарів по вантажівках та створенні графіку доставки для кожної вантажівки, мінімізуючи загальну відстань подорожі та дотримуючись обмежень потужності кожної вантажівки.

Ці задачі, що поєднують обмеження часу та потужності, є загальними в різних областях. Вони можуть бути перетворені на задачі TSP, якщо визначені обмеження щодо кількості вантажівок чи часу необхідності. Це відкриває можливості застосування методів, що використовуються для розв'язання TSP, для знаходження ефективних рішень у цих практичних сценаріях.

Задача TSP застосовується у сценаріях маршрутизації та планування, де важлива робота з кількома об’ктами. Деякі з прикладів використання:

- **о**дне з ключових застосувань mTSP виникає у плануванні роботи друкарського верстата для періодичного видання з різними тиражами. Це включає п'ять пар циліндрів, які одночасно друкують рулони паперу та обидві сторони сторінки. Вирішення полягає у визначенні форм, які використовуватимуться на кожному циклі та тривалості циклу, де витрати на зміну номерних знаків є важливими чинниками;

- вивчення проблеми планування маршрутів автобусів за допомогою mTSP з певними обмеженнями. Мета - знайти оптимальну модель завантаження автобусів для мінімізації кількості маршрутів та загальної пройденої відстані, уникнення перевантажень і дотримання політики обмеження часу на маршрути;

- аналіз варіантів маршрутів для перенесення депозитів між банківськими відділеннями та центральним офісом з метою мінімізації загальних витрат;

- використання багатоперіодного TSP для планування інтерв'ю між туристичними брокерами та продавцями в галузі туризму, де кожному брокеру відповідає певний набір стендів продавців;

- оптимізація замовлень на стані гарячої прокатки у металургійній промисловості з метою мінімізації витрат, розглядаючи замовлення як міста та використовуючи вартість переходу між ними як критерій;

- визначення оптимального маршруту для досягнення цілей місії за мінімально можливий час, де планувальник використовує mTSP для визначення шляху для планування кількох цілей та повернення до базового міста;

- використання mTSP для оптимізації розміщення геодезичних мереж у глобальній навігаційній системі.

Також існує цікавий випадок застосування mTSP в області розробки глобальних навігаційних супутникових систем (GNSS), які мають важливе значення для різноманітних сфер, включаючи раннє попередження про катастрофи, контроль за навколишнім середовищем та агропромисловість.

При розробці геодезичних мереж для GNSS використовуються супутникові системи, які допомагають визначити точне географічне положення об'єктів на землі та в просторі. Пристрої на базі супутникового обладнання реєструють

координати приймачів через серію сеансів спостереження. Коли виникає ситуація з кількома приймачами або різними робочими періодами, задача пошуку оптимального порядку сеансів для цих приймачів може бути сформульована як проблема багатопризначного комівояжера (mTSP).

Розробка глобальних навігаційних супутникових систем (GNSS) часто стикається з задачами оптимізації послідовності спостережень для приймачів, що можна сформулювати як проблему багатопризначного комівояжера (mTSP). GNSS використовує супутникові системи для точного визначення географічного положення об'єктів на землі та в просторі.

У сценарії GNSS кілька приймачів можуть використовуватися для одночасного збору даних з різних точок. Мета полягає в ефективному плануванні послідовності спостережень для цих приймачів таким чином, щоб мінімізувати загальну відстань або час, потрібний для збору інформації. Це включає визначення оптимального порядку відвідування різних місць або точок спостережень за допомогою приймачів GNSS.

Отже, mTSP застосовується для оптимізації послідовності збору даних приймачами GNSS з метою ефективного визначення географічного розташування об'єктів у світі. Це важливо для забезпечення точності, ефективності та оптимального використання супутникових систем у сферах, де потрібна точна геолокація та навігація.

1.2 Аналіз програм-аналогів, призначених для розв'язання задачі комівояжера

Генетичні алгоритми чудово підходять для розв'язання складних задач, зокрема, задачі комівояжера. Ці алгоритми знаходять оптимальний маршрут у графі, що проходить через всі точки з мінімальними витратами.

1. Concorde TSP Solver. Ця програма ефективно впорається з графами до 85 000 вершин. Вона надійно розв'язує задачі комівояжера та має відкритий вихідний код, що дозволяє безкоштовно користуватися нею (рис. 1.1).

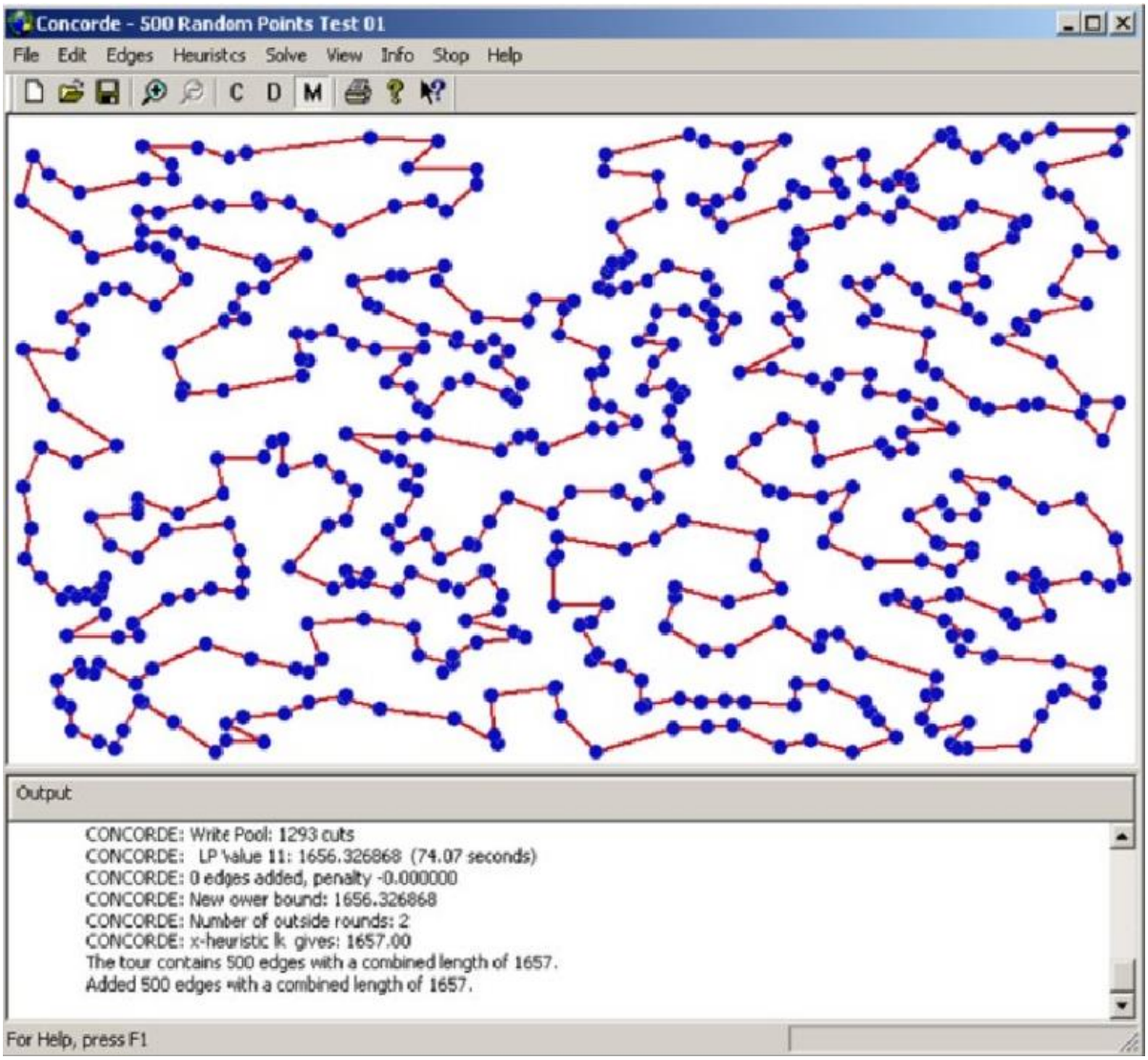


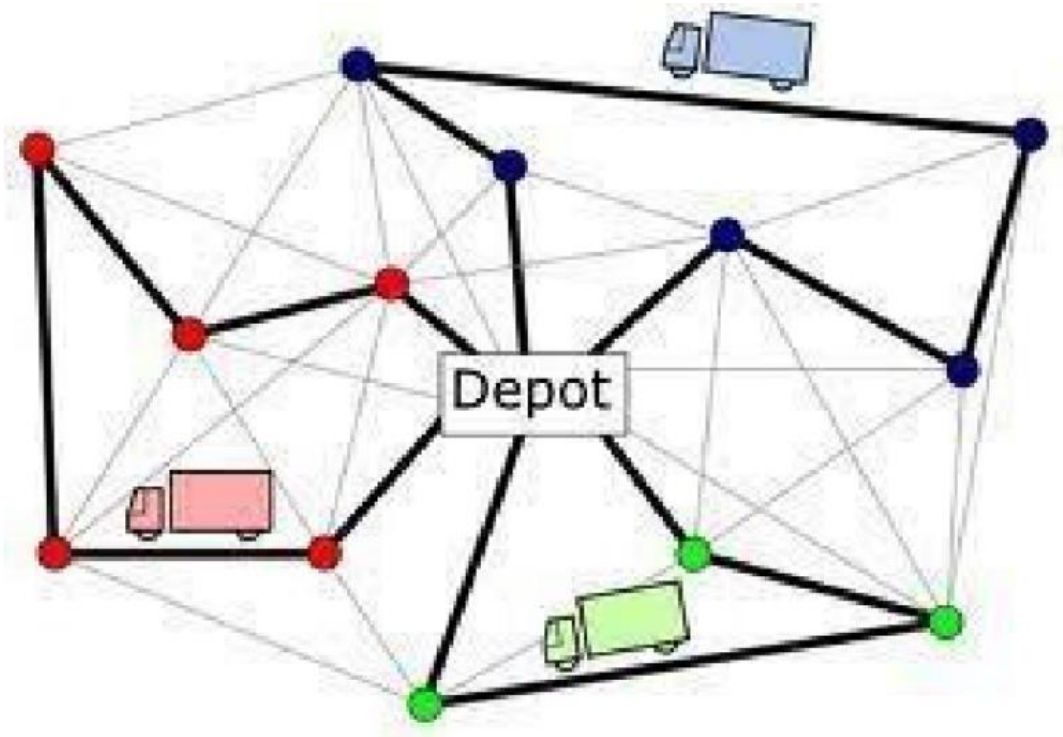
Рисунок 1.1 - Інтерфейс програми Concorde TSP Solver

2. LKH (Lin-Kernighan-Helsgaun). Цей інструмент базується на алгоритмі Ліна-Кернігана та демонструє високу ефективність при розв'язанні складних задач. Він здатний працювати з великими графами і, подібно до Concorde TSP Solver, має відкритий вихідний код.

Подібно до Concorde TSP Solver, LKH також має відкритий вихідний код, що надає користувачам можливість вивчати, модифікувати та адаптувати алгоритм під свої потреби. Це робить його не лише потужним інструментом для розв'язання конкретних задач, але й інструментом, що може бути оптимізований та адаптований для використання в різних областях.

Однією з ключових переваг LKH є його здатність оптимізувати шляхи у графах, що мають значний розмір, забезпечуючи при цьому ефективність та

точність результатів. Цей інструмент може бути використаний у широкому спектрі застосувань, від логістики та транспорту до наукових досліджень та маршрутизації в мережах (рис. 1.2).



Lin-Kernighan-Helsgaun-3 [12]

Рисунок 1.2 - Інтерфейс програми Lin-Kernighan-Helsgaun

3. Ant Colony Optimization (ACO). Ця програма імітує поведінку мурах для знаходження оптимального маршруту. Вона ефективно працює навіть з великою кількістю вершин та може додатково враховувати обмеження. Як і попередні рішення, ACO має відкритий вихідний код і є безкоштовною.

ACO моделює процес, схожий на той, що відбувається в природі, де мурахи залишають сліди феромонів під час свого руху. Ці феромони привертають інших мурах і можуть служити покажчиком оптимального шляху. Алгоритм враховує і підсилює шляхи з більшими концентраціями феромонів, що дозволяє системі знаходити найкращі рішення.

Також, схоже на попередні рішення, ACO відкритий для розробників і користувачів, надаючи можливість не лише використовувати його для розв'язання

конкретних задач, але й адаптувати та вдосконалювати його функціонал за необхідності.

Безкоштовна доступність та відкритий вихідний код роблять ACO привабливим вибором для науковців, розробників та інженерів, які шукають ефективні методи оптимізації маршрутів. Його успішна застосованість в різних галузях, включаючи логістику, транспорт, телекомунікації та багато інших, підтверджує його універсальність та практичну цінність (рис. 1.3).

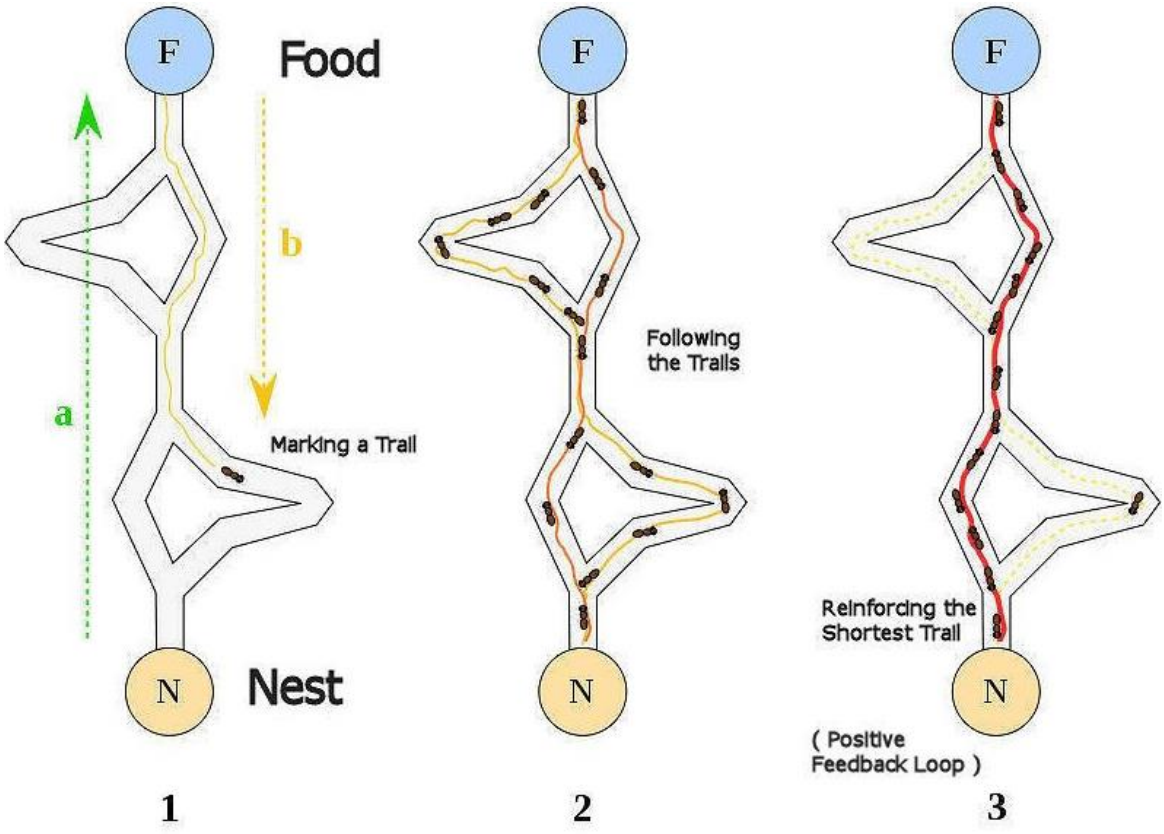


Рисунок 1.3 - Інтерфейс програми Ant Colony Optimization

Застосування цих програм допомагає швидко та ефективно розв'язувати складні задачі комівояжера, використовуючи генетичні алгоритми для пошуку оптимального шляху через всі точки (табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз програм-аналогів задачі комівояжера

Додаток	Особливості	Переваги	Недоліки
TSPLIB95	Безкоштовний, може працювати з файлами у форматі TSPLIB	Є багато різних графів, можливість порівняти різні алгоритми та реалізації	Не має графічного інтерфейсу, тільки командний рядок
Concorde TSP Solver	Може розв'язувати найбільші задачі комівояжера, має реалізації для багатьох мов програмування	Дуже швидкий та точний, використовується в наукових дослідженнях	Платний, складно налаштовувати
Ant Colony Optimization	Використовує мурахиний алгоритм, є безкоштовним та має графічний інтерфейс	Ефективний та точний, може працювати з великими графами	Може бути повільним для складних задач, потребує налаштування параметрів

1.3 Обґрунтування сфери застосування генетичних алгоритмів

Генетичний алгоритм (ГА) - це метод оптимізації, який моделює процеси природного відбору та генетики для пошуку оптимальних рішень у просторі можливих варіантів. Основними концепціями ГА є поняття хромосом, особин і популяції.

Хромосома у ГА відображає потенційні рішення проблеми. Вона представляється у вигляді рядка бітів або символів, де кожен біт або символ є

геном, який кодує певний аспект рішення. Наприклад, якщо ми шукаємо найкращий маршрут, кожен ген у хромосомі може відповідати певній точці в маршруті. Таким чином, хромосома є цифровим представленням потенційного рішення проблеми.

Популяція в ГА складається з набору хромосом, які утворюють перше покоління. Через ітеративний процес еволюції, популяція змінюється та вдосконалюється за рахунок різних операцій, таких як схрещування (комбінація генетичних властивостей), мутація (випадкова зміна деяких генів), та відбір (відбір найкращих рішень для наступного покоління). Кожне покоління популяції намагається покращити розв'язок задачі на основі визначеної функції придатності, яка визначає, наскільки хорошим є кожне рішення у вирішенні поставленої задачі.

Однією з ключових особливостей популяції в ГА є різноманітність. Різноманіття хромосом у популяції сприяє здатності алгоритму вийти з локальних мінімумів та знаходити оптимальні рішення в просторі можливих варіантів.

Процес розв’язування за допомогою генетичного алгоритму (GA) ґрунтується на визначенні параметрів проблеми, що потребує вирішення. Початкова популяція створюється випадковим чином, після чого визначаються функція відповідності та швидкість відбору, а також виконуються генетичні операції, такі як відбір, схрещування та мутація. У випадку виконання умови ітераційної конвергенції, популяція складається з найкращих індивідів; інакше, створюється нове покоління для повторення генетичних операцій у циклі, доки не буде виконано умови завершення.

Процес розв'язання за допомогою генетичних алгоритмів (GA) складається з кількох ключових операцій. Перша операція полягає у виборі M особин з попереднього покоління згідно з ймовірнісним розподілом, де кожна особина має ймовірність Ps (xi) = f (xi) / Eif (xi), де f (xi) відображає придатність моделі xi. Наступна операція, кросовер, включає обмін генетичною інформацією між двома випадково обраними особинами зі старої популяції для створення нащадків. Мутація полягає в процесі генерації нових генів шляхом випадкового вибору бітів для інверсії операції. Це допомагає уникнути застрягання генетичного алгоритму

у локальних оптимумах, розширює область пошуку та оптимізації. У випадку схожості особин та значень пристосованості в популяції, мутація визначає подальшу еволюцію. Процес пошуку оптимального рішення заснований на кодуванні задачі, оцінці продуктивності, виборі рішень та генетичних операціях для отримання нових наборів рішень. Генетичні алгоритми мають численні переваги у вирішенні задач оптимізації, зокрема, високий порядок пошуку, дослідницькі та саморозвиваючі можливості.

Підсумовуючи, процес пошуку оптимального рішення за допомогою алгоритму генетичного пошуку може бути розкритий через такі етапи: 1) кодування задачі, яку треба вирішити; 2) випадкове визначення початкових рішень X (0) = (X1, X2, ..., Xn); (3) оцінка продуктивності поточної групи та обчислення придатності f (xi) для кожного окремого x у даній групі x (t); відбір конкретної кількості рішень з поточного рішення як об'єктів генетичної операції відповідно до результатів оцінки; генетична операція обраного розв'язку для отримання нового набору розв'язків; повернення для оцінки нового рішення; якщо поточне рішення для виконання вимог або еволюційного процесу досягає певного значення, розрахунок завершується або продовжується. Генетичний алгоритм на початкових ітераціях відрізняється добрим і поганим співіснуванням індивідів, з підвищенням кількості ітерацій особи з високою придатністю супроводжуються генетичними. Генетичні методи мають багато переваг для вирішення оптимізаційних завдань, таких як вибір умов. Зокрема, GA має високий рівень пошуку, а сам пошук володіє дослідницькими та саморозвиваючими можливостями.

Звідси, процес визначення найбільш вдалих рішень за допомогою генетичного алгоритму розкладається на декілька ключових кроків. Спочатку відбувається кодування задачі, тобто придумування способу представлення можливих рішень. Далі випадковим чином вибирається початковий набір можливих рішень. Потім оцінюється корисність кожного рішення у поточній групі. Найкращі рішення обираються для подальших генетичних операцій. Кожен наступний крок включає генетичні дії з обраними рішеннями для створення нового набору можливих варіантів. Цей цикл повторюється, поки поточні рішення не

відповідатимуть вимогам або не досягнуть певної якості. У початкових етапах алгоритму якість рішень може варіюватися, але зі збільшенням кількості кроків зазвичай підвищується якість найбільш пристосованих рішень. Генетичні алгоритми мають численні переваги для оптимізації завдань, включаючи вибір умов та високий рівень пошуку, який сприяє дослідженню та самовдосконаленню.

Генетичний алгоритм — результат мультидисциплінарної інтеграції, що перетворився на самоорганізовану та адаптивну інтегровану технологію. Як ефективний глобальний метод пошуку, він широко використовується в різних галузях: від інженерного проектування та виробництва до штучного інтелекту, біоінженерії, соціальних наук та фінансів.

Розвідка нафти — один із прикладів широкого застосування генетичних алгоритмів. Вона використовується для прогнозування видобутку нафтових родовищ, оптимізації розробки, інтерпретації каротажу та визначення розподілу проникності пласта. Незважаючи на численні переваги, є недоліки та обмеження в застосуванні генетичних алгоритмів. Вони можуть швидко сходити до локальної збіжності та не є універсальними. Хоча вони мають великий потенціал, але їхні можливості вирішення завдань і в області розвідки та розробки нафти є обмеженими.

Задача оптимізації компенсації реактивної потужності в енергосистемі представляє собою складну багатопараметричну та багатообмежену задачу змішаного нелінійного програмування. Вона охоплює безперервні змінні (наприклад, напруга вузла та реактивна потужність генератора) та дискретні змінні (такі як механізм РПН перемикача навантаження та перемикальна група компенсаційного конденсатора). Така складність робить оптимізацію цієї задачі надзвичайно важкою.

Традиційні методи лінійного, нелінійного та змішаного цілочисельного програмування мають обмеження у наближеній обробці дискретних змінних і не завжди відповідають реальним потребам планування реактивної потужності. У таких умовах генетичний алгоритм виявляється особливо ефективним для

багатокритерійної гібридної оптимізації, маючи стабільність у збіжності. Він представляє собою розширений метод глобальної оптимізації.

Генетичний алгоритм обмежений різними обмеженнями та оцінюється за допомогою функції придатності. Основна перевага полягає у високому значенні придатності, що дозволяє розвивати і вдосконалювати характеристики рішень. Іншими словами, генетичний алгоритм використовує відбір, кросинговер, мутацію та інші генетичні операції для того, щоб популяція поступово стала оптимальною.

Недоліком простого генетичного алгоритму є тривалість обчислень та його схильність до локальних екстремумів при обмеженій кількості популяції. Тому висувається ідея покращеного генетичного алгоритму або комбінації генетичного алгоритму з іншими методами для підвищення швидкості та точності обчислень.

У цьому випадку генетичний алгоритм використовується разом з імітованим алгоритмом вибору відпалу, щоб уникнути локального оптимального рішення. Цей підхід використовується як система управління реактивною потужністю та напругою в реальному часі, що дозволяє ефективно контролювати роботу енергосистеми, забезпечуючи безпечну та економічну експлуатацію обленерго.

1.4 Висновок до розділу 1

У даному розділі дослідження було проведено дослідження застосування генетичних алгоритмів для розв'язання задачі комівояжера. Було проаналізовано предметну область задачі, обґрунтовано застосування генетичних алгоритмів в розв'язанні даної задачі.

У процесі дослідження було проведено порівняльний аналіз програм-аналогів, що розв'язують задачу комівояжера, та вибрано оптимальний підхід до реалізації генетичного алгоритму.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА ІЗ ЗАСТОСУВАННЯМ ГЕНЕТИЧНОГО АЛГОРИТМУ

2.1 Обгрунтування застосування генетичного алгоритму для розв'язання задачі комівояжера

Основою паралельного генетичного алгоритму (PGA) є генетичний алгоритм (GA), який представляє собою глобальний, адаптивний та ймовірнісний алгоритм оптимізації. Він виник з моделі еволюції органічних систем та ефективно моделює генетику та еволюцію біологічних популяцій у природі. ГА використовує природні принципи еволюції, такі як відбір, кросовер, мутація, видалення та перенесення.

Математично цей еволюційний процес є стандартним алгоритмом пошуку оптимального рішення шляхом ітераційного перебору багатьох елементів у наборі, що належить до класу проблем недетермінованого поліноміального часу (NP). Простий генетичний алгоритм (SGA) можна визначити як SGA = (M, C, F, o, Ps, Pc, Pm, T), де:

- C — це фіксований рядок бітів;
- F — функція оцінки придатності;
- M — початкова популяція біологічних об'єктів;
- Ps, Pc, Pm — ймовірності відбору, кросинговеру та мутації відповідно.

В процесі вирішення задач NP серія GA може привести до збільшення довжини хромосоми великим простором вибірки. Це може спричинити збільшення часової складності алгоритму. Однак ми внесли зміни у послідовний генетичний алгоритм для паралельної обробки, що призвело до зменшення часової складності. PGA використовує дві основні модифікації порівняно з генетичним алгоритмом (рис. 2.1).

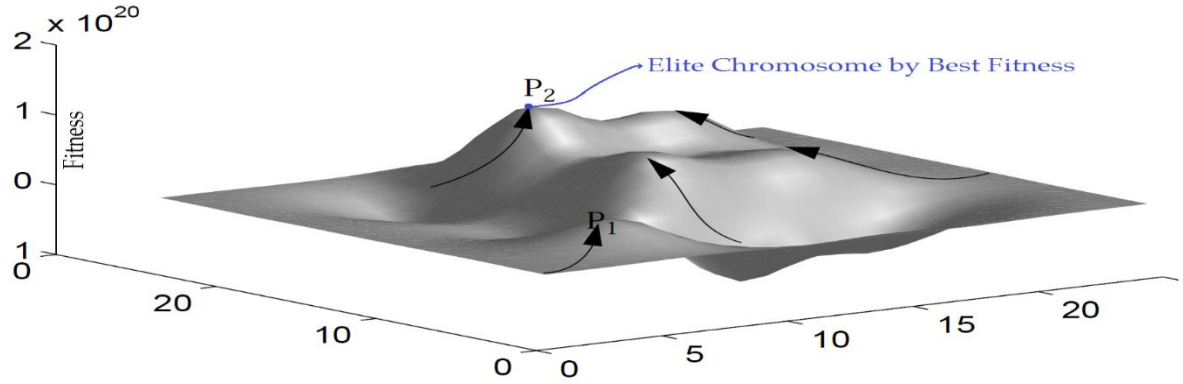


Рисунок 2.1 – Паралельність в генетичному алгоритмі

PGA - це асинхронний алгоритм, що працює з максимальною ефективністю на паралельних комп’ютерах MIMD (Multiple Instruction, Multiple Data - техніка для паралельних обчислень). Його стратегія пошуку базується на обміні інформацією між активними та розумними особами, в той час як GA використовує велику кількість пасивних осіб. Абстрактно, PGA - це паралельний пошук з обміном інформацією між особинами в популяції.

Сьогодні паралельні обчислення широко використовуються для розв'язання складних задач. Однією з добре відомих комбінаторних задач, що вирішуються за допомогою паралельних обчислень, є задача комівояжера.

PGA використовує паралельний підхід до розв'язання задач, розподіляючи обчислення між різними комп'ютерами чи обчислювальними вузлами. Це дозволяє виконувати обчислення швидше, розподіляючи завдання між багатьма ресурсами одночасно.

Основна ідея полягає в тому, щоб розділити популяцію особин на групи, які працюють паралельно, тобто кожна група аналізує і обробляє свої власні дані. Вони можуть обмінюватись інформацією періодично для спільної оптимізації.

PGA дозволяє значно прискорити обчислення завдяки використанню паралельних обчислювань, що особливо корисно при розв'язанні складних задач, де потрібно проводити велику кількість ітерацій або великі обчислювальні ресурси.

Використання паралельних обчислень в PGA дає можливість ефективно вирішувати проблеми, які вимагають значних обчислювальних зусиль,

забезпечуючи при цьому швидкість та ефективність в процесі пошуку оптимального рішення.

Таском комівояжера (TSP) є знаходження найкоротшого маршруту, який проходить через всі міста без повторень. Це важлива задача в інформатиці та застосується в різних сферах, від логістики до робототехніки.

У розв'язанні TSP за допомогою генетичних алгоритмів існують кілька моделей паралельних реалізацій:

1. Одиночне населення "Господар / Раб" (фітнес). Один центральний хостер (господар) керує алгоритмом, а інші підсистеми (раби) обчислюють фітнес функції. Цей підхід ефективний для розподілу обчислювальних завдань.
2. Одиночна популяція дрібнозернистих або клітинних PGA. У цьому випадку, велика задача розбивається на більш дрібні фрагменти, які обробляються окремо. Це дозволяє оптимізувати обчислення.
3. Багатонаселення (зі рівнем міграції). Використовує кілька підпопуляцій, які працюють незалежно одна від одної. Особини періодично мігрують між цими підпопуляціями, сприяючи обміну інформацією та збагаченню.
4. Ієрархічний підхід. Використовує різні рівні підпопуляцій з різними рівнями взаємодії між ними. Це дозволяє побудувати складніші системи з обмеженою взаємодією, сприяючи великій гнучкості.

Кожен з цих підходів має свої переваги та обмеження. Наприклад, одиночна популяція з використанням "раба" може бути ефективною для розподілу завдань, але потребує великого контролю та комунікації між підсистемами. Багатопопуляційний підхід із міграцією дозволяє ефективніше дослідження різних областей пошуку рішень.

Ця модель головний-підлеглий використовується для реалізації генетичних алгоритмів (GA) з однією популяцією. В ній є головний процесор, що виконує операції відбору, кросинговеру та мутації, тоді як підлеглі процесори виконують оцінку функцій придатності окремих особин. Це показано на рисунку 2.2 глобальної моделі головного-підлеглого з єдиною популяцією.

У цій моделі частина популяції розподіляється між підлеглими процесорами

для оцінки придатності кожної окремої особини. Головний процесор також зберігає частину популяції, щоб проводити оцінку паралельно з підлеглими процесорами. Генетичні операції, крім оцінки, виконуються лише головним процесором. Кожен підлеглий процесор отримує від головного процесора частку популяції для обробки на кожному поколінні.

Генетичні операції - це ключовий елемент генетичних алгоритмів, що включають:

1. Відбір (Selection). Це процес вибору найкращих розв'язків для наступного покоління. Особини з більшою придатністю (які краще вирішують проблему) мають більше шансів потрапити у наступне покоління.
2. Кросинговер (Crossover). Ця операція полягає у створенні нових особин шляхом комбінування інформації з двох батьківських особин. Наприклад, для бітових рядків, це може бути обмін підстроками між батьківськими особинами.
3. Мутація (Mutation). Мутація — це випадкові зміни в генотипі окремої особини, які можуть виникати внаслідок помилок або природних варіацій. Ця операція допомагає уникнути застрягання в локальних екстремумах.
4. Елітаризм (Elitism). Це стратегія, що виключає деякі з найкращих особин з поточного покоління і передає їх без змін у наступне покоління. Це допомагає зберегти найкращі розв'язки, уникнути втрати важливої інформації.

Генетичні операції взаємодіють, створюючи нове покоління особин, яке сподівається більш ефективно розв'язати задачу, ніж попереднє покоління (рис. 2.2).

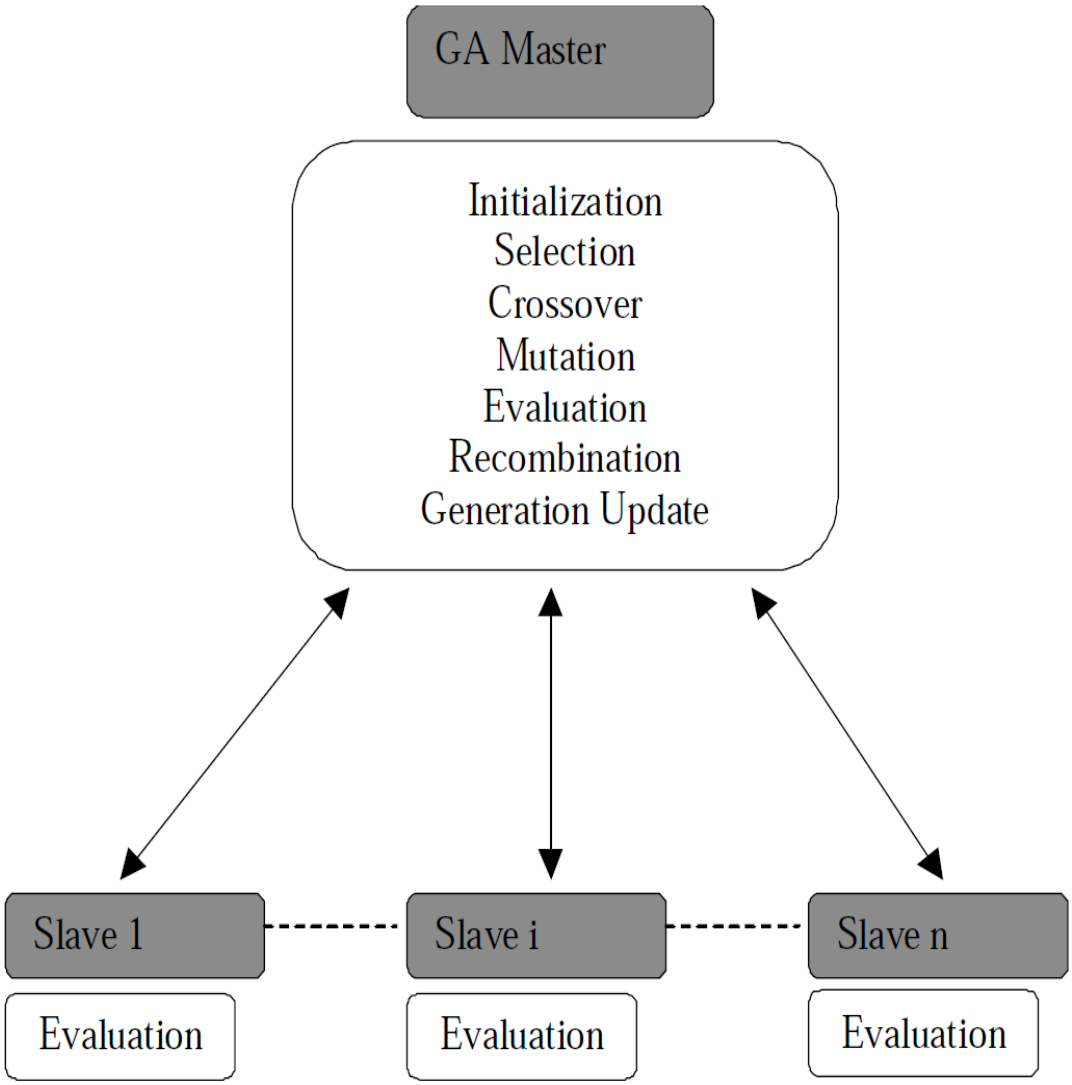


Рисунок 2.2 – Застосування генетичного алгоритму для розв’язання задачі комівояжера

2.2 Розробка UML-діаграм функціонування інформаційної технології розв’язання задачі комівояжера

Стандарт UML, який застосовується до об'єктно-орієнтованого підходу, забезпечує відповідні погляди на систему, так що в усіх термінах система може бути описана зі статичного (структурного) і динамічного аспектів. Він використовується для розробки програмного забезпечення, яке потребує плану, пропонує можливість візуалізації в багатьох вимірах і рівнях деталізації та

підходить для оновлення старих систем. Цілком певно, що така дія спростить процес отримання рішення.

Представимо процес моделювання спочатку через статичні, а потім динамічні діаграми. Це хороший спосіб вирішення, оскільки UML описує вихідний код, моделі допомагають візуалізувати систему такою, якою вона є або якою вона має бути, і дозволяють визначити структуру та поведінку системи. Моделі документують рішення, які ми приймали, і пропонують рішення, якими ми керувалися під час побудови системи та конкретних програм.

Діаграма випадків використання відображає функціональні вимоги, яким повинна відповідати система. Вона складається з одного актора та випадків використання, що випливають з опису генетичних алгоритмів (ініціалізація, оцінка, відбір, схрещування та мутація). Усі випадки вживання у відповідному контексті, і між ними існує взаємодія (рис. 2.3).

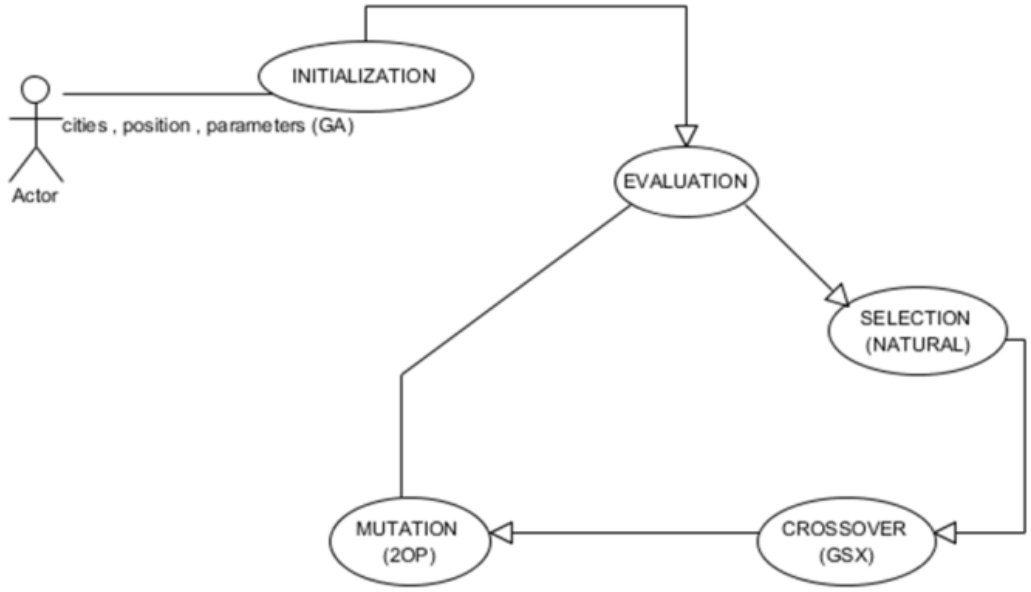


Рисунок 2.3 – Діаграма прецедентів

Діаграми прецедентів – це потужний інструмент у сфері розробки програмного забезпечення, який допомагає визначати функціональність системи з точки зору її користувачів. Ці діаграми моделюють взаємодію між користувачами та системою, визначаючи різні сценарії використання.

У своїй сутності діаграми прецедентів складаються з акторів та прецедентів.

Актори – це будь-які зовнішні сутності, які мають взаємодію з системою, будь то користувачі, інші системи або зовнішні сервіси. Прецеденти – це конкретні дії або функції, які виконує система, щоб задовольнити потреби акторів.

Кожен прецедент відображається у вигляді овалу з назвою, що описує дію, яку виконує система. Актори позначаються у діаграмі прецедентів у вигляді іконок, які представляють різних учасників системи. Взаємодія між акторами та прецедентами відображається за допомогою стрілок, що показують напрямок обміну інформацією чи взаємодії.

Ці діаграми дозволяють розробникам чітко зрозуміти функціональність системи та потреби користувачів. Вони стають основою для подальшого проектування та розробки програмного забезпечення, допомагаючи командам зрозуміти, як система повинна працювати в реальному середовищі.

Діаграми прецедентів також сприяють виявленню можливих проблем у функціональності системи та дозволяють зробити необхідні зміни ще на ранніх етапах розробки. Вони допомагають уникнути непорозумінь між розробниками та клієнтами, оскільки надають зрозуміле та візуальне представлення того, як система буде взаємодіяти з її користувачами.

До акторів діаграми прецедентів належить лише користувач системи, який здійснює операції генетичних алгоритмів над задачею комівояжера, а саме ініціалізацію, оцінювання, формування вибірки, мутацію та кросовер.

Діаграми діяльності – це ще один інструмент в арсеналі моделювання систем, який допомагає візуалізувати послідовність дій, які відбуваються у конкретному процесі або функціональній частині системи. Вони дозволяють описати, як система реагує на зовнішні події, як взаємодіють її елементи та яка послідовність дій відбувається в певному сценарії.

Основні елементи діаграм діяльності – це дії, рішення, відмітки та переходи між ними. Дії представляють окремі кроки або дії, які виконуються в рамках процесу. Рішення вказують на вибір між альтернативними шляхами або діями. Відмітки вказують на певні точки в процесі, які можуть бути використані для відстеження або аналізу. Переходи показують послідовність дій і зв'язки між

різними елементами діаграми.

Ці діаграми стали популярним інструментом у бізнес-аналізі, проектуванні програмного забезпечення, моделюванні бізнес-процесів та управлінні проектами. Вони допомагають командам легше розуміти потоки дій, виявляти можливі проблеми та оптимізувати процеси.

Діаграми діяльності дозволяють не лише описати послідовність подій, але й виявити можливі варіанти розвитку подій, враховуючи різні сценарії та умови. Вони стають ефективним засобом спілкування між розробниками, бізнес-аналітиками та клієнтами, оскільки надають зрозуміле та візуальне уявлення про те, як система повинна працювати в реальних умовах.

Узагальнюючи, діаграми діяльності є важливим інструментом для моделювання та аналізу процесів в системах, що дозволяє зрозуміти та оптимізувати послідовність дій у різних сценаріях взаємодії системи з її елементами та зовнішніми факторами.

Діаграма діяльності – це дії, які виконуються, а саме ця діаграма дає загальне уявлення про дії, які далі декомпозуються (рис. 2.4). Потім наступні діаграми представляють декомпозицію згаданих видів діяльності на піддіяльності, що містять конкретні дії (рис. 2.5, 2.6).

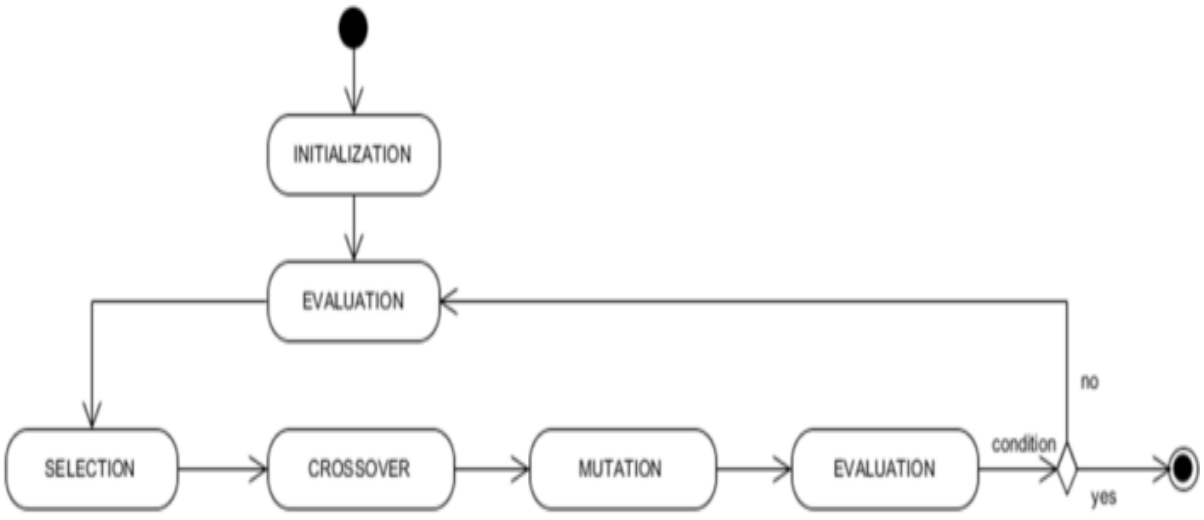


Рисунок 2.4 – Діаграма діяльності

Рисунок 2.5 дає візуальне представлення того, як розраховується процес відбору, так звана функція придатності та ймовірність для кожного члена окремо, тоді як на рисунку 2.6 ми вказали, як використання отриманих значень із попередньої діяльності та процедури відбору учасників ґрунтується на їх вірогідності.

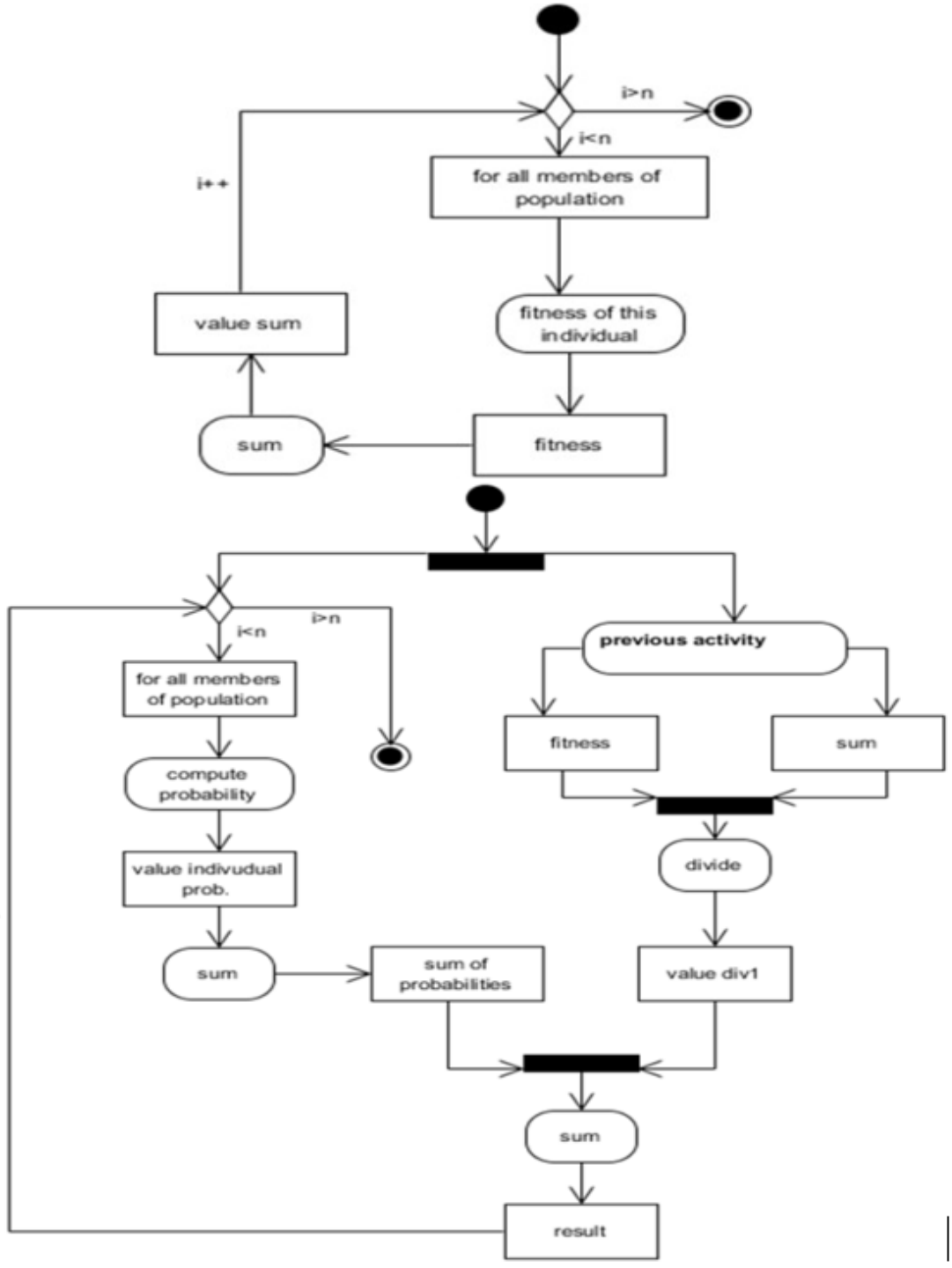


Рисунок 2.5 – Деталізована діаграма діяльності генетичної селекції

Розширення діаграми діяльності – це процес розгортання деталей, який зазвичай включає додавання більш дрібних етапів до основних дій. Це допомагає розкрити кожну дію на менші складові частини, розширюючи розуміння процесу.

Крім того, уточнення умов та можливих виборів стає частиною деталізації. Це означає врахування альтернативних шляхів у залежності від умов, що дозволяє більш точно відобразити потік подій.

Додатково, використання відміток для позначення ключових моментів у процесі дозволяє виділити важливі точки, що вимагають спеціальної уваги або обробки.

Опис взаємодії з іншими системами або користувачами, а також додавання пояснень для складних чи важливих аспектів процесу, робить діаграму більш повною та зрозумілою для всіх зацікавлених сторін.

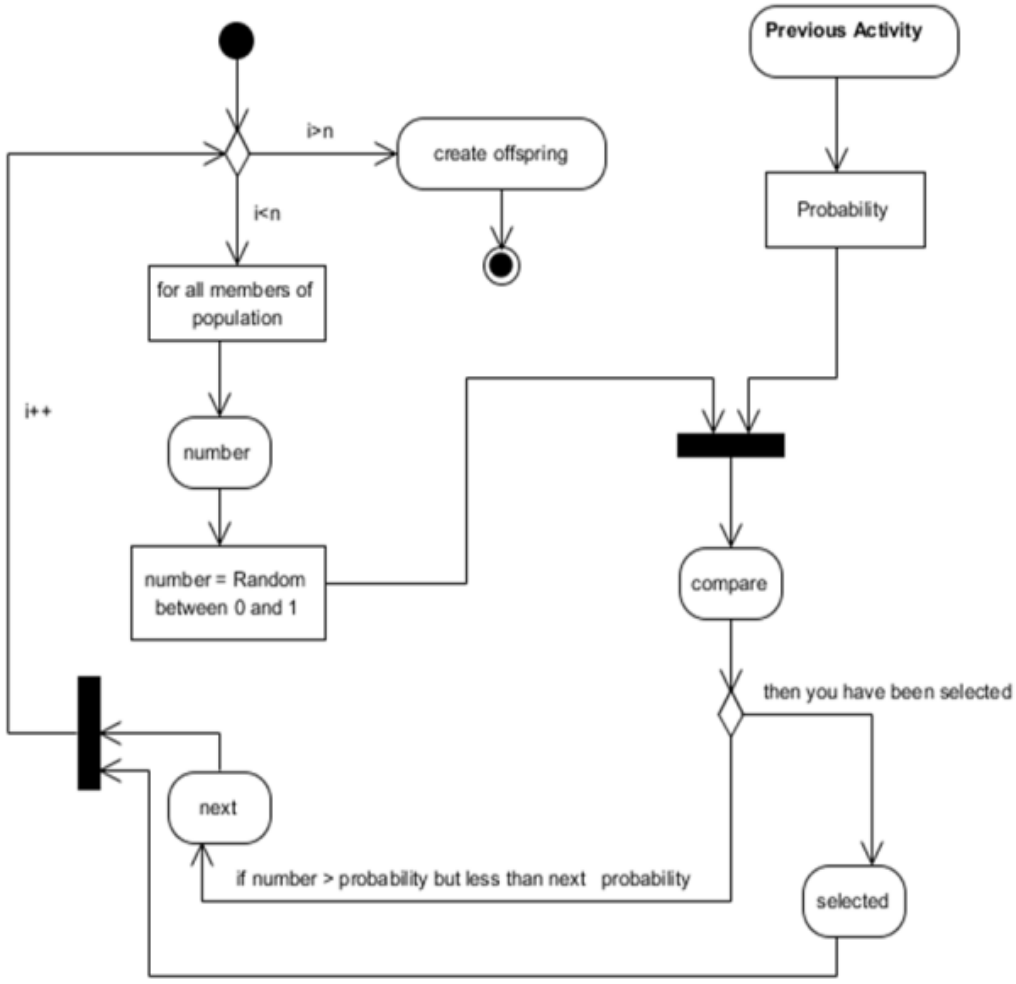


Рисунок 2.6 – Деталізована діаграма діяльності відбору учасників

Діаграма послідовності (рис. 2.7) представляє так званий оператор перетину GSX і спосіб його роботи. Ця діаграма представляє взаємну взаємодію між об’єктами (батьківським, дочірнім, методами GSX), яка загалом представляє серію обмінів повідомленнями між класами, з чітко позначеною послідовністю та часовим ходом надсилання та отримання повідомлень.

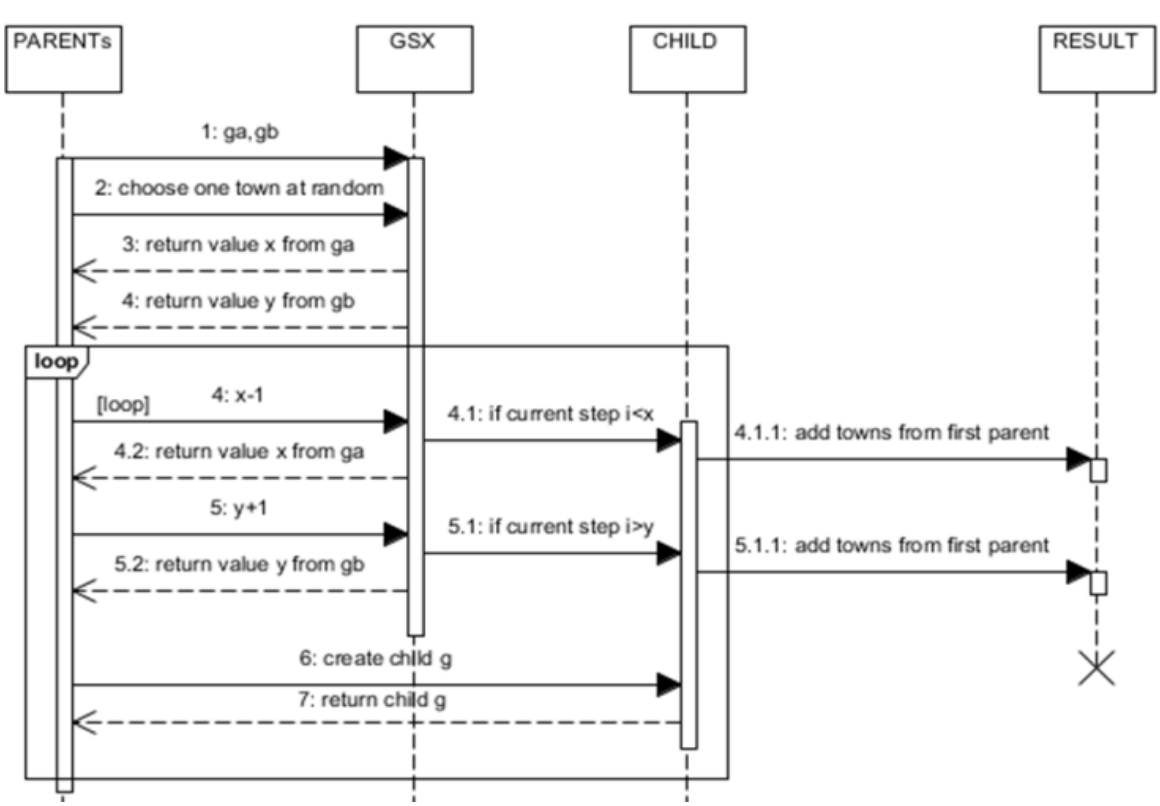


Рисунок 2.7 –Діаграма послідовності процесу оптимізації маршруту комівояжера

Діаграми послідовності у моделюванні програмного забезпечення створюють візуальне уявлення про взаємодію об'єктів системи в часі. Вони показують послідовність повідомлень, що передаються між об'єктами для виконання певного функціоналу або сценарію.

Ці діаграми малюють послідовність взаємодії між різними об'єктами у системі, відображаючи, як об'єкти взаємодіють між собою через повідомлення. Вони вказують порядок, в якому об'єкти відправляють та отримують повідомлення, зрозумілі умови та взаємні впливи між ними.

Кожна діаграма послідовності містить об'єкти, які взаємодіють між собою,

стрілки (повідомлення), що показують напрямок передачі інформації, та кілька стереотипів, що вказують типи взаємодії між об'єктами (наприклад, стереотипи "виклик" або "відповідь").

Ці діаграми є корисним інструментом для уточнення поведінки системи, опису сценаріїв використання та розробки програмного забезпечення, оскільки вони дають зрозуміле уявлення про взаємодію об'єктів у системі в часі.

Процедура кросоверу в генетичному алгоритмі для задачі комівояжера використовується для створення нових потенційних розв'язків шляху комівояжера через поєднання частин шляхів батьківських розв'язків.

У процедурі кросоверу два батьківські розв'язки вибираються з популяції, а потім випадковим чином обирається точка перетину (точка, де шлях розбивається на частини для обміну). Частини шляхів від кожного батьківського розв'язку обмінюються, утворюючи новий розв'язок.

Наприклад, якщо уявити, що маємо два шляхи (розв'язки) для комівояжера:

1. ABCDEFGH
2. IHGFEDCBA

Та вибрана точка перетину - наприклад, між "D" та "E". Тоді можемо виконати кросовер, обмінявши частини шляхів:

1. ABCDEDCBA
2. IHGFEDFGH

Потім, можливо, виконаємо операції зміни порядку міста для унікальності шляху (наприклад, перетворення "EDCBA" в "ABCDE") та виправлення дублювання міста, щоб отримати прийнятний розв'язок.

На рисунках 2.8 і 2.9 представлені дії, які здійснюються в рамках діяльності «схрещування», і описують оператора GSX з іншого боку. На попередній діаграмі послідовності чітко вказано порядок і взаємодію між об’єктами, які беруть участь у цих діях, і ці діаграми показують, що акцент робиться на діях та їх виконанні та на існуючих умовах.

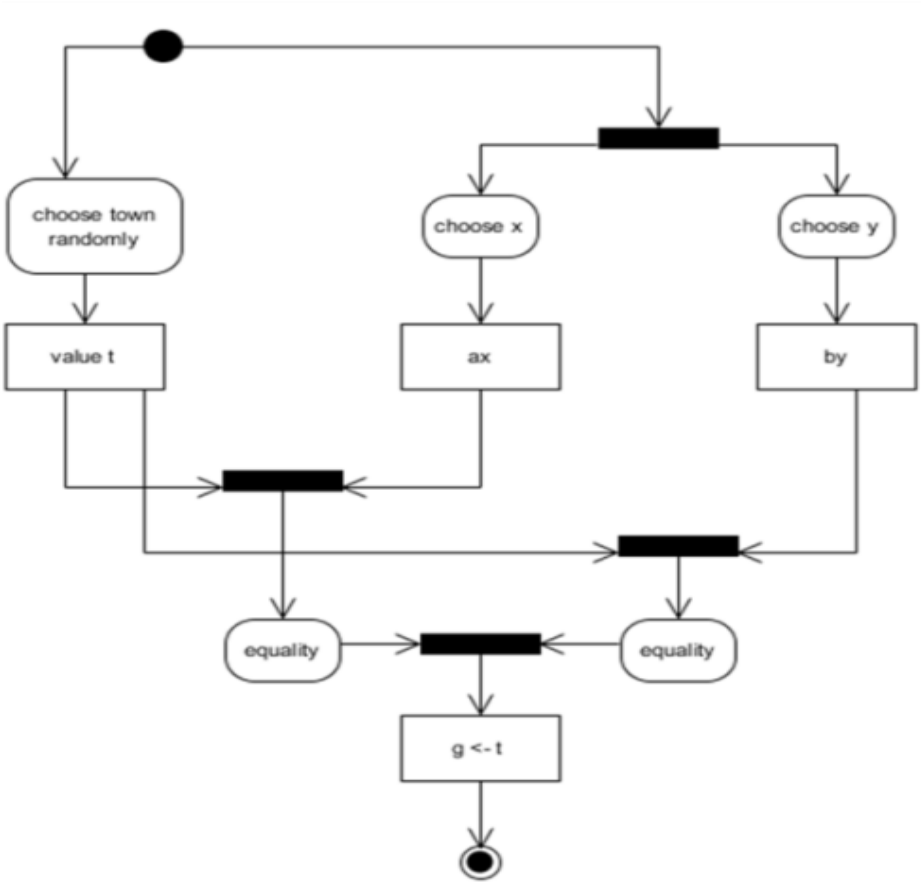


Рисунок 2.8 – Діаграма діяльності для процедури кросоверу

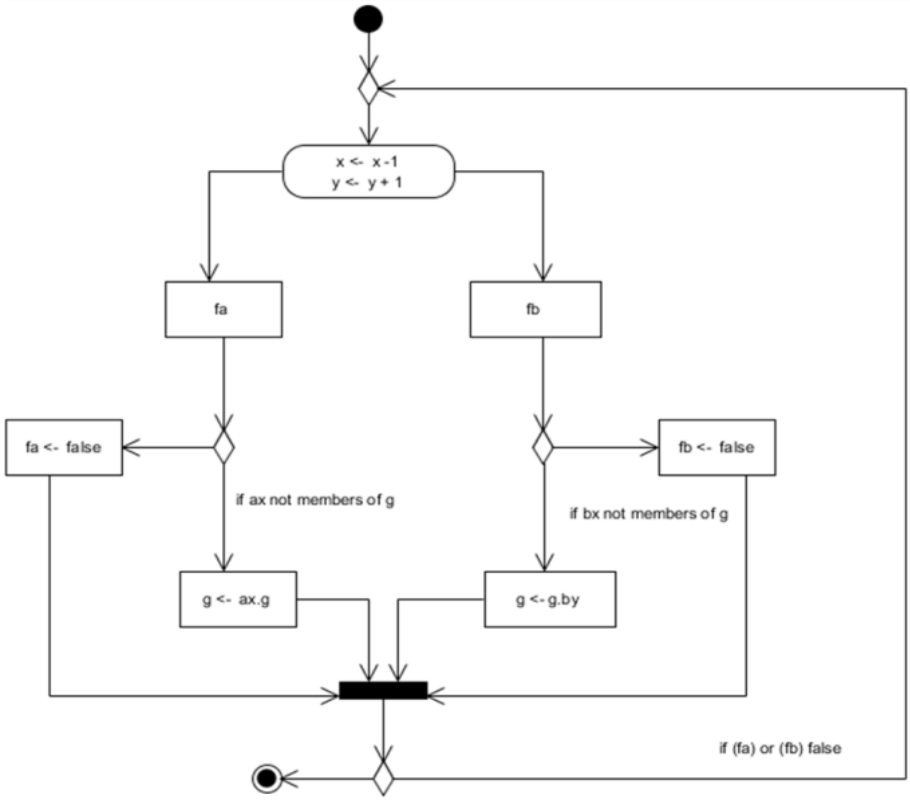


Рисунок 2.9 – Діаграма дільності для кросоверу за жадібним алгоритмом

Це створює новий розв'язок, який може бути випробуваний на ефективність (за допомогою функції оцінки) та може стати членом нової популяції для наступної ітерації генетичного алгоритму.

Процедура кросоверу за жадібним алгоритмом в генетичних алгоритмах може відрізнятися в залежності від застосованої стратегії. Однак, основна ідея полягає в тому, що при кросовері використовуються жадібні правила вибору частин рішень для об'єднання.

У жадібному кросовері вибір частин шляхів (розв'язків) для об'єднання базується на певних жадібних критеріях. Ці критерії можуть бути засновані на локальних оптимальних рішеннях або специфічних евристиках для задачі комівояжера.

Наприклад, одним з можливих жадібних підходів для кросоверу в задачі комівояжера може бути вибір кращих частин шляхів (наприклад, підрядків міст), основуючись на їхній ефективності або оптимальності з точки зору довжини шляху.

Для кросоверу вибираються певні фрагменти шляхів з батьківських розв'язків на основі цих жадібних критеріїв. Потім об'єднують ці фрагменти для створення нового розв'язку.

Мутація в генетичних алгоритмах використовується для інтродукції різноманітності в популяцію шляхів комівояжера шляхом випадкових змін у існуючих розв'язках.

В діаграмі діяльності мутація може бути відображена так:

1. Початок процесу мутації. Стартова точка, в якій розпочинається операція мутації.

2. Вибір розв'язку для мутації. Відбір конкретного розв'язку або шляху для внесення змін.

3. Випадкові зміни. Внесення випадкових змін в обраний розв'язок, таких як зміна порядку міст або обмін двох місць.

4. Оцінка нового розв'язку. Після мутації розв'язку відбувається оцінка отриманого нового шляху, можливо за допомогою функції оцінки, що визначає

його ефективність.

5. Прийняття чи відхилення нового розв'язку. Залежно від оцінки, може прийматися рішення щодо збереження або відкидання нового шляху.

6. Завершення процесу мутації. Завершення операції мутації для даного розв'язку.

Мутація в генетичних алгоритмах допомагає зберегти різноманіття в популяції та уникнути передчасної збіжності до локальних оптимумів, дозволяючи алгоритму розвиватися та досліджувати нові можливі шляхи рішень (рис. 2.10).

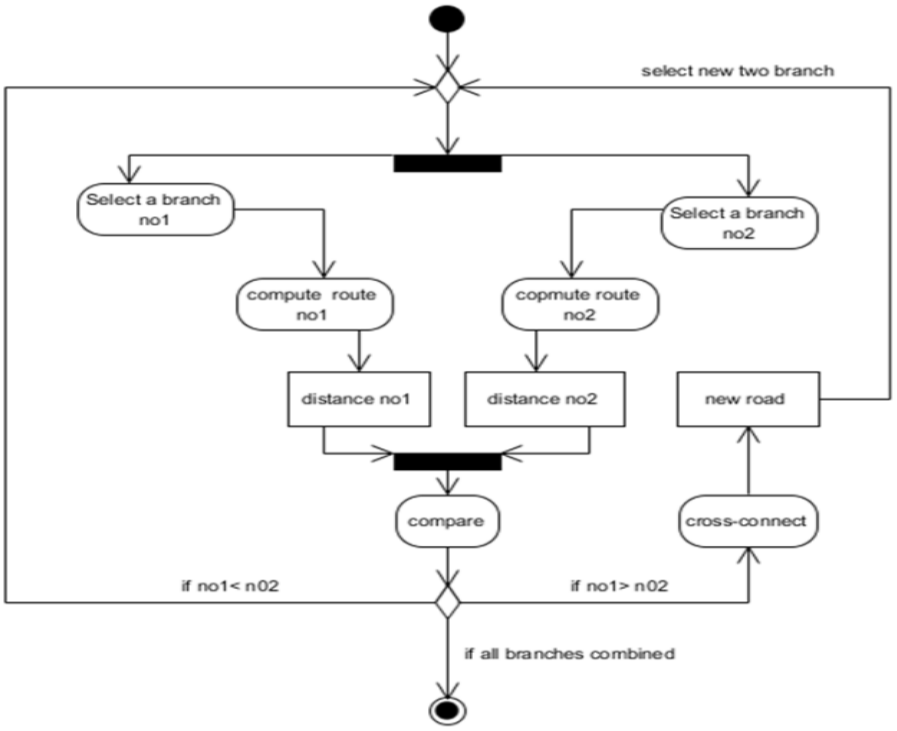


Рисунок 2.10 – Діаграма діяльності мутації

2.3 Розробка структури інформаційної технології розвязання задачі комівояжера з застосуванням генетичного алгоритму

Керівник, після того, як здійснив рандомізацію початкової популяції, виконує сортування за рівнем придатності. Потім, видаляє задану кількість найгірших хромосом, враховуючи рівень відбору, до тих пір, поки не буде можливо провести їх заміну новими нащадками. Перед розподілом, кількість хромосом, що

буде замінена новими нащадками, визначається вибраною функцією. Потім, шляхом розподілу на кількість процесорів, обчислюється заміна для кожного з робочих елементів. Цей параметр разом з початковим індексом параметрів масиву передається робочим завданням.

Найсуттєвіша частина нашої роботи вже зроблена. Робочі елементи повинні працювати самостійно, без потреби опорних ниток. Процес генерації виконаний повністю незалежно. Коли батьки розділяються між робочими елементами та генерують нащадків, це підвищує незалежність наших процесів. Вставка отриманих нащадків у глобальний масив популяції (розташований у головному) є єдиною стадією, яка вимагає послідовності. Однак цей масив завжди можна зберегти як відсортований набір у будь-який момент. Загалом, алгоритм сортування в менеджері не потрібен.

Структура інформаційної технології розвязання задачі комівояжера з застосуванням генетичного алгоритму представлена на рисунку 2.11.

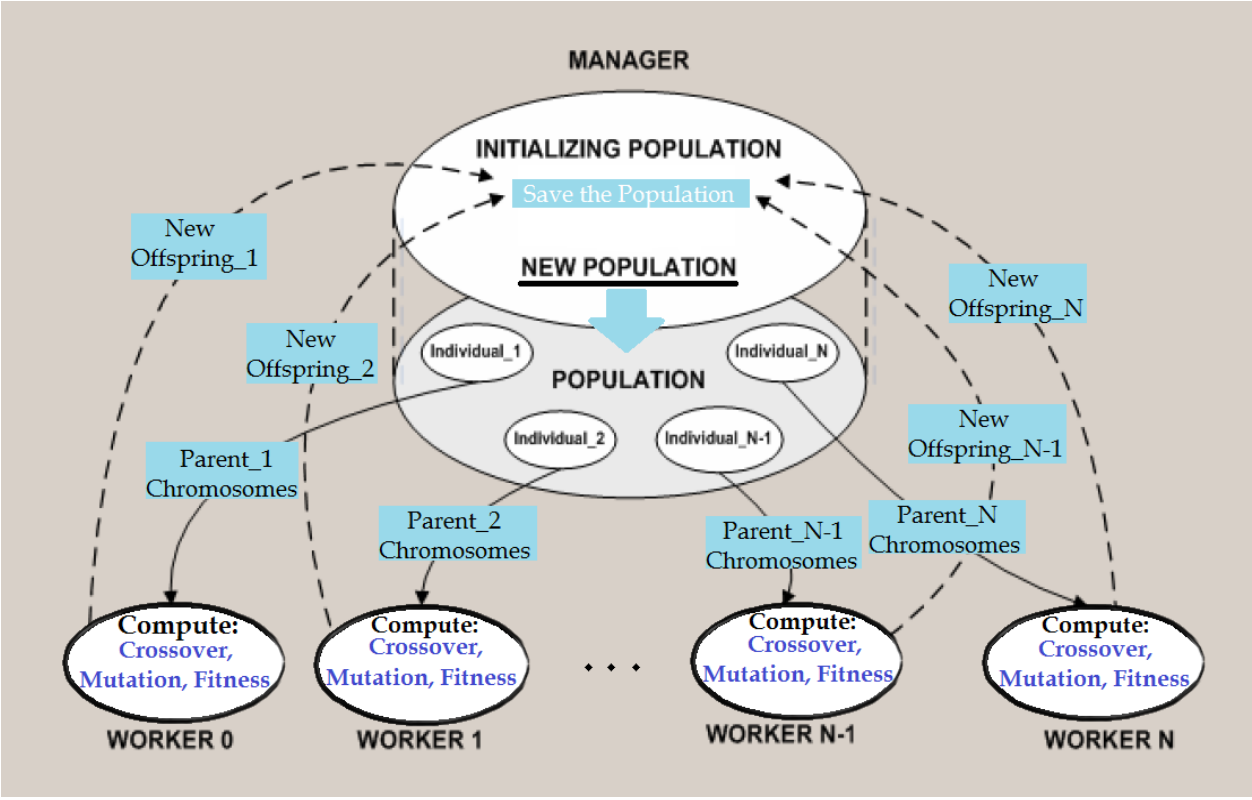


Рисунок 2.11 – Структура інформаційної технології розвязання задачі комівояжера з застосуванням генетичного алгоритму

Для розробки паралельного алгоритму була використана функціональна декомпозиція паралельної програми, відповідно до парадигми "менеджер/працівники". Оцінка продуктивності та профілювання паралелізму були проведені на основі реалізації багатозадачної програми. Експериментальна паралельна комп'ютерна платформа — це комп'ютер з кількома процесорами, що включає вісім робочих станцій, що взаємодіють через батьківське завдання або мережевий комутатор.

Керуючий процес (ранг 0) виконує всі генетичні операції для першої популяції та розподіляє обчислювальне навантаження між робочими процесами. Він виконує такі види діяльності:

- Ініціалізація популяції (рандомізація);

- Сортуванняи хромосоми за значенням відповідності (хромосома з найнижчим значенням розміщується на першому місці масиву);

- Перевірка елітності хромосом і вибір найгіршої хромосом для видалення;

- Визначення ймовірності відбору відповідно до відповідності хромосом;

- Вироблення нового покоління будівельниками до кількості ядер для виконання генетичних операторів;

- Отримання оціненого потомства від працівників і зберігання нових хромосом за формулою, проіндексованою в масиві;

- Створення нової популяції, поєднуючи будь-яке отримане потомство від робочих у масиві;

- Перевірка умовного завершення для розрізання циклу;

- Друк обчисленого найкоротшого шляху.

Операції робочого процесу такі:

- Виконання вибору колеса рулетки;

- Виконання рекомбінації (вибирається 2 точки для кросовера);

- Виконання звичайної випадкової мутації (вибираються та обмінюються два міста);

- Оцінювання придатності нової хромосоми.

2.4 Висновок до розділу 2

Під час розробки інформаційної технології для розв'язання задачі комівояжера з використанням генетичного алгоритму було побудовано UML-діаграми, що відображають структуру та взаємодію компонентів системи. Структурні схеми застосування генетичного алгоритму в задачі комівояжера чітко відображають процес оптимізації маршруту та взаємодію між окремими етапами алгоритму. Ці діаграми і схеми стали основою для подальшої програмної реалізації алгоритму та його належного тестування, забезпечуючи зручний аналіз та розуміння роботи системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА ІЗ ЗАСТОСУВАННЯМ ГЕНЕТИЧНОГО АЛГОРИТМУ

3.1 Обгрунтування вибору мови програмування і середовища розробки

Розглянемо декілька мов програмування для програмної реалізації інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму.

Java – це об'єктно-орієнтована мова програмування, розроблена Sun Microsystems (пізніше придбана Oracle). Написані на Java програми переважно компілюються у спеціальний байт-код, що забезпечує можливість роботи на будь-якій віртуальній машині Java (JVM), при цьому не залежачи від архітектури комп'ютера. Перевагою цього методу компіляції програм є повна незалежність байт-коду від операційної системи та обладнання, що дозволяє запускати програми Java на кожному пристрої, що має відповідну віртуальну машину [18].

Ще однією впливовою характеристикою технології Java є гнучка система безпеки, чому сприяло виконання програми повністю контрольоване віртуальною машиною. Усі операції, що перевищують встановлені повноваження програми, негайно переривають роботу. Ще однією важливою характеристикою є те, що для Java притаманне використання автоматичного збирача сміття для управління пам'яттю протягом життєвого циклу об'єкта. Розробник вирішує, коли створювати об'єкти, а віртуальна машина відповідає за звільнення пам'яті після того, як об'єкт більше не потрібен. Якщо більше немає посилань на конкретний об'єкт, збирач сміття може автоматично видалити його з пам'яті. Вивіз сміття дозволений у будь-який час. В ідеалі це відбувається, коли програма неактивна. Java не підтримує вказівники на стиль C / C ++. Це робиться з міркувань безпеки та надійності, щоб збирач сміття міг переміщати об'єкти вказівника.

Джеймс Гослінг почав розробляти проект мови програмування JAVA в

липні 1991 року, для використання його в одному із своїх багаточисельних проектів set – top box. Мова спочатку називалась Oak “Дуб” на честь дуба, який ріс перед офісом Гослінга, але в кінці вибір був зупинений на JAVA, назва була вибрана із списку випадковим чином.

Гослінг вирішив запропонувати як додаток до віртуальної машини, яка буде мати стиль C C++. Sun випустила свій продукт під іменем JAVA 1.0. Девіз звучав (пишеш один раз запускаєш всюди). Слід відзначити і налаштовану безпеку, яка дозволяє використовувати межу на рівні файлів доступа.

Більшість веб–браузерів володіли можливістю запускати JAVA– аплети (програмні компонети в двоїчному коді які виконуються в вікні браузера) на веб–сторінках. Завдяки всьому цьому JAVA в дуже короткий час стала дуже популярною мовою.

В грудні 1998 року появилась JAVA 2. Нова версія пропонувала більшість конфігурацій, створених спеціально для різних типів платформ, наприклад JDEE додаток типу enterprise в той же час як stripped+down Sun ME був придуманий для мобільних платформ.

Sun переймінувала нові версії JD2 наступним чином: JAVA EE; JAVA ME; JAVA SE;

JAVA залишається стандартом, яким керується JAVA Community Process (процес який дозволяє заінтересованим лицям приймати участь у формуванні майбутніх специфікацій JAVA). Sun запропонувала більшість частин JAVA безкоштовно, незалежно на статус власника програмного забезпечення. Доходи від JAVA поступають за рахунок продажі спецалізованих продуктів, таких як Enterprise Java System.

Java - це мова програмування загального призначення, яка дотримується парадигми об’єктно-орієнтованого програмування та підходу одноразового запису й використання всюди. Java використовується в настільних, веб-, мобільних і корпоративних додатках. Java не настільки зручна для розробників, як Python, але вона досить проста для будь-якого розробника з базовим розумінням фреймворків, пакетів, класів та об’єктів. Він простий, стандартизований і передбачуваний, що

дозволяє навчитися мислити в правильному напрямку [14].

Мова програмування C# представляє собою об'єктно-орієнтовану мову з безпечною системою типізації, спеціально призначену для використання на платформі .NET. Її розробка відбувалася в рамках дослідницького підрозділу Microsoft Research, зусиллями видатних фахівців, таких як Андерс Гейлсберг, Скот Вілтамут і Пітер Гольде.

C# створювалася з метою надання мови програмування високого рівня для використання на платформі CLR (Common Language Runtime). Завдяки цьому, мова тісно взаємодіє з функціоналом CLR, що визначає систему типізації та інші характеристики C#. У розвитку мови протягом версій від 1.1 до 2.0 спостерігалося значне збагачення її можливостей, обумовлене саме розвитком CLR. Зазначимо, що версія C# 3.0 порушила цю закономірність, додаючи розширення мови, які не базуються на розширеннях платформи .NET.

Однією з основних переваг C# є використання CLR, яка надає мові та іншим .NET-орієнтованим мовам ряд можливостей, недоступних для "класичних" мов програмування. Наприклад, збірка сміття в C# не реалізована безпосередньо в мові, а здійснюється CLR для програм, написаних на C# [13].

Принципові рішення, впроваджені у мові програмування C#, визначають її основні характеристики та функціональні можливості:

- компонентно-орієнтований підхід до програмування. Цей принцип базується на ідеї розробки програм за допомогою компонентів, які можна знову використовувати. У мові C#, це реалізується через використання поняття класів та об'єктно-орієнтованого програмування (ООП);

- властивості як засіб інкапсуляції даних. Властивості дозволяють забезпечити контроль доступу до полів класів та забезпечити шлях для читання та запису їх значень, зберігаючи при цьому інкапсуляцію даних;

- обробка подій. Мова C# надає розширені засоби для роботи з подіями, включаючи обробку винятків за допомогою оператора try;

- делегати у C# дозволяють вказувати на функції або методи як параметри, реалізуючи тим самим покажчик на функцію в мовах C і C++.

- індексатори дозволяють звертатися до елементів класу-контейнера за допомогою операторів індексу, схожих на роботу з масивами;

- перевантажені оператори. В C# можна перевантажувати оператори, щоб визначити власне значення для використання їх з екземплярами класів;

- оператор foreach. Цей оператор дозволяє легко перебирати всі елементи класів-колекцій, спрощуючи обробку даних;

- механізми boxing і unboxing. Boxing дозволяє конвертувати значимі типи даних в типи-об'єкти, а unboxing – навпаки;

- атрибути використовуються для додавання метаданих до коду, що спрощує взаємодію з COM-моделлю та іншими аспектами програмування;

- прямокутні масиви визначаються як набір елементів з доступом за номером індексу, і вони мають однакову кількість стовпців і рядків.

Ці основні рішення формують основу C# як сучасної та потужної мови програмування для платформи .NET.

Microsoft Visual Studio - це комплексний набір інструментів для розробки програмного забезпечення, розроблений фірмою Microsoft. Основною метою Visual Studio є надання інтегрованого середовища розробки (IDE) та ряду інших засобів для зручного створення, тестування та вдосконалення програм. Детальніше про деякі з ключових характеристик Visual Studio:

- інтегроване середовище розробки (IDE). Visual Studio надає потужне та зручне IDE, яке об'єднує в собі редактор коду, відладчик, дизайнер інтерфейсів, інструменти для збірки та управління версіями;

- підтримка різноманітних технологій. Visual Studio дозволяє розробляти різноманітні типи програм, включаючи консольні програми, застосунки з графічним інтерфейсом (включаючи Windows Forms), веб-сайти, веб-застосунки та служби, як в рідному, так і в керованому кодах;

- підтримка платформ Microsoft. Visual Studio орієнтований на розробку для різних платформ Microsoft, таких як Microsoft Windows, Windows Mobile, Windows Phone, Windows CE. Він також підтримує різні версії .NET Framework, .NET Compact Framework і Microsoft Silverlight;

- мови програмування. Visual Studio підтримує різні мови програмування, включаючи C#, Visual Basic, C++, F# та інші. Кожна мова має своє відповідне середовище та інструменти;

- тестування та відлагодження. Інтегровані інструменти для автоматичного та ручного тестування, а також відлагодження дозволяють розробникам створювати високоякісне програмне забезпечення;

- спільна робота та керування кодом. Visual Studio забезпечує інструменти для спільної роботи команд розробників, а також можливості керування версіями коду за допомогою системи контролю версій;

- розширення та розширюваність. Велика кількість додаткових компонентів, шаблонів та розширень дозволяє розробникам розширювати можливості Visual Studio та пристосовувати її до своїх потреб.

Visual Studio є ключовим інструментом для розробки на платформі Microsoft, і його функціональність охоплює весь життєвий цикл розробки програмного забезпечення.

Visual Studio – це невід'ємна частина робочого процесу для багатьох розробників, які спеціалізуються на платформі Microsoft. Цей інтегрований розробницький середовище (IDE) надає широкий спектр інструментів і можливостей, які допомагають спростити і поліпшити кожен етап створення програмного забезпечення.

Починаючи з створення нового проекту, Visual Studio пропонує різноманітні шаблони, які включають в себе різні типи додатків, від класичних консольних програм до складних веб-додатків і мобільних застосунків. Інтуїтивний інтерфейс сприяє швидкому старту, а інтегровані засоби автоматизації дозволяють зосередитися на креативному процесі розробки.

Під час активного кодування розробники можуть скористатися багатьма корисними функціями, такими як автоматичне доповнення коду, перевірка синтаксису, а також вбудовані інструменти для відлагодження та профілювання програми. Завдяки інтеграції з системами контролю версій, розробники можуть ефективно керувати версіями свого коду та спілкуватися з іншими членами

команди.

Інструменти Visual Studio відображають сучасні вимоги розробки програмного забезпечення, допомагаючи не лише в створенні традиційних додатків для мобільних телефонів і комп'ютерів, але й у розробці хмарних застосунків. Процеси тестування, відлагодження і розгортання програм в хмарі аналогічні процесам створення .NET-застосунків. Важливою новацією в Visual Studio є інструменти для багатопоточної розробки, які охоплюють як некерований, так і .NET Framework.

Visual Studio виводить на новий рівень інтерфейс, використовуючи Windows Presentation Foundation (WPF). Уведено наступне покоління інструментів ASP.NET, забезпечено підтримку динамічних розширень в мовах програмування C# і Visual Basic. Також представлені нові шаблони проектів, інструментарій для документування тестових сценаріїв та значна кількість нових бібліотек, спрямованих на підтримку функцій Windows 7.

Visual Studio Ultimate, що формально відома як Visual Studio Team System з кодовою назвою Rosario, представляє собою новий інструмент для спільної розробки застосунків. Інтегроване середовище розробки (IDE - Integrated Development Environment) Visual Studio є багатовіконним та володіє широким спектром можливостей. Усередині його можна виділити три основні вікна:

- вікно «Обозревателя решений» (Solution Explorer). В цьому вікні представлена структура побудованого рішення, що дозволяє оглядати та керувати різними елементами проекту;

- вікно властивостей (Properties). У вікні Свойства можна переглядати та редагувати властивості вибраного елемента рішення. Це дозволяє змінювати параметри та налаштування обраних об'єктів у проекті;

- вікно документів. В цьому вікні відображається вміст вибраного документа або файлу. Це вікно також може відображати інші документи, а список доступних документів показується у верхній частині вікна.

Visual Studio Ultimate надає зручний та ефективний інтерфейс для розробки та спільної роботи над проектами.

Порівняльна характеристика цих мов подана у таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Назва	Опис	Переваги	Недоліки
C#	Розробка ОС та офісних рішень для платформи Microsoft, розробка веб-додатків, настільних графічних додатків (використовуються платформи Windows Forms и WPF), серверних додатків в сфері створення динамічного веб-контенту за допомогою платформи ASP.NET, мобільних додатків для операційної системи Windows Phone, що розроблена компанією Microsoft.	Висока швидкість розробки в малобюджетних проектах Простота і лаконічність коду Наявність власного середовища розробки Можливості спадкування й універсалізації Типи, вбудовані в мову, представлені класами Збереження і об’єднання кращих рис мов C і C++ Підвищення ефективності коду, оскільки виконавче середовище CLR являє собою компілятор проміжної мови Зручність побудови різних типів додатків дозволяє легко розробляти Web-служби	Проблеми при розробці додатків під не-Windows платформи Сильна прив’язка до Microsoft Нема самодостатньості додатків, так як використовується .net framework Відсутність повноцінних деструкторів, повноцінних макросів, дуже обмежена підтримка шаблонів

Продовження таблиці 3.1

C++	Створення ОС, прикладних програм, драйверів приладів, додатків для вбудованих систем, високопродуктивних серверів, а також програмування ігор.	Висока сумісність з мовою C Підтримка різних стилів програмування Можливість побудови загальних контейнерів і алгоритмів для різних типів даних Кросплатформенність Доступність і популярність	Погано продуманий синтаксис робить мову незручною в деяких задачах Нема самодостатньості додатків, для цього використовується runtime Продуктивність праці програмістів на мові виявляється невиправдано низькою, а продукт праці є низькоякісним
Java	Створення десктопів і аплетів, мобільних додатків, серверних додатків, що орієнтовані на роботу з мережею, програмування ігор.	Універсальність і широта застосування Кросплатформенність Відкритий вихідний код і велика кількість бібліотек Гнучка система безпеки Висока продуктивність Багатозадачність	Мова не має таких властивостей об'єктно-орієнтованої мови, як індивідуальні змінні та множинне спадкування Відсутність спливаючих підказок до методів, які можна примінити до певного об’єкта

Продовдення таблиці 3.1

Delphi	Об'єктно-орієнтована мова програмування, нащадок Pascal, більш відома як основна мова програмування середовища Delphi.	Підтримка багатьох компіляторів Зручність в створенні експертних систем Якісний графічний інтерфейс Простота	Консервативність

Із усіх розглянутих мов програмування, було обрано C# як найбільш оптимальну для розробки додатку. Вибір середовища розробки є критичним на етапі створення програми. Одним із найпопулярніших інструментів для розробки є Visual Studio від Microsoft – інтегроване середовище розробки (IDE), яке надає розробникам потужні інструменти для програмування, тестування та налагодження програм.

Переваги Visual Studio порівняно з іншими середовищами розробки:

1. Широкий функціонал. Visual Studio має значний набір інструментів, таких як автодоповнення коду, відладчик, система контролю версій тощо. Ці функції полегшують розробникам роботу з кодом.

2. Підтримка мов програмування. Від Visual Studio підтримуються багато мов, включаючи C#, C++, Visual Basic і інші. Це дає можливість розробникам вибирати мову, яка найкраще відповідає їх проекту.

3. Інтеграція з продуктами Microsoft. Visual Studio добре поєднується з іншими продуктами Microsoft, такими як Microsoft Azure, Microsoft SQL Server і Microsoft Office. Це дозволяє легко створювати програми, які взаємодіють з іншими продуктами Microsoft.

4. Спільнота розробників. У Visual Studio дуже активна спільнота розробників, які допомагають один одному у вирішенні проблем та відповіді на питання. Це дозволяє отримувати швидку підтримку та знаходити відповіді на більшість запитань.

3.2 Створення інтерфейсу користувача інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму

Спершу опишемо бібліотеки, які були застосовані при розробці інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму.

System надає доступ до базових класів і типів .NET Framework, необхідних для будь-якого додатку. Collections.Generic містить узагальнені колекції, такі як список, словник, чергу. З їх допомогою можна зберігати і обробляти дані програми. Diagnostics включає класи для логування та відстеження роботи програми, що важливо при налагодженні та тестуванні. Drawing надає базову підтримку малювання та графіки. З її допомогою створюються візуальні елементи інтерфейсу. IO відповідає за роботу з файлами, потоками, серіалізацією даних. Linq містить засоби LINQ для зручної роботи з даними у коді. Threading використовується для організації багатопоточності в додатку, що покращує продуктивність та асинхронність. Tasks дозволяє писати асинхронний код на основі завдань. Windows.Forms – GUI фреймворк для створення графічного інтерфейсу додатка. VisualBasic.PowerPacks додає додаткові GUI елементи. TSP.Core містить базову функціональність самої програми. TSP.TimerGraphs відповідає за побудову графіків та таймерів. ZedGraph – стороння бібліотека для побудови графіків.

Головна форма MainForm відповідає за інтерфейс користувача. Вона містить методи для ініціалізації форми, створення об'єктів інших класів, обробки подій інтерфейсу. Також визначає властивості для кількості популяції, ядер процесора, контейнери для малювання міст і шляхів.

Для представлення даних в часі використовуються списки точок PointPairList – для побудови графіків залежності часу від пристосованості і часу від поколінь.

Реалізована можливість запуску алгоритму в потоці Thread. Використовується статичний метод UiInvoke для безпечного оновлення інтерфейсу з іншого потоку.

Клас GeneticAlgorithm містить реалізацію самого генетичного алгоритму – ініціалізації, селекції, схрещування, мутації.

Загалом структура програми дозволяє гнучко налаштовувати параметри алгоритму, відстежувати хід роботи за допомогою графіків, масштабувати на багатоядерні процесори.

Ця частина коду відповідає за асинхронне оновлення елементів інтерфейсу користувача з іншого потоку, в якому виконується генетичний алгоритм.

Використовуються делегати (SetValueCallback, SetMaxValueCallback та інші), щоб безпечно викликати методи елементів управління (прогрес-бар, лейбли) з стороннього потоку за допомогою Invoke().

Реалізовані методи SetValue, SetMaxValue для оновлення прогрес-бару, SetGenerationText, SetLenghtText для оновлення тексту лейблів.

Також є методи для додавання/видалення графічних елементів маршруту (AddLineShape, RemoveLineShape) та оновлення їх координат (SetPoint).

Метод SetNumPopEnable вмикає/вимикає елемент введення кількості популяції.

Отже, ця частина відповідає за коректну роботу інтерфейсу під час асинхронного виконання алгоритму в окремому потоці. Все оновлення інтерфейсу відбувається безпечно через Invoke().

Наступна частина містить реалізацію генетичного алгоритму для задачі комівояжера.

Спочатку визначаються допоміжні змінні - генератор випадкових чисел, змінна для збереження найкращого рішення.

Викликається метод SetCitiesPosition для розміщення місць на формі.

Обчислюється кількість ядер процесора для паралельних обчислень.

Ініціалізується клас GeneticAlgorithm, що містить основну логіку алгоритму.

Далі йде цикл, в якому на кожній ітерації виконуються:

1. Сортування популяції за пристосованістю (fitness) – Bubble Sort
2. Елітизм – перевірка чи поточне рішення краще за найкраще знайдене.

3. Селекція – вибір хромосом для схрещування.

4. Схрещування обраних батьківських хромосом з певною ймовірністю.

5. Мутація деяких генів/міст хромосом з певною ймовірністю.

6. Оновлення найкращого рішення, якщо знайдено краще.

7. Оновлення інтерфейсу користувача.

Таким чином, реалізується класичний цикл генетичного алгоритму для оптимізації шляху комівояжера.

Спочатку для кожної хромосоми обчислюється ймовірність вибору для схрещування відповідно до пристосованості - Rank_Trim(). Кращі хромосоми мають більшу ймовірність.

Розглянемо наступну частину коду:

```
#region Reproduction
// Definition Probability According by chromosome fitness
// create Pn[N_keep]; Rank_Trim();
if (pGAToolStripMenuItem.Checked) // Parallel Genetic Algorithm
{
MultiThreading
Task Parallelism
Parallel.For ...
}
if (threadParallelismToolStripMenuItem.Checked) // PGA by
{
ReproduceByParallelThreads();
}
else if (taskParallelismToolStripMenuItem.Checked) // PGA by
{
ReproduceByParallelTask();
}
else if (parallelForToolStripMenuItem.Checked) // PGA by
```

```
{
PReproduction(rand);
}
else // Series Genetic Algorithm
{
#region Series Reproduct Code
Reproduction(rand); #endregion
}
#endregion
count++;
SetGenerationText(count.ToString());
//lblGeneration.Text = count.ToString();
//
SetValue(toolStripProgressBar1.Value + 1);
//toolStripProgressBar1.Value++;
//
}
while (count < toolStripProgressBar1.Maximum && Isotropy_Evaluatuon());
//
//toolStripProgressBar1.Value = toolStripProgressBar1.Maximum;
SetValue(toolStripProgressBar1.Maximum);
//
// UnLock numUpDownPop SetNumPopEnable(true);
//
// The END Stop();
}
#region Generation Tools
//find percent of All chromosome rate for delete Amiss(xRate) or
Useful(Nkeep) chromosome
//x_Rate According by chromosome fitness Average private void x_Rate()
```

```
{
// calculate Addition of all fitness double sumFitness = 0;
for (var i = 0; i < PopulationNumber; i++)
sumFitness += Genetic.Population[i].Fitness;
// calculate Average of All chromosome fitness
var aveFitness = sumFitness / PopulationNumber; //Average of all chromosome
fitness
_nKeep = 0; // N_keep start at 0 till Average fitness chromosome for (var i = 0; i
< PopulationNumber; i++)
if (aveFitness >= Genetic.Population[i].Fitness)
{
_nKeep++; // counter as 0 ~ fitness Average + 1
}
if (_nKeep <= 0) _nKeep = 2;
}
// Definition Probability According by chromosome fitness private void
Rank_Trim()
{
// First Reserve Possibility Number for every Remnant chromosome
// chromosome Possibility Function is:
// (1 + N_keep - No.chromosome) / ( ∑ No.chromosome)
// Where as at this program No.chromosome Of Array begin as Number 0
// There for No.chromosome in Formula = No.chromosome + 1
// then function is: if (n == N_keep)
N_keep
// Possibility[No.chromosome] = (n - No.chromosome) / (n(n+1) / 2)
//
_pn = new double[_nKeep]; // Create chromosome possibility Array Cell as
double sum = ((_nKeep * (_nKeep + 1)) / 2); // (∑ No.chromosome) ==
(n(n+1) / 2)
```

pn[0] = nKeep / sum; // Father (Best - Elite) chromosome Possibility for (var i = 1; i < _nKeep; i++)

{

+2] = 0.1)

}

// Example: if (Pn[Elite] = 0.4 & Pn[Elite +1] = 0.2 & Pn[Elite

// Then Own: 0 <= R <= 0.4 ===> Select chromosome[Elite]

// 0.4 < R <= 0.6 ===> Select chromosome[Elite +1]

// 0.6 < R <= 0.7 ===> Select chromosome[Elite +2]

// etc ...

_pn[i] = ((_nKeep - i) / sum) + _pn[i - 1];

}

Ця частина коду відповідає за репродукцію (схрещування і мутацію) хромосом в генетичному алгоритмі.

Далі виконується сам процес схрещування в залежності від режиму:

1. Послідовний алгоритм (Series) – викликається метод Reproduction()
2. Паралельний за допомогою багатопоточності (Threads) – ReproduceByParallelThreads()
3. Паралельний за допомогою завдань (Tasks) – ReproduceByParallelTask()
4. Паралельний за допомогою Parallel.For – PReproduction()

В кожному випадку виконується схрещування батьківських хромосом з певною ймовірністю crossrate, після чого виконується мутація генів/міст з ймовірністю mutrate.

Таким чином, реалізується репродукція популяції з використанням різних методів паралелізації в .NET.

Розглянемо і опишемо в скороченому вигляді велику наступну структурну частину коду.

#region Reproduction // Definition Probability According by chromosome fitness

```
// create Pn[N_keep]; Rank_Trim();
        if (pGAToolStripMenuItem.Checked) // Parallel Genetic Algorithm
{ MultiThreading Task Parallelism
Parallel.For ... }
        if (threadParallelismToolStripMenuItem.Checked) // PGA by
{ ReproduceByParallelThreads(); } else if
(taskParallelismToolStripMenuItem.Checked) // PGA by { ReproduceByParallelTask();
} else if (parallelForToolStripMenuItem.Checked) // PGA by { PReproduction(rand); }
else // Series Genetic Algorithm { #region Series Reproduct Code Reproduction(rand);
#endregion } #endregion
        count++; SetGenerationText(count.ToString());
        SetValue(toolStripProgressBar1.Value + 1);
        }while(count<toolStripProgressBar1.Maximum && Isotropy_Evaluatuon());
        #region Generation Tools //x_Rate According by chromosome fitness Average
private void x_Rate()
{
//Code }
        // Definition Probability According
//by chromosome fitness private void Rank_Trim()
{
//Code
}
        //Return chromosome with Probability
private Chromosome Rank(Random rand) {
//Code
```

Функціональність, представлена в цьому фрагменті коду, відповідає за ключові етапи циклу генетичного алгоритму: селекцію батьківських хромосом для репродукції, перевірку критерію зупинки алгоритму та визначення розміру репродуктивної популяції.

По-перше, метод Rank_Trim() обчислює ймовірності вибору кожної

хромосоми з поточної популяції для подальшого схрещування. Ці ймовірності залежать від значення фітнес-функції хромосом - чим вище пристосованість хромосоми, тим більша ймовірність, що вона буде обрана. Таким чином реалізується механізм селекції, який сприяє виживанню найбільш пристосованих особин.

По-друге, власне вибір батьківських хромосом для схрещування в подальшому відбувається у методі Rank(). Тут на основі згенерованого випадкового числа та попередньо розрахованих ймовірностей обирається хромосома з поточної популяції. Так формуються "батьки" для наступного етапу репродукції.

По-третє, у методі Isotropy_Evaluation() реалізована перевірка критерію зупинки алгоритму. Якщо понад 50% хромосом гірші за найкращу хромосому в популяції, то вважається що досягнута "ізотропність" і подальші ітерації алгоритму не приведуть до поліпшення результату.

Нарешті, метод x_Rate() слугує для визначення розміру репродуктивної популяції N_keep, тобто кількості хромосом, які перейдуть в наступне покоління після репродукції. Це дозволяє позбутися найгірших хромосом і сприяє конвергенції алгоритму.

Надалі програма реалізує паралельну репродукцію (схрещування і мутацію) хромосом в генетичному алгоритмі з використанням багатопоточності, завдань (Tasks) та Parallel.For.

Спочатку визначається кількість потоків або завдань відповідно до кількості ядер процесора.

Далі для кожного потоку або завдання викликається метод PReproduction, який виконує схрещування та мутацію частини хромосом популяції. Синхронізація між потоками відбувається за допомогою семафорів або методів очікування завершення Task.

Окремо реалізований послідовний варіант (Reproduction), який працює в одному потоці.

В методі PReproduction спочатку обираються батьківські хромосоми через

виклик Rank(). Далі виконується схрещування, мутація та оцінка пристосованості дитячої хромосоми.

Таким чином досягається прискорення роботи алгоритму за рахунок розпаралелювання найбільш ресурсовитратної частини - репродукції популяції.

Надалі реалізована обробка події кліку миші на формі. У випадку кліку створюється нове місто в позиції курсора. Для цього спочатку перевіряється, що клік відбувся всередині робочої області форми. Потім викликається метод Stop() для зупинки алгоритму, очищення попередніх даних і виклик методу create_City() для створення нового OvalShape з заданими координатами.

Коли користувач змінює значення на полі введення розміру популяції numPopulation, викликається відповідний обробник події і оновлюється властивість PopulationNumber.

При закритті форми викликається метод зупинки алгоритму Stop() для коректного вивільнення ресурсів.

Меню "File -> New" викликає процедуру очищення усіх даних - видалення міст та ліній зв'язків, обнулення лічильників тощо.

Реалізовані також функції імпорту та експорту даних про розташування міст з текстових файлів.

Окремо присутня можливість генерувати задану кількість випадкових міст з урахуванням мінімальної відстані між ними для уникнення накладання.

Отже, вся ця функціональність забезпечує гнучкість та зручність керування програмою для користувача.

Розглянемо заключні операції по розробці функціоналу.

```
private void pGAToolStripMenuItem_Click(object sender, EventArgs e)
{
// if checked then Parallel Genetic Algorithm Enable
pGAToolStripMenuItem.Checked = !pGAToolStripMenuItem.Checked;
if       (pGAToolStripMenuItem.Checked)
taskParallelismToolStripMenuItem.Checked = true; else
{
```

```
        taskParallelismToolStripMenuItem.Checked = false;
        threadParallelismToolStripMenuItem.Checked = false;
parallelForToolStripMenuItem.Checked = false;
        }
        // show Panel
        ProcessPriorityToolStripMenuItem.ShowDropDown();
pGAToolStripMenuItem.ShowDropDown();
        }
        }
        public struct ThreadToken
        {
        public ThreadToken(int threadNo, int length, int startIndex)
        {
        No = threadNo;
        Length = length;
        StartIndex = startIndex;
        Rand = new System.Random();
        }
        public int No;
        public int Length;
        public int StartIndex;
        public System.Random Rand;
        };
        }
```

Ця частина коду містить обробку подій кнопок "Start/Stop" та "Pause/Resume", а також налаштування пріоритету потоку виконання алгоритму і режимів паралельного генетичного алгоритму.

Коли користувач натискає на кнопку "Start/Stop", виконується запуск або зупинка роботи генетичного алгоритму в окремому потоці _runTime.

При старті виконуються наступні дії:

- блокується можливість зміни розміру популяції;

- очищаються попередні графічні елементи інтерфейсу;

- створюється новий потік і запускається метод генетичного алгоритму Ga().

Аналогічно при паузі/продовженні алгоритму виконується призупинення/ поновлення потоку _runTime за допомогою Suspend()/Resume().

Також реалізоване налаштування пріоритету потоку виконання відповідно до обраного значення в меню priority. При зміні пріоритету викликається метод SetThreadPriority() для потоку _runTime.

Окремо є можливість вибору одного з трьох режимів паралельного генетичного алгоритму. Зміна режиму супроводжується скиданням відповідних прапорців.

При натисканні на кнопку "Start/Stop" відбувається запуск алгоритму в окремому потоці _runTime за допомогою методу Ga(). Перед запуском виконуються наступні підготовчі кроки:

1. Блокується можливість зміни розміру популяції користувачем, щоб уникнути збоїв.
2. Видаляються старі графічні елементи інтерфейсу (лінії маршруту).
3. Створюється новий список ліній для майбутньої візуалізації результатів.

Кнопка "Pause/Resume" дозволяє призупинити або продовжити виконання алгоритму в потоці за допомогою виклику Suspend() та Resume() відповідно.

Також реалізована можливість зміни пріоритету виконання потоку алгоритму через відповідне меню Priority. Після зміни викликається SetThreadPriority() для застосування нового пріоритету.

І нарешті, доступний вибір одного з трьох режимів паралельного генетичного алгоритму (PGA) - за допомогою багатопоточності, завдань або Parallel.For.

Отже, тут знаходиться вся бізнес-логіка, пов'язана з життєвим циклом та параметрами виконання генетичного алгоритму в програмі.

3.3 Тестування програмного забезпечення для розв’язанная задачі комівояжера із застосуванням генетичного алгоритму

Реалізація паралельної програми (Visual C# + .Net 4 MultiTasking) виконується в середовищі програмування Visual Studio 2010. Профілювання паралелізму виконано для випадку 500 міст. Результати обчислень паралельно та в серії трьох графіків: час-придатність, час-генерація, фітнес-генерація.

Очевидно, що комунікація є досить інтенсивною між менеджером і працівниками, які обмінюються даними про людей (хромосоми та фізичну форму). Комунікаційні транзакції виконуються через Task, тому затримка зв’язку дуже мала. Раніше основним недоліком паралельного генетичного підходу до обчислення TSP з алгоритмічною парадигмою менеджер/працівники є послідовне обчислення генетичних операцій – відбору, мутації та відтворення – процесом менеджера. Але тепер ці операції виконуються на робітниках.

При оцінці запропонованого паралельного алгоритму з послідовним алгоритмом, згідно з рисунками 3.1 і 3.2, можна побачити, що навіть придатність елітних хромосом у послідовних поколіннях покращується набагато швидше. Тому що незалежність у процесах відтворення (батьківська несхожість) у дітей є недостатньою конвергенцією. Результат покращується, щоб швидко досягти бажаної форми. Звичайно, враховуючи цифри оцінки на прикладі TSP, слід враховувати, що Fitness – це відстань між містами, тож наскільки нижча, ефективніша та краща вартість.

Тому можна зробити висновок, що ефективність оптимального пошуку маршруту для розв’язання задачі комівояжера зросла приблизно на 4,7% у порівнянні з аналогічними програмами розрахунку.

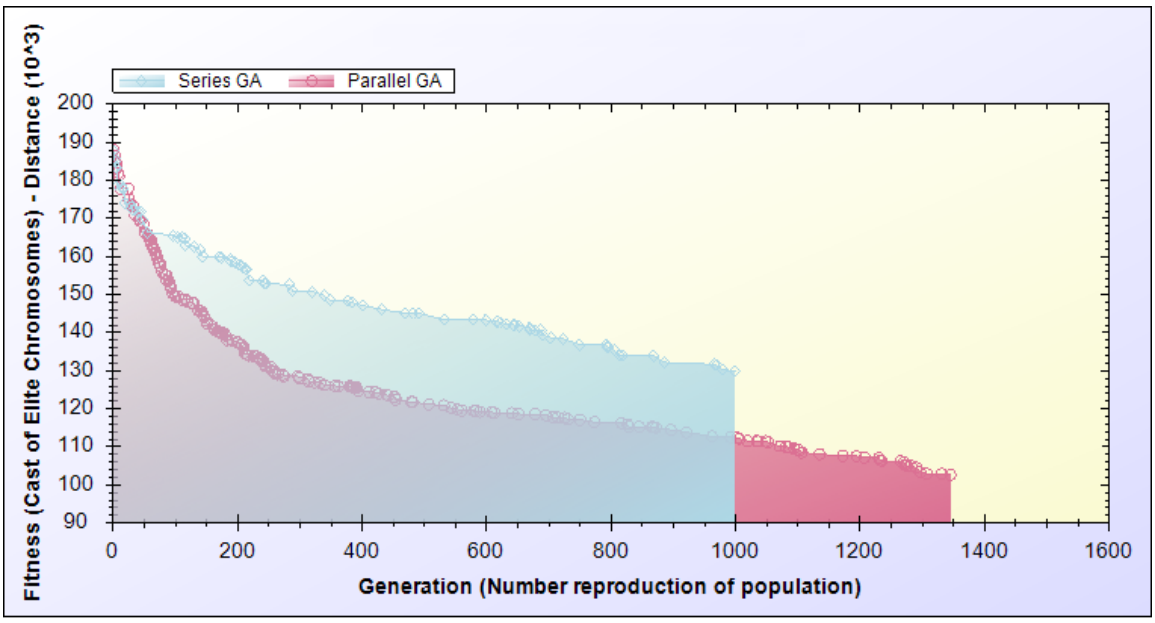


Рисунок 3.1 – Порівняння найкращої придатності будь-якого покоління як у стані послідовної, так і в паралельній обробці

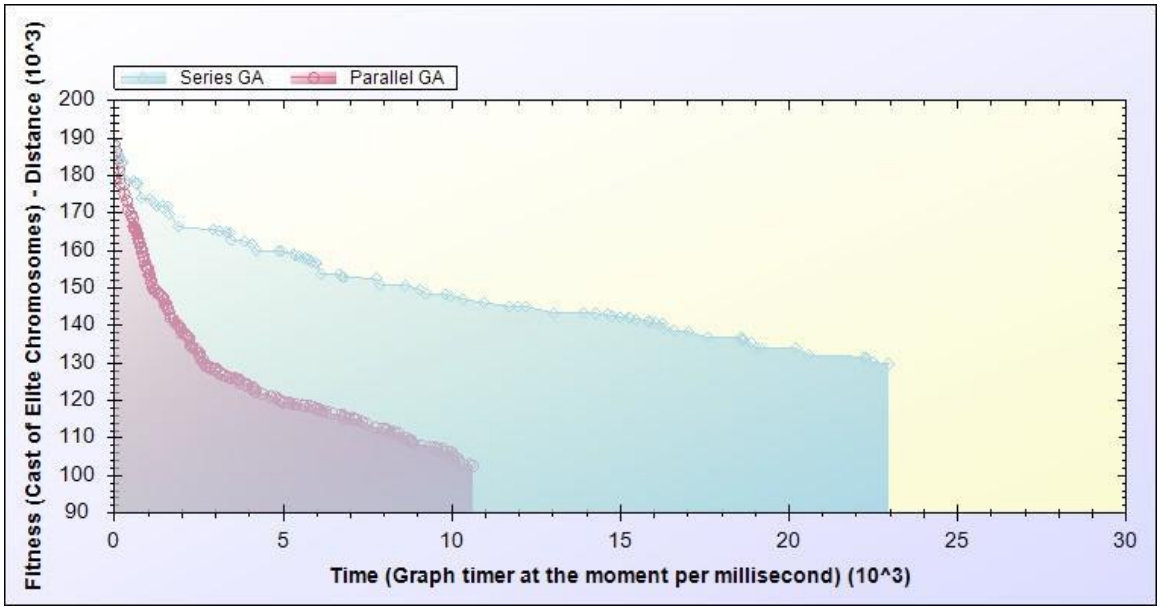


Рисунок 3.2 – Порівняння придатності елітних хромосом

Наступний результат оцінки результатів згідно з рисунком 3.3 полягає в тому, що темпи виробництва продукції є набагато швидшими, ніж стан серії.

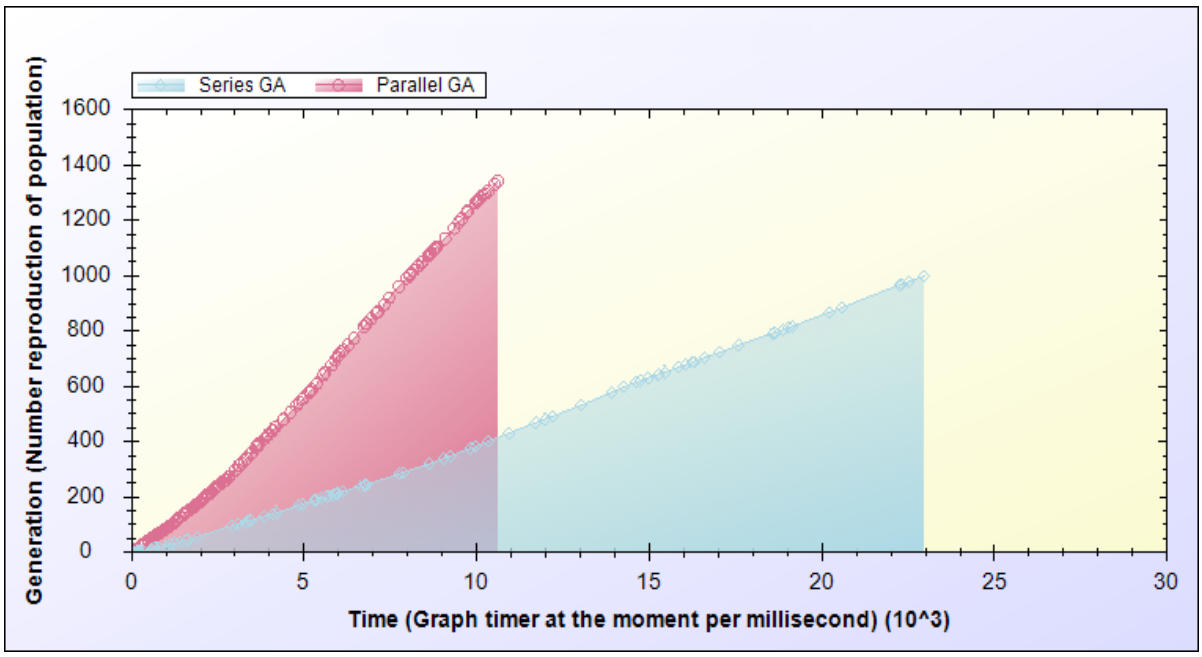


Рисунок 3.3 – Порівняння кількості поколінь, створених у кожен момент серії та паралельної обробки

3.4 Висновок до розділу 3

При виборі мови програмування та середовища розробки для створення програмного рішення, що вирішує задачу комівояжера за допомогою генетичного алгоритму, було обрано Visual Studio та мову програмування C#. У них реалізовані ключові модулі, які чітко відображають алгоритмічні аспекти та оптимізацію маршруту для даної задачі.

Тестування додатку на різних вхідних даних та сценаріях підтвердило його працездатність та ефективність у вирішенні вихідної задачі. Вибір конкретної мови та середовища розробки відповідає поставленим вимогам, забезпечуючи оптимальну швидкість реалізації програмного рішення. Розроблені модулі та їх випробування підтверджують надійність та функціональність отриманого продукту. Тестування розробленого програмного забезпечення підтверджує, що ефективність оптимального пошуку маршруту для розв’язання задачі комівояжера зросла приблизно на 4,7% у порівнянні з аналогічними програмами розрахунку.

4 ЕКОНОМІЧНА ЧАСТИНА

Виконання науково-дослідної роботи завжди передбачає отримання певних результатів і вимагає відповідних витрат. Результати виконаної роботи завжди дають нам нові знання, які в подальшому можуть бути використані для удосконалення та/або розробки (побудови) нових, більш продуктивних зразків техніки, процесів та програмного забезпечення.

Дослідження на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» може бути віднесено до фундаментальних і пошукових наукових досліджень і спрямоване на вирішення наукових проблем, пов’язаних з практичним застосуванням. Основою таких досліджень є науковий ефект, який виражається в отриманні наукових результатів, які збільшують обсяг знань про природу, техніку та суспільство, які розвивають теоретичну базу в тому чи іншому науковому напрямку, що дозволяє виявити нові закономірності, які можуть використовуватися на практиці.

Для цього випадку виконаємо такі етапи робіт:

1) здійснимо проведення наукового аудиту досліджень, тобто встановлення їх наукового рівня та значимості;

2) проведемо планування витрат на проведення наукових досліджень;

3) здійснимо розрахунок рівня важливості наукового дослідження та перспективності, визначимо ефективність наукових досліджень.

4.1 Оцінювання наукового ефекту

Основними ознаками наукового ефекту науково-дослідної роботи є новизна роботи, рівень її теоретичного опрацювання, перспективність, рівень розповсюдження результатів, можливість реалізації. Науковий ефект НДР на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» можна охарактеризувати двома показниками: ступенем наукової новизни та рівнем теоретичного опрацювання.

Значення показників ступеня новизни і рівня теоретичного опрацювання науково-дослідної роботи в балах наведені в таблицях 4.1 та 4.2.

Таблиця 4.1 – Показники ступеня новизни науково-дослідної роботи виставлені експертами

Ступінь новизни	Характеристика ступеня новизни	Значення ступеня новизни, бали		
		Експерти (ПІБ, посада)		
		1	2	3
Принципово нова	Робота якісно нова за постановкою задачі і ґрунтується на застосуванні оригінальних методів дослідження. Результати дослідження відкривають новий напрям в даній галузі науки і техніки. Отримані принципово нові факти, закономірності; розроблена нова теорія. Створено принципово новий пристрій, спосіб, метод	0	0	0
Нова	Отримана нова інформація, яка суттєво зменшує невизначеність наявних значень (по-новому або вперше пояснені відомі факти, закономірності, впроваджені нові поняття, розкрита структура змісту). Проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів	48	52	55
Відносно нова	Робота має елементи новизни в постановці задачі і методах дослідження. Результати дослідження систематизують і узагальнюють наявну інформацію, визначають шляхи подальших досліджень; вперше знайдено зв’язок (або знайдено новий зв’язок) між явищами. В принципі відомі положення розповсюджені на велику кількість об’єктів, в результаті чого знайдено ефективне рішення. Розроблені більш прості способи для досягнення відомих результатів. Проведена часткова раціональна модифікація (з ознаками новизни)	0	0	0
Традиційна	Робота виконана за традиційною методикою. Результати дослідження мають інформаційний характер. Підтверджені або поставлені під сумнів відомі факти та твердження, які потребують перевірки. Знайдено новий варіант рішення, який не дає суттєвих переваг в порівнянні з існуючим	0	0	0
Не нова	Отримано результат, який раніше зафіксований в інформаційному полі, та не був відомий авторам	0	0	0
Середнє значення балів експертів		51,7		

Згідно отриманого середнього значення балів експертів ступінь новизни характеризується як нова, тобто отримана нова інформація, яка суттєво зменшує невизначеність наявних знань (по-новому або вперше пояснені відомі факти, закономірності, впроваджені нові поняття, розкрита структура змісту) та проведено суттєве вдосконалення, доповнення і уточнення раніше досягнутих результатів.

Таблиця 4.2 – Показники рівня теоретичного опрацювання науково-дослідної роботи виставлені експертами

Характеристика рівня теоретичного опрацювання	Значення показника рівня теоретичного опрацювання, бали		
	Експерт (ПІБ, посада)		
	1	2	3
Відкриття закону, розробка теорії	0	0	0
Глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу	60	62	60
Розробка способу (алгоритму, програми), пристрою, отримання нової речовини	0	0	0
Елементарний аналіз зв'язків між фактами та наявною гіпотезою, класифікація, практичні рекомендації для окремого випадку тощо	0	0	0
Опис окремих елементарних фактів, викладення досвіду, результатів спостережень, вимірювань тощо	0	0	0
Середнє значення балів експертів	60,7		

Згідно отриманого середнього значення балів експертів рівень теоретичного опрацювання науково-дослідної роботи характеризується як глибоке опрацювання проблеми: багатоаспектний аналіз зв'язків, взаємозалежності між фактами з наявністю пояснень, наукової систематизації з побудовою евристичної моделі або комплексного прогнозу.

Показник, який характеризує рівень наукового ефекту, визначаємо за формулою [20]:

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор}, \quad (4.1)$$

де $k_{нов}, k_{теор}$ - показники ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи, $k_{нов} = 51,7$, $k_{теор} = 60,7$ балів;

0,6 та 0,4 – питома вага (значимість) показників ступеня новизни та рівня теоретичного опрацювання науково-дослідної роботи.

$$E_{нау} = 0,6 \cdot k_{нов} + 0,4 \cdot k_{теор} = 0,6 \cdot 51,7 + 0,4 \cdot 60,67 = 55,27 \text{ балів.}$$

Визначення характеристики показника $E_{нау}$ проводиться на основі висновків експертів виходячи з граничних значень, які наведені в таблиці 4.3.

Таблиця 4.3 – Граничні значення показника наукового ефекту

Досягнутий рівень показника	Кількість балів
Високий	70…100
Середній	50…69
Достатній	15…49
Низький (помилкові дослідження)	1…14

Відповідно до визначеного рівня наукового ефекту проведеної науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму», даний рівень становить 55,27 балів і відповідає статусу - середній рівень. Тобто у даному випадку можна вести мову про потенційну фактичну ефективність науково-дослідної роботи.

4.2 Розрахунок витрат на здійснення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп,

науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [20]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.2)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_о = 16800{,}00 \cdot 36 / 21 = 28800{,}00 \; грн.$$

Проведені розрахунки зведемо до таблиці 4.4.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник науково-дослідної роботи	16800,00	800,00	36	28800,00
Науковий співробітник	16500,00	785,71	32	25142,86
Консультант (фахівець логістичної служби)	17000,00	809,52	5	4047,62
Лаборант	7000,00	333,33	32	10666,67
Всього				68657,14

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{4.3}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{М_М \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}} \quad , \tag{4.4}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [10];

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$Т_р$ – середнє число робочих днів в місяці, приблизно $Т_р$ = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}35 \,/\, (21 \cdot 8) = 59{,}22 \ грн.$$
$$З_{р1} = 59{,}22 \cdot 16{,}00 = 947{,}57 \ грн.$$

Проведені розрахунки зведемо до таблиці 4.5.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Організація робочих місць учасників проекту	16,00	2	1,10	59,22	947,57
Установка обчислювального обладнання	4,35	3	1,35	72,68	316,17
Інсталяція програмного забезпечення	3,50	5	1,70	91,53	320,34
Підготовка бази даних	12,00	2	1,10	59,22	710,68
Проведення обчислювального експерименту	7,00	4	1,50	80,76	565,31
Тестування за змінних умов	5,00	3	1,35	72,68	363,42
Всього					3223,49

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$$З_{дод} = (68657,14 + 3223,49) \cdot 11 / 100\% = 7906,87 \text{ грн.}$$

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (4.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (68657,14 + 3223,49 + 7906,87) · 22 / 100% = 17553,25 грн.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму».

Витрати на матеріали на даному етапі проведення досліджень в основному пов’язані з використанням моделей елементів та моделювання роботи і досліджень за допомогою комп’ютерної техніки та створення експериментальних математичних моделей або програмного забезпечення, тому дані витрати формуються на основі витратних матеріалів характерних для офісних робіт.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_{вj} Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (4.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 2,0 · 184,00 · 1,1 - 0 · 0 = 404,80 грн.

Проведені розрахунки зведемо до таблиці 4.6.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір SOFT Ultra Plus	184,00	2,0	0	0	404,80
Папір для записів Parers SOFT Light A5	114,00	3,0	0	0	376,20
Органайзер офісний OFFICE SOFT	216,00	3,0	0	0	712,80
Канцелярське приладдя (набір офісного працівника)	210,00	2,0	0	0	462,00
Картридж для принтера Canon LBP6500	1865,00	1,0	0	0	2051,50
Диск оптичний NewOPtice CD-RW	28,00	5,0	0	0	154,00
Flesh-пам'ять Kingston 16 GB	99,00	2,0	0	0	217,80
Тека для паперів BOSS PAPERS BOX	75,00	6,0	0	0	495,00
Всього					4874,10

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» відсутні.

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення. Витрати за статтею «Спецустаткування» - відсутні.

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \tag{4.10}$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$$B_{прг} = 6380,00 \cdot 1 \cdot 1,1 = 7018,00 \text{ грн.}$$

Отримані результати зведемо до таблиці 4.7

Таблиця 4.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Visual Studio Community	1	6380,00	7018,00
Програмний пакет імітаційного моделювання	1	7400,00	8140,00
Всього			15158,00

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12} \quad , \tag{4.11}$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$$A_{обл} = (36899{,}00 \cdot 2) / (3 \cdot 12) = 2049{,}94 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 4.8.

Таблиця 4.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп'ютер	36899,00	3	2	2049,94
Робоче місце дослідника	10300,00	5	2	343,33
Пристрої виводу інформації	6520,00	4	2	271,67
Оргтехніка	7100,00	4	2	295,83
Приміщення лабораторії	470000,00	25	2	3133,33
ОС Windows 10	5700,00	3	2	316,67
Прикладний пакет Microsoft Office 2016	5120,00	3	2	284,44
Обладнання передачі даних	9600,00	5	2	320,00
Всього				7015,22

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{уі} \cdot t_i \cdot Ц_е \cdot K_{впі}}{\eta_i}, \quad (4.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вп i}$ – коефіцієнт, що враховує використання потужності, $K_{вп i} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$$B_e = 0,25 \cdot 280,0 \cdot 7,50 \cdot 0,95 / 0,97 = 525,00 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці 4.9.

Таблиця 4.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп'ютер	0,25	280,0	525,00
Робоче місце дослідника	0,08	280,0	168,00
Пристрої виводу інформації	0,12	3,5	3,11
Оргтехніка	0,32	2,0	4,80
Обладнання передачі даних	0,15	280,0	315,00
Всього			1015,91

Службові відрядження, що входять до складу обговорюваної теми "Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму", охоплюють різноманітні види витрат, пов'язаних з участю штатних працівників, працівників організацій, що діють на підставі цивільно-правових угод, аспірантів та інших учасників, присвячених дослідженню та розробці інформаційних технологій.

Перш за все, витрати на відрядження включають у себе розходи на перевезення та проживання штатних працівників та інших учасників проекту. Це може включати в себе вартість квитків на транспортні засоби, готелі, харчування

та інші пов'язані витрати, пов'язані з тимчасовим перебуванням на місці проведення роботи або подій.

До цього можуть долучатися і витрати, пов'язані з відрядженням аспірантів, які займаються розробленням досліджень. Це може включати витрати на навчання, консультації, а також необхідні матеріали та обладнання для успішного виконання досліджень.

Крім того, витрати на відрядження можуть стосуватися випробувань машин та приладів, які використовуються в процесі дослідження. Це включає в себе транспортування обладнання, його встановлення та налагодження, а також оплату послуг фахівців, які взяли участь у випробуваннях.

Нарешті, важливо відзначити, що витрати на відрядження також охоплюють участь у наукових з'їздах, конференціях та нарадах, що пов'язані з виконанням конкретних досліджень. Вони можуть включати в себе вартість участі, подорожі, проживання та інші витрати, необхідні для активної участі в цих заходах, що сприяють обміну знанням та дискусіям у відповідних галузях науки.

Витрати за статтею «Службові відрядження» відсутні.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» відсутні.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \qquad (4.15)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 60%.

$$I_в = (68657,14 + 3223,49) \cdot 60 / 100\% = 43128,38 \text{ грн.}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{H_{нзв}}{100\%}, \quad (4.16)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 115%.

$B_{нзв} = (68657,14 + 3223,49) \cdot 115 / 100\% = 82662,73$ *грн.*

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + A_{обл} + B_е + B_{св} + B_{сп} + I_в + B_{нзв}. \quad (4.17)$$

$B_{заг}$= *68657,14 +3223,49 +7906,87 +17553,25 +4874,10 +0,00 +0,00 + 15158,00 + 7015,22 +1015,91 +0,00 +0,00 +43128,38 +82662,73 = 251195,10 грн.*

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.18)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

$$ЗВ = 251195,10 / 0,9 = 279105,66 \text{ грн.}$$

4.3 Оцінювання важливості та наукової значимості науково-дослідної роботи

Оцінювання та доведення ефективності виконання науково-дослідної роботи фундаментального чи пошукового характеру є достатньо складним процесом і часто базується на експертних оцінках, тому має вірогідний характер.

Для обґрунтування доцільності виконання науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» використовується спеціальний комплексний показник, що враховує важливість, результативність роботи, можливість впровадження її результатів у виробництво, величину витрат на роботу.

Комплексний показник K_P рівня науково-дослідної роботи може бути розрахований за формулою:

$$K_P = \frac{I^n \cdot T_C \cdot R}{B \cdot t}, \quad (4.19)$$

де I – коефіцієнт важливості роботи. Приймемо $I = 4$;

n – коефіцієнт використання результатів роботи; $n = 0$, коли результати роботи не будуть використовуватись; $n = 1$, коли результати роботи будуть використовуватись частково; $n = 2$, коли результати роботи будуть використовуватись в дослідно-конструкторських розробках; $n = 3$, коли результати можуть використовуватись навіть без проведення дослідно-конструкторських розробок. Приймемо n=3;

T_C – коефіцієнт складності роботи. Приймемо $T_C = 3$;

R – коефіцієнт результативності роботи; якщо результати роботи плануються вище відомих, то $R = 4$; якщо результати роботи відповідають відомому рівню, то $R = 3$; якщо нижче відомих результатів, то $R = 1$. Приймемо $R = 4$;

B – вартість науково-дослідної роботи, тис. грн. Приймемо B =279105,66 грн;

t – час проведення дослідження. Приймемо t = 0,17 років, (2 міс.).

Визначення показників I, n, T_C, R, B, t здійснюється експертним шляхом або на основі нормативів[10].

$$K_P = \frac{I^n \cdot T_C \cdot R}{B \cdot t} = 4^3 \cdot 3 \cdot 4 \,/\, 279{,}1 \cdot 0{,}17 = 16{,}51.$$

Якщо $K_P > 1$, то науково-дослідну роботу на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» можна вважати ефективною з високим науковим, технічним і економічним рівнем.

4.4 Висновок до розділу 4

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» складають 279105,66 грн. Відповідно до проведеного аналізу та розрахунків рівень наукового ефекту проведеної науково-дослідної роботи на тему «Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму» є середній, а дослідження актуальними, рівень доцільності виконання науково-дослідної роботи $K_P > 1$, що свідчить про потенційну ефективність з високим науковим, технічним і економічним рівнем.

ВИСНОВКИ

У магістерській кваліфікаційній рооботі було розроблено інформаційну технологію розв’язання задачі комівояжера із застосуванням генетичного алгоритму. Проведено дослідження застосування генетичних алгоритмів для розв'язання задачі комівояжера. Було проаналізовано предметну область задачі, обґрунтовано застосування генетичних алгоритмів в розв'язанні даної задачі. У процесі дослідження було проведено порівняльний аналіз програм-аналогів, що розв'язують задачу комівояжера, та вибрано оптимальний підхід до реалізації генетичного алгоритму. Під час розробки інформаційної технології для розв'язання задачі комівояжера з використанням генетичного алгоритму було побудовано UML-діаграми, що відображають структуру та взаємодію компонентів системи. Структурні схеми застосування генетичного алгоритму в задачі комівояжера чітко відображають процес оптимізації маршруту та взаємодію між окремими етапами алгоритму. Ці діаграми і схеми стали основою для подальшої програмної реалізації алгоритму та його належного тестування, забезпечуючи зручний аналіз та розуміння роботи системи. При виборі мови програмування та середовища розробки для створення програмного рішення, що вирішує задачу комівояжера за допомогою генетичного алгоритму, було обрано Visual Studio та мову програмування C#. У них реалізовані ключові модулі, які чітко відображають алгоритмічні аспекти та оптимізацію маршруту для даної задачі.

Тестування додатку на різних вхідних даних та сценаріях підтвердило його працездатність та ефективність у вирішенні вихідної задачі. Вибір конкретної мови та середовища розробки відповідає поставленим вимогам, забезпечуючи оптимальну швидкість реалізації програмного рішення. Розроблені модулі та їх випробування підтверджують надійність та функціональність отриманого продукту. Тестування розробленого програмного забезпечення підтверджує, що ефективність оптимального пошуку маршруту для розв’язання задачі комівояжера зросла приблизно на 4,7% у порівнянні з аналогічними програмами розрахунку.

Отже всі задачі дослідження виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. С.І. Петришин, О.І. Гнаповський. Інформаційна технологія розв’язання задачі комівояжера із застосуванням генетичного алгоритму. Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2023)» − [Електронний ресурс]. – https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2023/paper/view/17457

2. I. Berninger, “Проблема маршрутизації транспортних засобів на Android/iOS,” Бакалаврська робота, Інститут комп’ютерних наук, дослідницька група DPS, Університет Інсбрука, 2014.

3. A. Homaifar, S. Guan, and G. E. Liepins, “Аналіз схеми проблеми комівояжера за допомогою генетичних алгоритмів,” Складні системи 6, с. 533-552, 1992.

4. A. Reese, “Генератори випадкових чисел в генетичних алгоритмах для безобмеженої та обмеженої оптимізації,” J Нелінійний аналіз, 71, 679–692, 2009.

5. Y. Yun, C. Moon, and D. Kim. “Гібридний генетичний алгоритм з адаптивною схемою локального пошуку для вирішення проблем у сфері багаторівневого постачання,” Comput Ind Eng 56, 821–838, 2009.

6. N. M. Razali and J. Geraghty, “Ефективність генетичного алгоритму з різними стратегіями вибору при розв’язанні проблеми комівояжера,” Процедури Всесвітнього конгресу з інженерії 2011, Том II WCE 2011, 6 - 8 липня 2011, Лондон, Великобританія.

7. K. Rani and V. Kumar, Int. J. Res.Eng. Tech. 2, 27-34. (2014)

8. Z. H. Ahmed, “Експериментальне дослідження гібридного генетичного алгоритму для проблеми максимального комівояжера,” Springer open journal, 2013.

9. K. Bryant, “Генетичні алгоритми та проблема комівояжера,” Магістерська робота, Коледж Харві Мадда, 2000.

10. Y. Wang, “Гібридний генетичний алгоритм з двома стратегіями локальної оптимізації для проблеми комівояжера,” Comput. Ind. Eng. 70, с. 124–133 (2014).

11. How to change the color of progressbar in C# .NET 3.5? Режим доступу: https://stackoverflow.com/questions/778678/how-to-change-the-color-of-progressbar-in-c-sharp-net-3-5 (дата звернення черв. 2022).

12. Наконечний С.І., Савіна С.С. Математичне програмування: Навч.посіб. – К.: КНЕУ, 2003. – 452 с.

13. C++ 17 STL. Стандартна бібліотека шаблонів. Яцек Галовіц. 2018. ISBN: 978-5-4461-0680-6

14. Парадигми програмування. Елементи програм C++ [Electronic resource]. – Mode of access : WWW/URL : http://om.univ.kiev.ua/users_upload/15/upload/ file/prog_lecture_01.pdf. – Title from the screen.

15. Кравець П. Об'єктно-орієнтоване програмування : навч. посібник / П.О. Кравець. – Львів: Видавництво Львівської політехніки, 2012. – 624 с.

16. Josuttis, Nicolai M. The C++ standard library : a tutorial and reference / Nicolai M. Josuttis.—2nd ed. c. 78-80.

17. ITVDN Об’єктно-орієнтоване програмування C# в деталях / ITVDN. – Режим доступу: https://itvdn.com/ua/blog/article/oop-csharp-basic

18. Глинський Я. М., Анохін В. Є., Ряжська В. А. C++ і C++ Builder. Львів : СПД Глинський, 2011.

19. Войтенко В.В. Морозов А.В. Теорія та практика (мова C++). — Житомир, 2002.

20. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія розв'язання задачі комівояжера із застосуванням генетичного алгоритму

Тип роботи: магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 80,2% Схожість 19,8%

Аналіз звіту подібності (відмітити потрібне):

✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Гнаповський О.І.

Керівник роботи Петришин С.І.

Опис прийнятого рішення

Магістерську кваліфікаційну роботу допущено до захисту

Особа, відповідальна за перевірку Озеранський В.С.

Додаток Б (обов'язковий)

Лістинг програми

```
using
System;
          using System.Collections.Generic;
          using System.Diagnostics;
          using System.Drawing;
          using System.IO;
          using System.Linq;
          using System.Threading;
          using System.Threading.Tasks;
          using System.Windows.Forms;
          using Microsoft.VisualBasic.PowerPacks;
          using TSP.Core;
          using TSP.TimerGraphs;
          using ZedGraph;
          using ThreadState = System.Threading.ThreadState;


          namespace TSP
          {
              public partial class MainForm : Form
              {
                  // Properties
                  [STAThread]
                  static void Main()
                  {
                      Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High;
                      Application.EnableVisualStyles();
                      Application.Run(new MainForm());
                  }


                  #region Define Varibale
                  PointPairList[] _pPlistTfg;
                  PointPairList[] _pPlistTgg;
                  PointPairList[] _pPlistGfg;


                  TimeFitnessGraph _tfg;
                  TimeGenerationGraph _tgg;
                  GenerationFitnessGraph _gfg;


                  CancellationTokenSource _tokenSource;
                  int _startedTick = 0;


                  public int CountCpuCore { get; set; } = 1;


                  // Number Population
                  public int PopulationNumber { get; set; } = 500;


                  // Number Keep Chromosome Size
                  int _nKeep = 0;
```

```
        // Double Array Pn for save Rank
        double[] _pn;


        private ShapeContainer ShapeContainerAllCityShape { get; set; }
        private List<LineShape> LineShapeWay { get; set; } = new List<LineShape>();
        public List<OvalShape> OvalShapeCity { get; set; } = new List<OvalShape>();


        // save number of all city
        public int CounterCity { get; private set; }
        public GeneticAlgorithm Genetic { get; private set; }


        // create new process or for end process
        Thread _runTime;


        #endregion


        public MainForm()
        {
            InitializeComponent();


            //
            // shapeContainer_allCityShape
            //
            ShapeContainerAllCityShape = new ShapeContainer
            {
                Location = new Point(0, 0),
                Margin = new Padding(0),
                Size = new Size(Width, Height),
                TabIndex = 0,
                TabStop = false
            };


            Controls.Add(ShapeContainerAllCityShape);
            //
            // Make up some data points from the Sine function
            _pPlistTfg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
            _pPlistTfg[0] = new PointPairList(); // For Series GA (Time-Fitness)
Data's
            _pPlistTfg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's


            _pPlistTgg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
            _pPlistTgg[0] = new PointPairList(); // For Series GA (Time-Fitness)
Data's
            _pPlistTgg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's


            _pPlistGfg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
            _pPlistGfg[0] = new PointPairList(); // For Series GA (Time-Fitness)
Data's
```

```
            _pPlistGfg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's
        }


        #region Thread Invoked
        public static void UiInvoke(Control uiControl, Action action)
        {
            if (!uiControl.IsDisposed)
            {
                if (uiControl.InvokeRequired)
                {
                    uiControl.BeginInvoke(action);
                }
                else
                {
                    action();
                }
            }
        }


        // ------------------------------------------------------------
        // This delegate enables asynchronous calls for setting
        // the Value property on a toolStripProgressBar control.
        delegate void SetValueCallback(int v);
        private void SetValue(int v)
        {
            // InvokeRequired required compares the thread ID of the
            // calling thread to the thread ID of the creating thread.
            // If these threads are different, it returns true.
            try
            {
                if (statusStrip1.InvokeRequired)
                {
                    var d = new SetValueCallback(SetValue);
                    Invoke(d, new object[] { v });
                }
                else
                {
                    toolStripProgressBar1.Value = v;
                }
            }
            catch { }
        }


        // ------------------------------------------------------------
        // This delegate enables asynchronous calls for setting
        // the Maximum.Value property on a toolStripProgressBar control.
        delegate void SetMaxValueCallback(int v);
        private void SetMaxValue(int v)
        {
            // InvokeRequired required compares the thread ID of the
            // calling thread to the thread ID of the creating thread.
            // If these threads are different, it returns true.
            try
            {
                if (statusStrip1.InvokeRequired)
                {
                    var d = new SetMaxValueCallback(SetMaxValue);
                    Invoke(d, new object[] { v });
                }
                else
```

```
        {
            toolStripProgressBar1.Maximum = v;
        }
    }
    catch { }
}


// ------------------------------------------------------------
// This delegate enables asynchronous calls for setting
// the Value property on a toolStripProgressBar control.
private void SetGenerationText(string v)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    try
    {
        UiInvoke(lblGeneration, delegate ()
        {
            lblGeneration.Text = v;
        });
    }
    catch { }
}
// ------------------------------------------------------------
private void SetLenghtText(string v)
{
    try
    {
        try
        {
            UiInvoke(lblLenght, delegate ()
            {
                lblLenght.Text = v;
            });
        }
        catch { }
    }
    catch { }
}
// ------------------------------------------------------------
delegate void AddShapeCallback(LineShape l);
private void AddLineShape(LineShape l)
{
    try
    {
        if (ShapeContainerAllCityShape.InvokeRequired)
        {
            var d = new AddShapeCallback(AddLineShape);
            Invoke(d, new object[] { l });
        }
        else
        {
            ShapeContainerAllCityShape.Shapes.Add(l);
        }
    }
    catch
    {
        // ignored
    }
}
// ------------------------------------------------------------
delegate void RemoveShapeCallback(LineShape l);
```

```
        private void RemoveLineShape(LineShape l)
        {
            try
            {
                if (ShapeContainerAllCityShape.InvokeRequired)
                {
                    var d = new RemoveShapeCallback(RemoveLineShape);
                    Invoke(d, new object[] { l });
                }
                else
                {
                    ShapeContainerAllCityShape.Shapes.Remove(l);
                }
            }
            catch { }
        }
        // ------------------------------------------------------------
        delegate void SetPointCallback(int l, Point p0, Point p1);
        private void SetPoint(int l, Point p0, Point p1)
        {
            try
            {
                LineShapeWay[l].X1 = p0.X + 10;
                LineShapeWay[l].X2 = p1.X + 10;
                LineShapeWay[l].Y1 = p0.Y + 10;
                LineShapeWay[l].Y2 = p1.Y + 10;
            }
            catch { }
        }
        // ------------------------------------------------------------
        private void SetNumPopEnable(bool en)
        {
            try
            {
                try
                {
                    UiInvoke(numPopulation, delegate ()
                    {
                        numPopulation.Enabled = en;
                    });
                }
                catch { }
            }
            catch { }
        }
        #endregion


        /// <summary>
        /// Genetic Algorithm for TSP
        /// </summary>
        public void Ga()
        {
            var rand = new Random(DateTime.Now.GetHashCode());
            var eliteFitness = double.MaxValue;
            //
            // set cities position
            SetCitiesPosition(OvalShapeCity);
            //
            // initialize Parallel Computing for GA
            CountCpuCore = CalcCountOfCpu(); // Calculate number of active core or
CPU for this app
            _tokenSource = new CancellationTokenSource();
            //
```

```
            // set Start TickTime
            _startedTick = Environment.TickCount;


            if (pGAToolStripMenuItem.Checked)  // clear Parallel points
            {
                _pPlistTfg[1].Clear();
                _pPlistGfg[1].Clear();
                _pPlistTgg[1].Clear();
            }
            else // clear Series points
            {
                _pPlistTfg[0].Clear();
                _pPlistGfg[0].Clear();
                _pPlistTgg[0].Clear();
            }
            //
            //               N , Pop, SR, MR, ReGen, CR
            Genetic = new GeneticAlgorithm(CounterCity, PopulationNumber, 10, 50,
10000, 75);

            var count = 0;
            SetValue(0);
            //toolStripProgressBar1.Value = 0;
            //
            SetGenerationText("0000");
            //lblGeneration.Text = "0000";
            //
            if (CounterCity <= 5)
                SetMaxValue(100);
            //toolStripProgressBar1.Maximum = 100;
            //
            else if (CounterCity <= 15)
                SetMaxValue(1000);
            //toolStripProgressBar1.Maximum = 1000;
            //
            else if (CounterCity <= 30)
                SetMaxValue(10000);
            //toolStripProgressBar1.Maximum = 10000;
            //
            else if (CounterCity <= 40)
                SetMaxValue(51000);
            //toolStripProgressBar1.Maximum = 51000;
            //
            else if (CounterCity <= 60)
                SetMaxValue(100000);
            //toolStripProgressBar1.Maximum = 100000;
            //
            else
                SetMaxValue(1000000);
            //toolStripProgressBar1.Maximum = 1000000;
            //
            //


            do
            {
                #region Selection
                #region Bubble Sort all chromosome by fitness
                //
                for (var i = PopulationNumber - 1; i > 0; i--)
                    for (var j = 1; j <= i; j++)
                        if      (Genetic.Population[j      -      1].Fitness      >
Genetic.Population[j].Fitness)
```

```
                        {
                            var ch = Genetic.Population[j - 1];
                            Genetic.Population[j - 1] = Genetic.Population[j];
                            Genetic.Population[j] = ch;
                        }
                //
                #endregion


                #region Elitism
                if (eliteFitness > Genetic.Population[0].Fitness)
                {
                    eliteFitness = Genetic.Population[0].Fitness;
                    SetTimeGraph(eliteFitness, count, true);


                    if  (dynamicalGraphicToolStripMenuItem.Checked)  //  Design  if
Graphically is ON
                    {
                        RefreshTour();
                    }
                    //
                    //-------------------------------------------------------------
---------------
                    SetLenghtText(Genetic.Population[0].Fitness.ToString());
                    //
                }
                //else  setTimeGraph(EliteFitness,  count,  false);  //  just  refresh
Generation Graph's


                #endregion
                x_Rate(); // Selection any worst chromosome for clear and ...
                #endregion


                #region Reproduction
                // Definition Probability According by chromosome fitness
                // create Pn[N_keep];
                Rank_Trim();


                if (pGAToolStripMenuItem.Checked) // Parallel Genetic Algorithm
                {
                    if   (threadParallelismToolStripMenuItem.Checked)   //   PGA   by
MultiThreading
                    {
                        ReproduceByParallelThreads();
                    }
                    else if (taskParallelismToolStripMenuItem.Checked) // PGA by Task
Parallelism
                    {
                        ReproduceByParallelTask();
                    }
                    else   if   (parallelForToolStripMenuItem.Checked)   //   PGA   by
Parallel.For ...
                    {
                        PReproduction(rand);
                    }
                }
                else // Series Genetic Algorithm
                {
                    #region Series Reproduct Code
```

```
                Reproduction(rand);
                #endregion
            }
            #endregion


            count++;
            SetGenerationText(count.ToString());
            //lblGeneration.Text = count.ToString();
            //
            SetValue(toolStripProgressBar1.Value + 1);
            //toolStripProgressBar1.Value++;
            //
        }
        while (count < toolStripProgressBar1.Maximum && Isotropy_Evaluatuon());


        //
        //toolStripProgressBar1.Value = toolStripProgressBar1.Maximum;
        SetValue(toolStripProgressBar1.Maximum);
        //
        // UnLock numUpDownPop
        SetNumPopEnable(true);
        //
        // The END
        Stop();
    }


    #region Generation Tools


    //find percent of All chromosome rate for delete Amiss(xRate) or Useful(Nkeep)
chromosome
    //x_Rate According by chromosome fitness Average
    private void x_Rate()
    {
        // calculate Addition of all fitness
        double sumFitness = 0;
        for (var i = 0; i < PopulationNumber; i++)
            sumFitness += Genetic.Population[i].Fitness;
        // calculate Average of All chromosome fitness
        var aveFitness = sumFitness / PopulationNumber; //Average of all
chromosome fitness
        _nKeep = 0; // N_keep start at 0 till Average fitness chromosome
        for (var i = 0; i < PopulationNumber; i++)
            if (aveFitness >= Genetic.Population[i].Fitness)
            {
                _nKeep++; // counter as 0 ~ fitness Average + 1
            }
        if (_nKeep <= 0) _nKeep = 2;
    }


    // Definition Probability According by chromosome fitness
    private void Rank_Trim()
    {
        // First Reserve Possibility Number for every Remnant chromosome
        // chromosome Possibility Function is:
        // (1 + N_keep - No.chromosome) / ( ∑ No.chromosome)
        // Where as at this program No.chromosome Of Array begin as Number 0
        // There for No.chromosome in Formula = No.chromosome + 1
        // then function is: if (n == N_keep)
```

```
            // Possibility[No.chromosome] = (n - No.chromosome) / (n(n+1) / 2)
            //
            _pn = new double[_nKeep]; // Create chromosome possibility Array Cell as
N_keep
            double sum = ((_nKeep * (_nKeep + 1)) / 2); // (∑ No.chromosome) == (n(n+1)
/ 2)
            _pn[0] = _nKeep / sum; // Father (Best - Elite) chromosome Possibility
            for (var i = 1; i < _nKeep; i++)
            {
                // Example: if ( Pn[Elite] = 0.4  &  Pn[Elite +1] = 0.2  &  Pn[Elite
+2]  = 0.1 )
                // Then Own:          0 <= R <= 0.4 ===> Select chromosome[Elite]
                //                 0.4 <  R <= 0.6 ===> Select chromosome[Elite +1]
                //                 0.6 <  R <= 0.7 ===> Select chromosome[Elite +2]
                // etc ...
                _pn[i] = ((_nKeep - i) / sum) + _pn[i - 1];
            }
        }


        // Return Father and Mather chromosome with Probability of chromosome fitness
        private Chromosome Rank(System.Random rand)
        {
            var r = rand.NextDouble();
            for (var i = 0; i < _nKeep; i++)
            {
                // Example: if ( Pn[Elite] = 0.6  &  Pn[Elite+1] = 0.3  &  Pn[Elite+2]
= 0.1 )
                // Then Own:          0 <= R <= 0.6  ===> Select chromosome[Elite]
                //                 0.6 <  R <= 0.9  ===> Select chromosome[Elite +1]
                //                 0.9 <  R <= 1    ===> Select chromosome[Elite +2]
                //
                if (r <= _pn[i]) return Genetic.Population[i];
            }
            return Genetic.Population[0]; // if don't run Modality of 'for' then
return Elite chromosome
        }


        // Check the isotropy All REMNANT chromosome (N_keep)
        public bool Isotropy_Evaluatuon()
        {
            // Isotropy percent is 50% of All chromosome Fitness
            var perIso = Convert.ToInt32(Math.Truncate(Convert.ToDouble(50 * _nKeep /
100)));
            var counterIsotropy = 0;
            var bestFitness = Genetic.Population[0].Fitness;
            //
            // i start at 1 because DNA_Array[0] is self BestFitness
            for (var i = 1; i < _nKeep; i++)
                if (bestFitness >= Genetic.Population[i].Fitness) counterIsotropy++;


            // G.A Algorithm did isotropy and app Stopped
            if (counterIsotropy >= perIso) return false;
            else return true; // G.A Algorithm didn't isotropy and app will continued
        }


        private void ReproduceByParallelThreads()
        {
            #region Parallel Reproduct Code
            var th = new Thread[CountCpuCore];
```

```
            // Create a semaphore that can satisfy up to three
            // concurrent requests. Use an initial count of zero,
            // so that the entire semaphore count is initially
            // owned by the main program thread.
            //
            var sem = new Semaphore(CountCpuCore, CountCpuCore);
            var isAlive = new bool[CountCpuCore];
            var isCompleted = new bool[CountCpuCore];


            var length = (PopulationNumber - _nKeep) / CountCpuCore;
            var divideReminder = (PopulationNumber - _nKeep) % CountCpuCore;


            for (var proc = 0; proc < th.Length; proc++)
            {
                var tt = new ThreadToken(proc,
                    length + ((proc == CountCpuCore - 1) ? divideReminder : 0),
                    _nKeep + (proc * length));


                th[proc] = new Thread((x) =>
                {
                    // Entered
                    sem.WaitOne();
                    isAlive[((ThreadToken)x).No] = true;

                    // work ...
                    PReproduction(((ThreadToken)x).StartIndex,
((ThreadToken)x).Length, ((ThreadToken)x).Rand);

                    // We have finished our job, so release the semaphore
                    isCompleted[((ThreadToken)x).No] = true;
                    sem.Release();
                });
                SetThreadPriority(th[proc]);
                th[proc].Start(tt);
            }


        startloop:
            foreach (var alive in isAlive) // wait parent starter for start all
children.
                if (!alive)
                    goto startloop;


                endLoop:
            sem.WaitOne();
            foreach (var complete in isCompleted) // wait parent to interrupt for
finishes all of children jobs.
                if (!complete)
                    goto endLoop;


            // Continue Parent Work
            sem.Close();
            #endregion
        }
        private void ReproduceByParallelTask()
        {
            #region Parallel Reproduct Code
```

```
            var tasks = new Task[CountCpuCore];


            var length = (PopulationNumber - _nKeep) / CountCpuCore;
            var divideReminder = (PopulationNumber - _nKeep) % CountCpuCore;


            for (var proc = 0; proc < tasks.Length; proc++)
            {
                var tt = new ThreadToken(proc,
                    length + ((proc == CountCpuCore - 1) ? divideReminder : 0),
                    _nKeep + (proc * length));


                tasks[proc] = Task.Factory.StartNew(x =>
                {
                    // work ...
                    PReproduction(((ThreadToken)x).StartIndex,
((ThreadToken)x).Length, ((ThreadToken)x).Rand);

                }, tt, _tokenSource.Token);// TaskCreationOptions.AttachedToParent);
            }


            // When user code that is running in a task creates a task with the
AttachedToParent option,
            // the new task is known as a child task of the originating task,
            // which is known as the parent task.
            // You can use the AttachedToParent option to express structured task
parallelism,
            //  because the parent task implicitly waits for all child tasks to
complete.
            // The following example shows a task that creates one child task:
            Task.WaitAll(tasks);


            // or


            //Block until all tasks complete.
            //Parent.Wait(); // when all task are into a parent task
            #endregion
        }
        /// <summary>
        /// Series Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
        /// </summary>
        /// <param name="rand"></param>
        public void Reproduction(System.Random rand) // Series
        {
            for (var i = _nKeep; i < PopulationNumber; i++)
            {
                //
                // for send and check Father & Mather chromosome
                Chromosome rankFather, rankMather;


                // have a problem (maybe Rank_1() == Rank_2()) then Father == Mather
                // Solve Problem by Loop checker
                do
                {
                    rankFather = Rank(rand);
                    rankMather = Rank(rand);
```

```
                }
                while (rankFather == rankMather);
                //
                // CrossoverHelper
                var child = rankMather.Crossover(rankFather, new System.Random());
                //
                //  run MutationHelper
                //
                child.Mutation(new System.Random());
                //
                // calculate children chromosome fitness
                //
                child.Evaluate();


                Interlocked.Exchange(ref  Genetic.Population[i],  child);  //  atomic
operation between multiple Thread shared
            }
        }
        /// <summary>
        /// Parallel Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
        /// </summary>
        public void PReproduction(int startIndex, int length, System.Random rand) //
Parallel
        {
            for (var i = startIndex; i < (startIndex + length) && i < PopulationNumber;
i++)
            {
                //
                // for send and check Father & Mather chromosome
                Chromosome rankFather, rankMather;


                // have a problem (maybe Rank_1() == Rank_2()) then Father == Mather
                // Solve Problem by Loop checker
                do
                {
                    rankFather = Rank(rand);
                    rankMather = Rank(rand);
                }
                while (rankFather == rankMather);
                //
                // CrossoverHelper
                var child = rankMather.Crossover(rankFather, new System.Random());
                //
                //  run MutationHelper
                //
                child.Mutation(new System.Random());
                //
                // calculate children chromosome fitness
                //
                child.Evaluate();


                Interlocked.Exchange(ref  Genetic.Population[i],  child);  //  atomic
operation between multiple Thread shared
            }
        }


        /// <summary>
        /// Parallel Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
```

```
        /// </summary>
        /// <param name="rand"></param>
        /// <returns></returns>
        public void PReproduction(System.Random rand) // Parallel.For
        {
            Parallel.For(_nKeep, PopulationNumber,
                         new    ParallelOptions()    {    MaxDegreeOfParallelism    =
CountCpuCore, CancellationToken = _tokenSource.Token },
                         (i, loopState) =>
                         {
                             // have a problem (maybe Rank_1() == Rank_2()) then Father
== Mather
                             // Solve Problem by Loop checker
                             Chromosome rankFather, rankMather;
                             do
                             {
                                 Monitor.Enter(rand);
                                 rankFather = Rank(rand);
                                 rankMather = Rank(rand);
                                 Monitor.Exit(rand);
                             }
                             while (rankFather == rankMather);
                             //
                             // CrossoverHelper
                             var   child   =   rankMather.Crossover(rankFather,   new
System.Random());
                             //
                             //  run MutationHelper
                             //
                             child.Mutation(new System.Random());
                             //
                             // calculate children chromosome fitness
                             //
                             child.Evaluate();

                             Interlocked.Exchange(ref  Genetic.Population[i],  child);
// atomic operation between multiple Thread shared

                             if       (_tokenSource.IsCancellationRequested       ||
_tokenSource.Token.IsCancellationRequested)
                             {
                                 loopState.Stop();
                                 loopState.Break();
                                 return;
                             }
                         });
        }


        #endregion


        private void SetCitiesPosition(List<OvalShape> ovalShapeCity)
        {
            Chromosome.CitiesPosition.Clear();
            foreach (var city in ovalShapeCity)
                Chromosome.CitiesPosition.Add(city.Location);
        }


        private void Stop()
        {
            if (_runTime != null)
            {
```

```
                if (_runTime.IsAlive)
                {
                    SetNumPopEnable(true); // Enable population numUpDown
                    UiInvoke(btnStartStop, delegate ()
                    {
                        btnStartStop.Checked = false;
                    });
                    UiInvoke(btnPauseResume, delegate ()
                    {
                        btnPauseResume.Checked = false;
                    });
                    try
                    {
                        if (pGAToolStripMenuItem.Checked)
                        {
                            _tokenSource.Cancel();
                        }
                        _runTime.Abort();
                    }
                    catch { }
                    RefreshTour();
                }
            }
        }


        private int CalcCountOfCpu()
        {
            var numCore = 0;


            #region Find number of Active CPU or CPU core's for this Programs


            var                             affinityDec                             =
Process.GetCurrentProcess().ProcessorAffinity.ToInt64();
            var affinityBin = Convert.ToString(affinityDec, 2); // toBase 2
            foreach (var anyOne in affinityBin.ToCharArray())
                if (anyOne == '1') numCore++;


            #endregion


            //if (numCore > 2) return --numCore;
            return numCore;
        }


        private void SetThreadPriority(Thread th)
        {
            if (th != null)
            {
                if (th.ThreadState != ThreadState.Aborted &&
                   th.ThreadState != ThreadState.AbortRequested &&
                   th.ThreadState != ThreadState.Stopped &&
                   th.ThreadState != ThreadState.StopRequested)
                {
                    switch (Process.GetCurrentProcess().PriorityClass)
                    {
                        case ProcessPriorityClass.AboveNormal:
                            th.Priority = ThreadPriority.AboveNormal;
                            break;
```

```
                        case ProcessPriorityClass.BelowNormal:
                            th.Priority = ThreadPriority.BelowNormal;
                            break;
                        case ProcessPriorityClass.High:
                            th.Priority = ThreadPriority.Highest;
                            break;
                        case ProcessPriorityClass.Idle:
                            th.Priority = ThreadPriority.Lowest;
                            break;
                        case ProcessPriorityClass.Normal:
                            th.Priority = ThreadPriority.Normal;
                            break;
                        case ProcessPriorityClass.RealTime:
                            th.Priority = ThreadPriority.Highest;
                            break;
                    }
                    //
                    // Set Thread Affinity
                    //
                    Thread.BeginThreadAffinity();
                }
            }
        }


        private void SetTimeGraph(double eliteFitness, long generation, bool
fitnessRefreshed)
        {
            var timeLenght = (Environment.TickCount - _startedTick) / 10; // Convert
to MiliSecond
            if (pGAToolStripMenuItem.Checked)
            {
                if (fitnessRefreshed)
                {
                    _pPlistTfg[1].Add(timeLenght, eliteFitness);
                    _pPlistGfg[1].Add(generation, eliteFitness);
                }
                _pPlistTgg[1].Add(timeLenght, generation);
            }
            else
            {
                if (fitnessRefreshed)
                {
                    _pPlistTfg[0].Add(timeLenght, eliteFitness);
                    _pPlistGfg[0].Add(generation, eliteFitness);
                }
                _pPlistTgg[0].Add(timeLenght, generation);
            }
        }


        private void refreshDGV_CityPositions()
        {
            dgvCity.Rows.Clear();


            for (var count = 0; count < OvalShapeCity.Count; count++)
            {
                dgvCity.Rows.Add(new      object[]      {      count      +      1,
OvalShapeCity[count].Location.ToString() });
            }
        }
```

```
        private void create_City(Point e)
        {
            CounterCity++;
            toolStripStatuslblNumCity.Text = CounterCity.ToString();
            var newCity = new OvalShape();
            //
            // newCity
            //
            newCity.BackColor = Color.Red;
            newCity.BackStyle = BackStyle.Opaque;
            newCity.BorderColor = Color.Red;
            newCity.Cursor = Cursors.Hand;
            newCity.Location = new Point(e.X, e.Y);
            newCity.Size = new Size(20, 20);
            newCity.Click += ovalShape_Click;


            OvalShapeCity.Add(newCity);
            ShapeContainerAllCityShape.Shapes.Add(newCity);
        }


        private void RefreshTour()
        {
            try
            {
                Point point1, point0;
                for (var c = 1; c <= CounterCity; c++)
                    try
                    {

//this.shapeContainer_allCityShape.Shapes.Remove(lineShape_Way[c]);
                        RemoveLineShape(LineShapeWay[c]);
                        //
                    }
                    catch { break; }


                for (var c = 1; c < CounterCity; c++)
                {
                    // pop[0] is Elite chromosome or best less Distance ------------
-----------
                    point1                                                          =
OvalShapeCity[Genetic.Population[0].Genome[c]].Location;
                    point0    =    OvalShapeCity[Genetic.Population[0].Genome[c    -
1]].Location;


                    try
                    {
                        var d = new SetPointCallback(SetPoint);
                        BeginInvoke(d, new object[] { c, point0, point1 });
                    }
                    catch { }


                    //this.shapeContainer_allCityShape.Shapes.Add(lineShape_Way[c]);
                    AddLineShape(LineShapeWay[c]);
                    //
                }
                // design line between city 0 & last city
                // pop[0] is Elite chromosome or best less Distance
```

```
                point1  =  OvalShapeCity[Genetic.Population[0].Genome[CounterCity  -
1]].Location;
                point0 = OvalShapeCity[Genetic.Population[0].Genome[0]].Location;


                try
                {
                    var d2 = new SetPointCallback(SetPoint);
                    BeginInvoke(d2, new object[] { 0, point0, point1 });
                }
                catch { }


                //this.shapeContainer_allCityShape.Shapes.Add(lineShape_Way[0]);
                AddLineShape(LineShapeWay[0]);
            }
            catch { }
        }


        private void ovalShape_Click(object sender, EventArgs e)
        {
            CounterCity--;
            OvalShapeCity.Remove((OvalShape)sender);
            ShapeContainerAllCityShape.Shapes.Remove(((OvalShape)sender)); // Remove
Selected Shape
            // Minus 1 as City Number's
            toolStripStatuslblNumCity.Text = CounterCity.ToString();
            //
            // Refresh City Positions List
            refreshDGV_CityPositions();
        }


        private void MainForm_MouseMove(object sender, MouseEventArgs e)
        {
            toolStripStatuslblLocate.Text = "X = " + e.X.ToString() + " ,  Y = " +
e.Y.ToString();
        }


        private void MainForm_MouseClick(object sender, MouseEventArgs e)
        {
            var mPosition = new Point(e.X - 10, e.Y - 10);
            if (mPosition.X > 1 && mPosition.X < Width - 300 && mPosition.Y > 65 &&
mPosition.Y < Height - 85)
            {
                Stop();
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                LineShapeWay.Clear();
                create_City(mPosition);
                //
                // Refresh City Positions List
                refreshDGV_CityPositions();
            }
        }
```

Додаток В (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА ІЗ ЗАСТОСУВАННЯМ ГЕНЕТИЧНОГО АЛГОРИТМУ

Виконав: студент 2 курсу, групи ЗКН-22м
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Гнаповський О.І.
(прізвище та ініціали)

Керівник: к.т.н., ст. викл. каф. КН
Петришин С.І.
(прізвище та ініціали)
«07» 12 2023 р.

Вінниця ВНТУ – 2023 рік

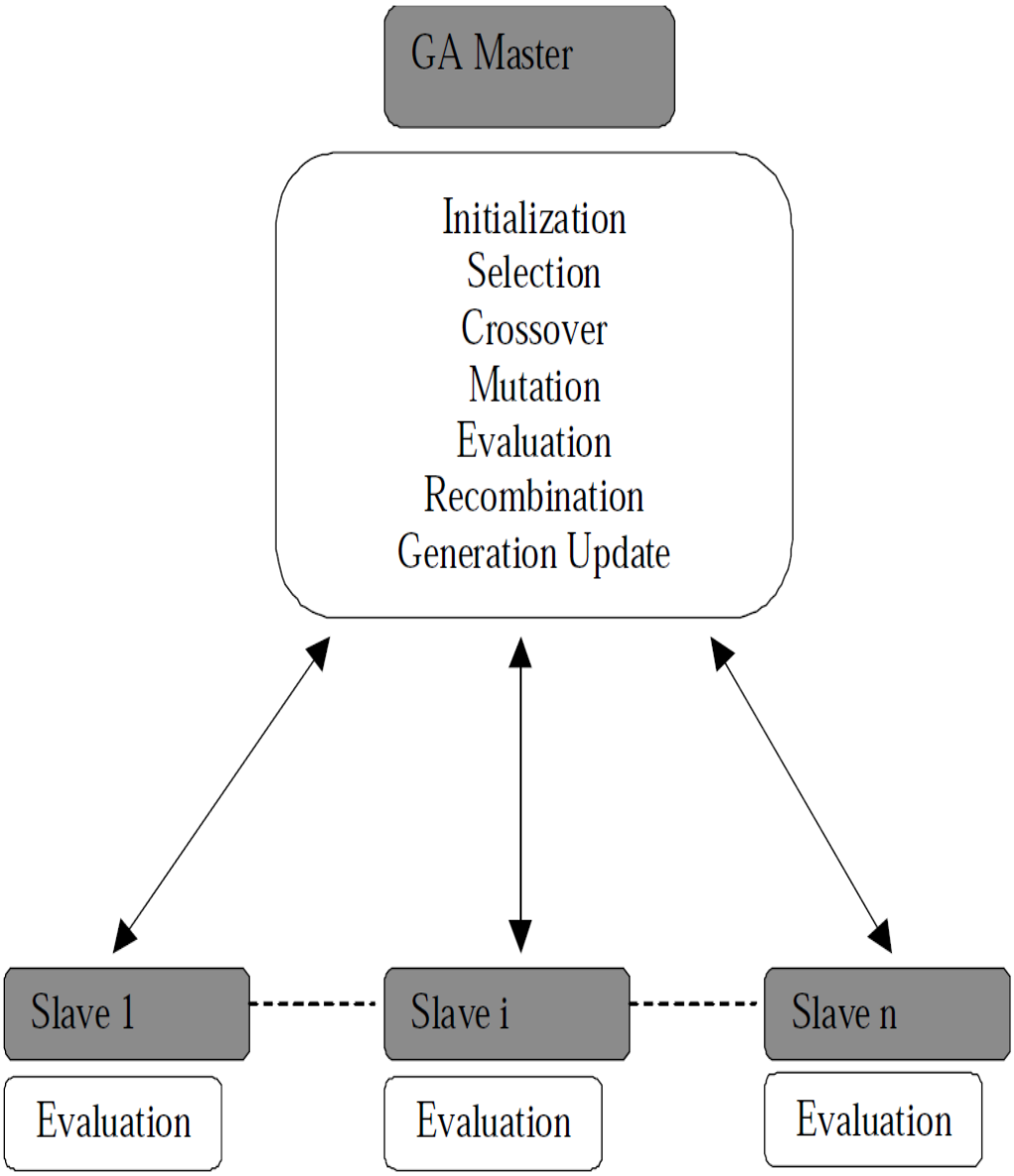


Рисунок В.1 – Структура інформаційної технології розв'язання задачі комівояжера із застосуванням генетичного алгоритму

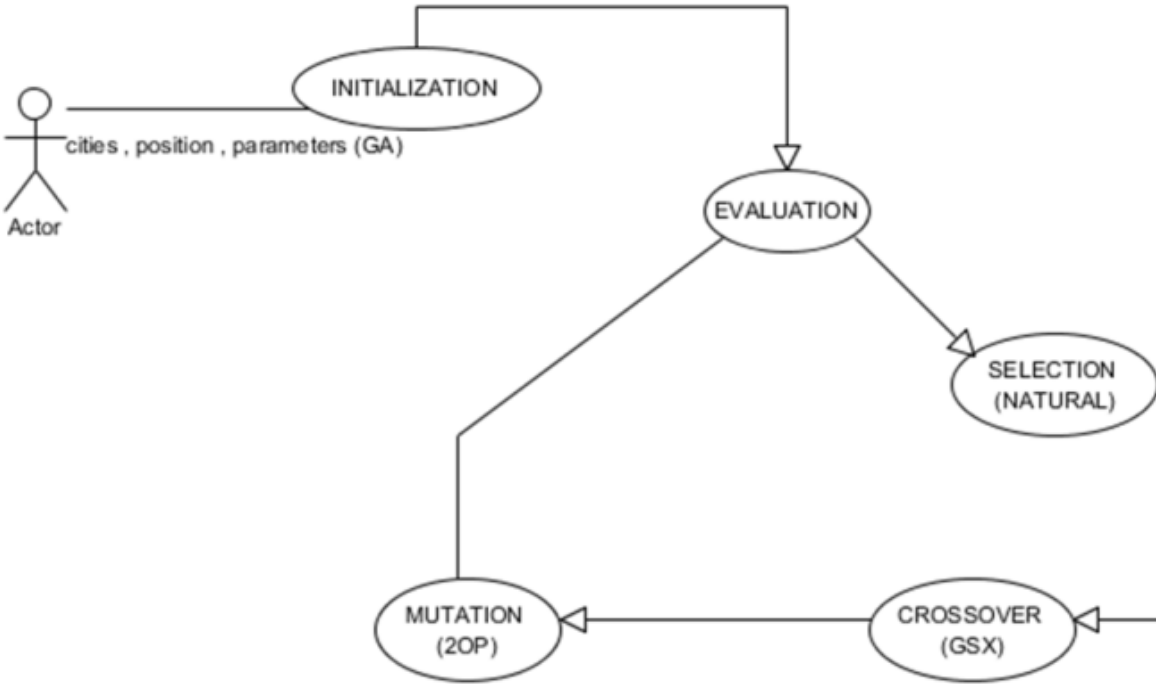


Рисунок В.2 – Діаграма прецедентів

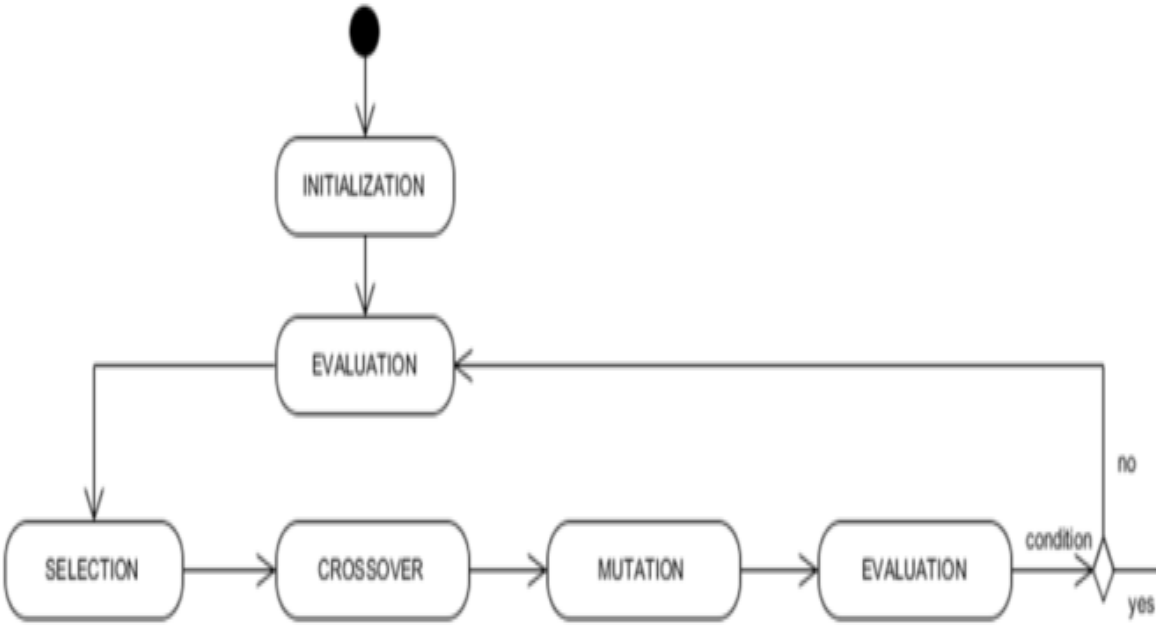


Рисунок В.3 – Діаграма діяльності

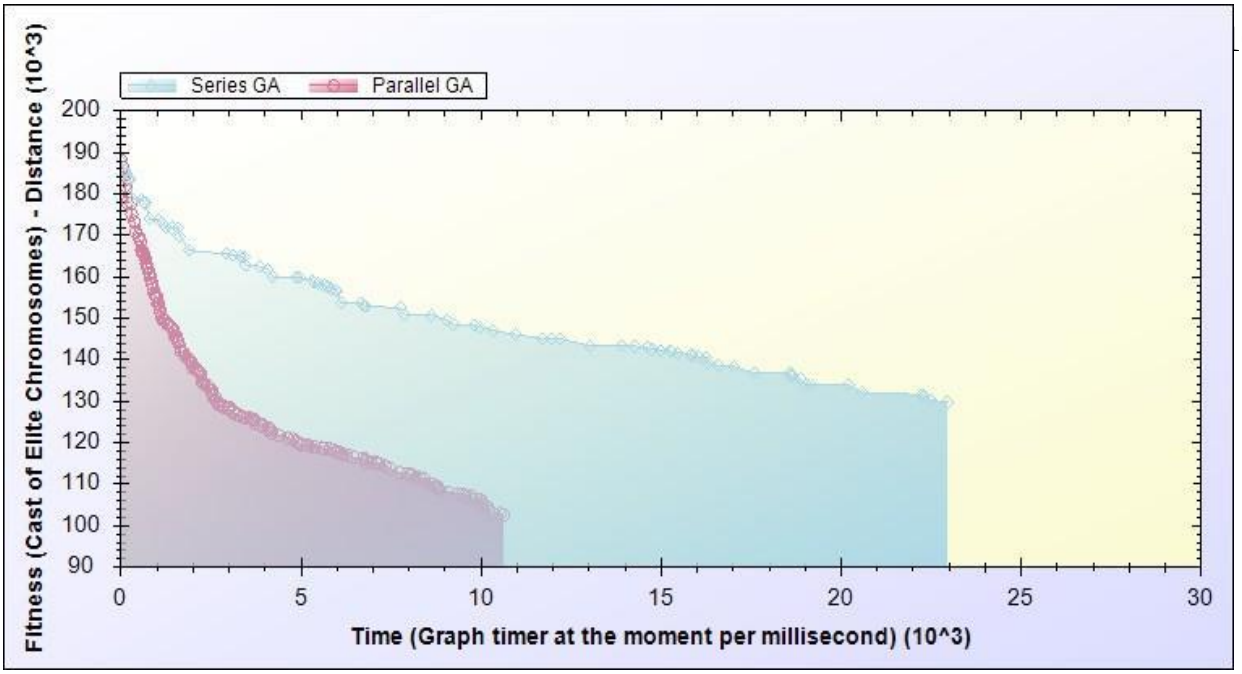


Рисунок В.5 – результат оптимізації побудованого маршруту для розв'язання задачі комівояжера

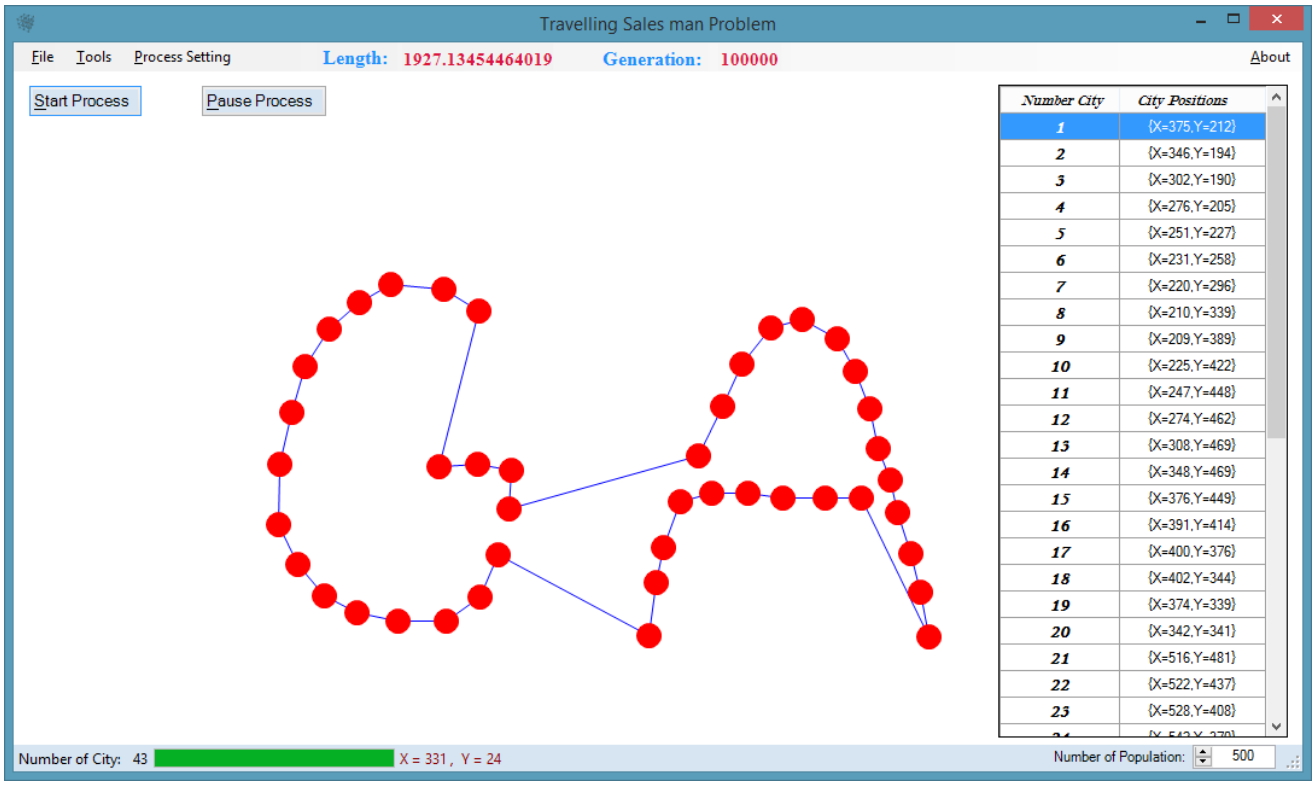


Рисунок В.6 – Скріншот роботи програми

Додаток Г (довідниковий)

Інструкція користувача

Після запуску програми користувач може виконати наступні дії.

Меню "File -> New" викликає процедуру очищення усіх даних - видалення міст та ліній зв'язків, обнулення лічильників тощо. В програмі реалізовані також функції імпорту та експорту даних про розташування міст з текстових файлів.

Окремо присутня можливість генерувати задану кількість випадкових міст з урахуванням мінімальної відстані між ними для уникнення накладання.

Коли користувач натискає на кнопку "Start/Stop", виконується запуск або зупинка роботи генетичного алгоритму в окремому потоці.

При старті виконуються наступні дії:

- блокується можливість зміни розміру популяції;
- очищаються попередні графічні елементи інтерфейсу;
- створюється новий потік і запускається метод генетичного алгоритму Ga().

Аналогічно при паузі/продовженні алгоритму виконується призупинення/ поновлення потоку. Також реалізоване налаштування пріоритету потоку виконання відповідно до обраного значення в меню priority.

Окремо є можливість вибору одного з трьох режимів паралельного генетичного алгоритму (PGA) - за допомогою багатопоточності, завдань або Parallel.For. Зміна режиму супроводжується скиданням відповідних прапорців.

При натисканні на кнопку "Start/Stop" відбувається запуск алгоритму в окремому потоці. Перед запуском виконуються наступні підготовчі кроки:

- Блокується можливість зміни розміру популяції користувачем, щоб уникнути збоїв.
- Видаляються старі графічні елементи інтерфейсу (лінії маршруту).
- Створюється новий список ліній для майбутньої візуалізації результатів.

Кнопка "Pause/Resume" дозволяє призупинити або продовжити виконання алгоритму в потоці (рис. Г.1).

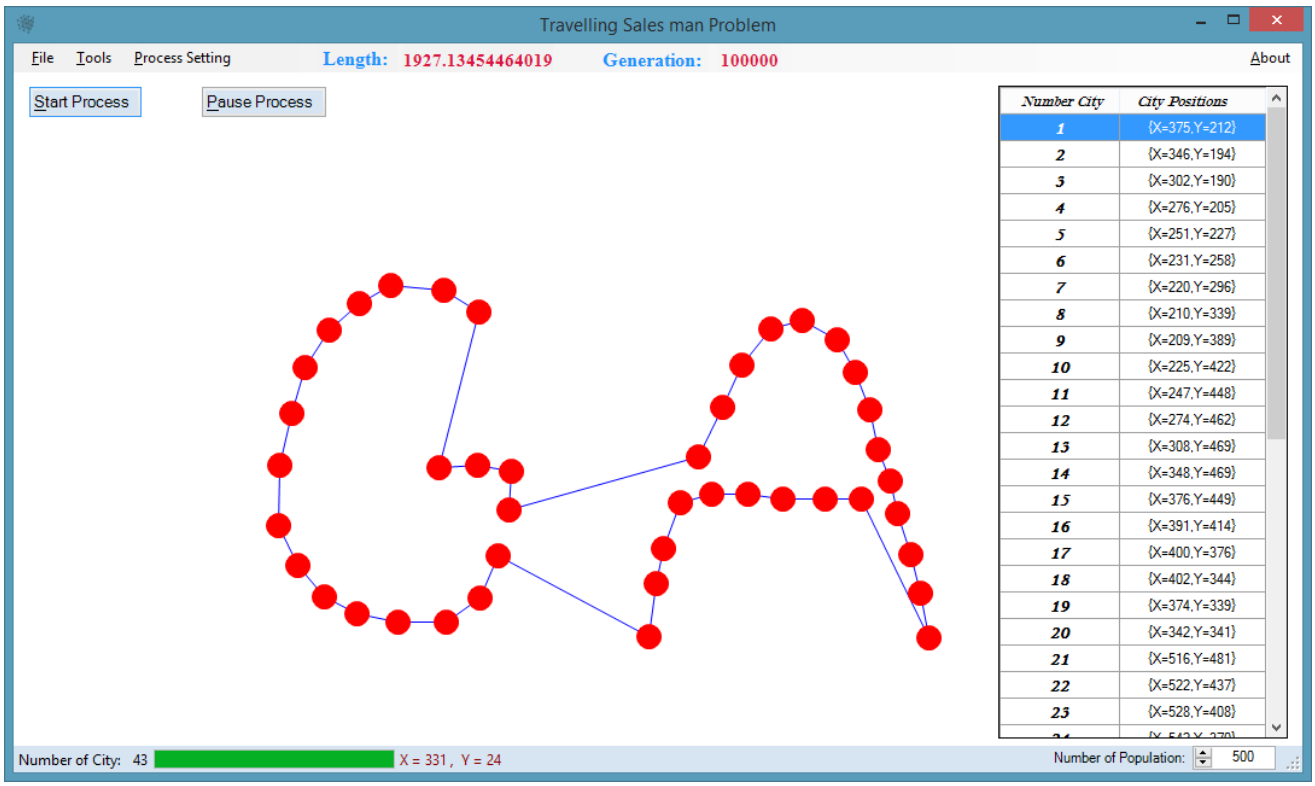


Рисунок Г.1 – Скріншот роботи програми