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))

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

Виконав: студент 4 курсу, групи 4КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Горовий А.В.
(прізвище та ініціали)

Керівник: асистент каф. КН
Ваховська Л.М.
(прізвище та ініціали)
«04» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ
Барабан М.В.
(прізвище та ініціали)
«05» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
"05" травня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Горовому Андрію Валерійовичу

1. Тема роботи: Розробка програмного модуля задачі комівояжера.

Керівник роботи: Ваховська Любов Михайлівна, асистент.

Затверджені наказом вищого навчального закладу "20" березня 2025 року № 97

2. Термін подання студентом роботи 30 травня 2025 р.

3. Вихідні дані до роботи:

ОС Windows; об'єктно-орієнтована мова програмування C#;
кількість міст; кількість популяцій ГА.

4. Зміст текстової частини (перелік питань, які потрібно розробити):

Обґрунтування доцільності розробки програмного модуля задачі комівояжера;

Розробка програмного модуля задачі комівояжера;
Програмна реалізація модуля задачі комівояжера.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
Структурна схема реалізації генетичного алгоритму задачі комівояжера;
Діаграма прецедентів;
Діаграма послідовності;
Головне вікно програмного модуля.
.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1 – 3	Ваховська Л.М., асистент		

7. Дата видачі завдання 05 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обґрунтування доцільності розробки програмного модуля задачі комівояжера	05.05.25	07.05.25	
2	Розробка програмного модуля задачі комівояжера	08.05.25	17.05.25	
3	Програмна реалізація модуля задачі комівояжера	18.05.25	26.05.25	
4	Висновки	27.05.25	28.05.25	
5	Оформлення додатків	29.05.25	30.05.25	
6	Розробка інструкції користувача	01.06.25	02.06.25	
7	Оформлення матеріалів до захисту БКР	03.06.25	04.06.25	

Студент _____ Горовий А.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ваховська Л.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 104 сторінок формату А4, на яких є 16 рисунків, список використаних джерел містить 21 найменування.

У даній роботі розглядається розробка програмного модуля для вирішення задачі комівояжера, зокрема, досліджено різні методи та алгоритми для оптимізації маршруту. Проведено аналіз існуючих програмних рішень, визначено їх переваги та недоліки. Описано етапи проектування, розробки та тестування програмного модуля, а також здійснено порівняння ефективності новозапропонованого рішення з іншими доступними інструментами. Розроблений модуль продемонстрував високі результати при розв’язуванні задач різної складності.

Ключові слова: задача комівояжера, маршрут, генетичний алгоритм.

ABSTRACT

The bachelor's thesis consists of 104 A4 pages, containing 16 figures, and the list of references contains 21 titles.

This paper focuses on the development of a software module for solving the traveling salesman problem, specifically examining various methods and algorithms for route optimization. An analysis of existing software solutions has been conducted, highlighting their advantages and disadvantages. The stages of designing, developing, and testing the software module are described, along with a comparison of the performance of the proposed solution against other available tools. The developed module demonstrated high efficiency when solving problems of varying complexity.

Keywords: traveling salesman problem, route, genetic algorithm.

ЗМІСТ

ВСТУП

Актуальність теми. Розробка програмного модуля для розв’язання задачі комівояжера є актуальною через її широке застосування в оптимізаційних задачах логістики, транспорту, планування маршрутів та інших галузях, що потребують мінімізації витрат і часу на пересування між визначеними пунктами.

Задача комівояжера належить до класу NP-складних задач [1], для яких відсутні поліноміальні алгоритми точного розв’язку. Це обумовлює необхідність розробки ефективних евристичних, метаевристичних та апроксимаційних алгоритмів, що забезпечують знаходження субоптимальних рішень за прийнятний час.

У сучасних умовах цифровізації та автоматизації бізнес-процесів значення подібних модулів зростає, оскільки вони знаходять застосування в транспортній логістиці, кур’єрських службах, виробничих системах та навіть у сфері біоінформатики. Створення програмного модуля, що використовує сучасні алгоритми, такі як генетичні алгоритми, методи рою частинок або алгоритм мурашиних колоній, дозволяє підвищити ефективність розв’язку задач маршрутизації та зменшити витрати на логістичні операції.

Метою дослідження є підвищення зручності графічного інтерфейсу користувача при розв’язання задачі комівояжера з використанням сучасних оптимізаційних методів, що забезпечують ефективне знаходження субоптимальних маршрутів при мінімальних витратах часу та обчислювальних ресурсів.

Також можна виділити наступні задачі дослідження:

1. Проаналізувати існуючі методи та алгоритми розв’язання задачі комівояжера, їх переваги та обмеження.
2. Розглянути можливості застосування евристичних та метаевристичних підходів для ефективного розв’язку задачі.
3. Розробити програмний модуль, що реалізує обрані алгоритми для знаходження оптимального маршруту.

4. Провести тестування та порівняльний аналіз ефективності запропонованих алгоритмів.

5. Визначити сфери застосування розробленого модуля та оцінити його практичну значущість у реальних задачах маршрутизації.

Об’єкт дослідження – процес побудови оптимального маршруту між множиною заданих точок у задачі комівояжера.

Предмет дослідження – програмні засоби реалізації модуля задачі комівояжера.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

1.1 Аналіз сучасних досліджень в області задачі комівояжера

Задача комівояжера (англ. Travelling Salesman Problem (TSP)) є однією з найвідоміших та найбільш досліджуваних задач комбіторної оптимізації. Вона полягає у знаходженні найкоротшого маршруту, який проходить через задану множину міст, відвідуючи кожне лише один раз і повертаючись у вихідну точку [1]. Незважаючи на уявну простоту формулювання, задача належить до класу NP-складних, що зумовлює значні труднощі при розв’язанні великих її екземплярів.

У сучасних наукових дослідженнях простежується активна увага до розробки та вдосконалення алгоритмів, здатних ефективно розв’язувати TSP у прийнятні строки. Традиційні точні методи, зокрема повний перебір, метод гілок і меж (branch and bound), динамічне програмування (алгоритм Гельд-Карпа), хоча й гарантують знаходження оптимального рішення, мають обмежену практичну придатність через експоненційну складність [2, 3].

Натомість все більшого поширення набувають евристичні та метаевристичні методи. Серед евристик слід виокремити алгоритми найближчого сусіда, жадібні методи та інсерційні стратегії, що забезпечують швидкий розрахунок наближеного рішення, проте часто не гарантують високої точності [4]. Більш перспективними є метаевристичні підходи, зокрема генетичні алгоритми [5], алгоритми рою частинок [6], мурашині алгоритми (Ant Colony Optimization) [7], табу-пошук та їх гібридні комбінації [5]. Ці методи демонструють добру збіжність до оптимального або близького до нього результату, особливо в задачах великої розмірності.

Важливим напрямом є також використання гібридних підходів, які поєднують класичні евристики з елементами машинного навчання, що дозволяє покращити адаптивність алгоритмів до конкретних типів графів або

обмежень. Зокрема, дослідники застосовують нейронні мережі, включаючи трансформерні архітектури та графові нейронні мережі, для наближеного передбачення маршрутів, а також reinforcement learning для навчання оптимізаційних стратегій [3].

Значна частина робіт також фокусується на варіаціях задачі комівояжера, таких як симетрична і несиметрична TSP, TSP з часовими вікнами, стохастична TSP, багатокомівояжерна задача (mTSP), а також TSP з обмеженими ресурсами [2]. У контексті практичних застосувань, TSP використовується у транспортній логістиці, автоматичному плануванні, розробці схем інтегральних мікросхем, обслуговуванні дронів, робототехніці та біоінформатиці.

Класична задача комівояжера передбачає знаходження найкоротшого маршруту, що проходить через усі задані міста один раз із поверненням до початкової точки. Проте, у процесі практичного застосування виникає потреба в урахуванні додаткових умов і обмежень, що призводить до формування численних варіацій задачі комівояжера. Ці варіації відображають складність реального світу та роблять модель більш адаптованою до прикладних задач у логістиці, робототехніці, виробництві, обслуговуванні та інших сферах.

Однією з найпоширеніших є асиметрична задача комівояжера (ATSP) [8, 9], в якій відстань (або вартість переміщення) з пункту *A* до пункту *B* може відрізнятися від відстані з *B* до *A*. Така постановка є характерною для дорожніх мереж з одностороннім рухом, різними дорожніми умовами або транспортними потоками. Асиметричність значно ускладнює задачу і потребує адаптації алгоритмів.

Інша важлива варіація – задача комівояжера з часовими вікнами (TSP-TW) [9]. У цій моделі кожен вузол (місто) пов'язаний з часовим інтервалом, протягом якого його можна відвідати. Ця варіація надзвичайно актуальна для кур’єрських служб і доставки товарів, коли клієнти доступні лише у визначені проміжки часу. Завдання ускладнюється необхідністю врахування часу обслуговування і можливості очікування.

Стохастична задача комівояжера (STSP) враховує випадкові фактори, такі як непередбачувані зміни відстаней, часу доставки або наявність міст у маршруті [10]. Вона застосовується в умовах невизначеності та неповної інформації, наприклад, в управлінні маршрутами при змінних погодних умовах або динамічному попиті.

Багатокомівояжерна задача (mTSP) моделює ситуацію, коли маршрути мають бути побудовані не для одного, а для декількох агентів. Такий підхід часто використовується в логістиці для оптимізації роботи автопарку або команд сервісного обслуговування. Важливими підзадачами є балансування навантаження між агентами та мінімізація загальної вартості маршрутів.

TSP з обмеженими ресурсами (TSP-RC) враховує обмеження на ресурси, наприклад, запас пального, вантажопідйомність, кількість зупинок тощо. Вона має особливе значення в транспортному плануванні та автоматизованих системах управління мобільними роботами.

Ще однією цікавою модифікацією є TSP з пріоритетами (PTSP), у якій деякі пункти мають вищу важливість і повинні бути відвідані раніше або з мінімальним відхиленням від ідеального часу. Це особливо важливо у медичній логістиці, екстрених перевезеннях та VIP-обслуговуванні.

У контексті високих обчислювальних витрат, також досліджується ієрархічна або багаторівнева задача комівояжера, де спочатку оптимізується маршрут для регіональних центрів, а потім — для локальних. Це дозволяє зменшити обчислювальну складність і більш ефективно масштабувати розв’язки на великі території.

Окрім цього, в сучасних дослідженнях розглядаються варіанти задачі комівояжера на графах з обмеженнями: TSP на графах із перешкодами, TSP з альтернативними маршрутами, TSP на динамічних або змінних графах, а також задачі, пов’язані з частковим покриттям міст або TSP з обов’язковими та необов’язковими точками відвідування.

1.2 Аналіз відомих програм аналогів задачі комівояжера

Проаналізуємо відомі програми-аналоги задачі комівояжера та визначимо їх ключові особливості, переваги та недоліки.

Concorde TSP Solver є однією з найпотужніших і найточніших програм для розв’язання симетричної задачі комівояжера. Вона реалізує комбінацію точних методів, зокрема алгоритм гілок і меж (branch-and-bound), метод січних площин (cutting planes) та поліноміальні евристики. Програма здатна обробляти графи з кількістю вершин до 85 000, що робить її придатною для наукових задач великого масштабу. Основною перевагою Concorde є її здатність гарантувати глобально оптимальний результат у випадку класичної TSP. Серед інших плюсів – активна підтримка, наявність бібліотек для інтеграції в інші програми, використання форматів даних TSPLIB. Проте серед недоліків слід відзначити відносно високу обчислювальну вартість на великих даних, а також складність у налаштуванні для модифікованих варіантів задачі (наприклад, з часовими вікнами чи ресурсними обмеженнями). На рисунку 1.1 можна побачити інтерфейс даного додатку.

LKH (Lin-Kernighan-Helsgaun) – це вдосконалена реалізація одного з найвідоміших евристичних алгоритмів Ліна-Кернігана, що адаптована для дуже швидкого та ефективного наближеного розв’язання задачі комівояжера [6]. Розроблена К’єльом Хельсгауном, LKH демонструє високу точність та обчислювальну ефективність, часто знаходячи рішення, які на практиці дуже близькі до оптимальних. Основною перевагою алгоритму є його здатність обробляти великі графи за порівняно короткий час, що робить LKH ідеальним інструментом для задач, де потрібна швидка реакція або обмежені ресурси. LKH також підтримує асиметричні задачі, багатоекземплярні задачі (мTSP), задачі з часовими вікнами та інші варіації. Проте недоліком є відсутність гарантії отримання оптимального розв’язку, а також залежність ефективності від правильного налаштування параметрів алгоритму. На рисунку 1.2 можна побачити інтерфейс даного додатку.

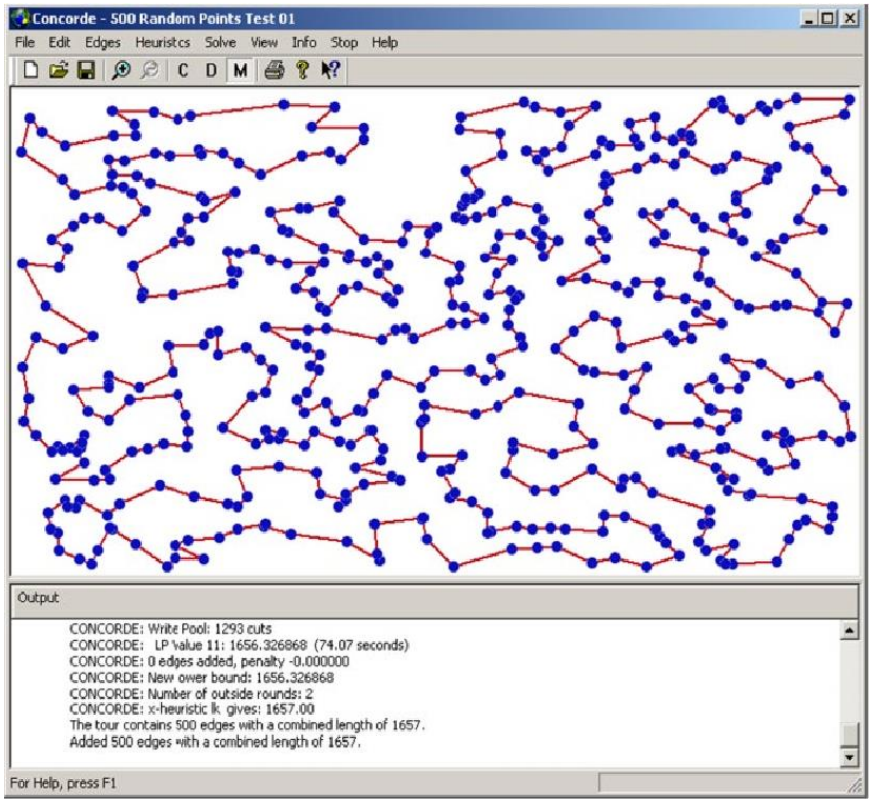


Рисунок 1.1 – Інтерфейс додатку Concorde TSP Solver

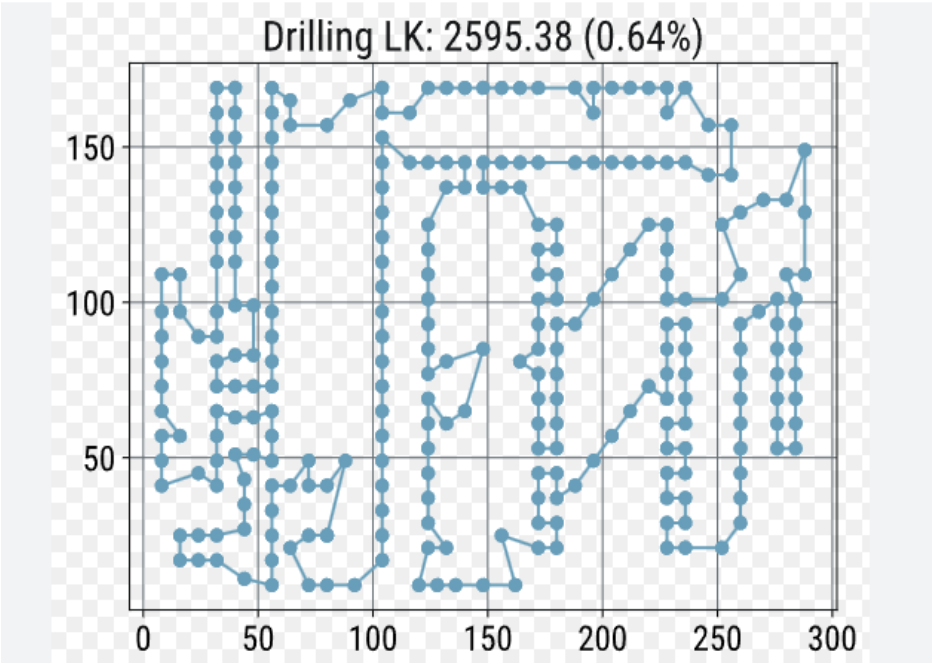


Рисунок 1.2 – Інтерфейс додатку LKH

Ant Colony Optimization (ACO) є класом метаевристичних алгоритмів, натхнених поведінкою реальних мурах, що використовують феромонні сліди для знаходження найкоротших шляхів між джерелами їжі. У контексті TSP, ACO забезпечує ефективний розв’язок завдяки колективному пошуку рішень, де кожна «мураха» формує маршрут, орієнтуючись на феромонні мітки та евристичну інформацію (наприклад, відстані). ACO особливо ефективний при розв’язанні складних варіацій задачі комівояжера, зокрема таких, що включають часові вікна, динамічні зміни графа чи обмеження на ресурси. Перевагою алгоритму є його адаптивність, гнучкість і здатність працювати у багатьох модифікаціях задачі без суттєвих змін у структурі коду. Проте серед недоліків варто відзначити повільнішу збіжність у порівнянні з LKH, чутливість до вибору параметрів (інтенсивність феромону, коефіцієнт випаровування тощо), а також можливість застрягання в локальних мінімумах. На рисунку 1.3 можна побачити інтерфейс даного додатку.

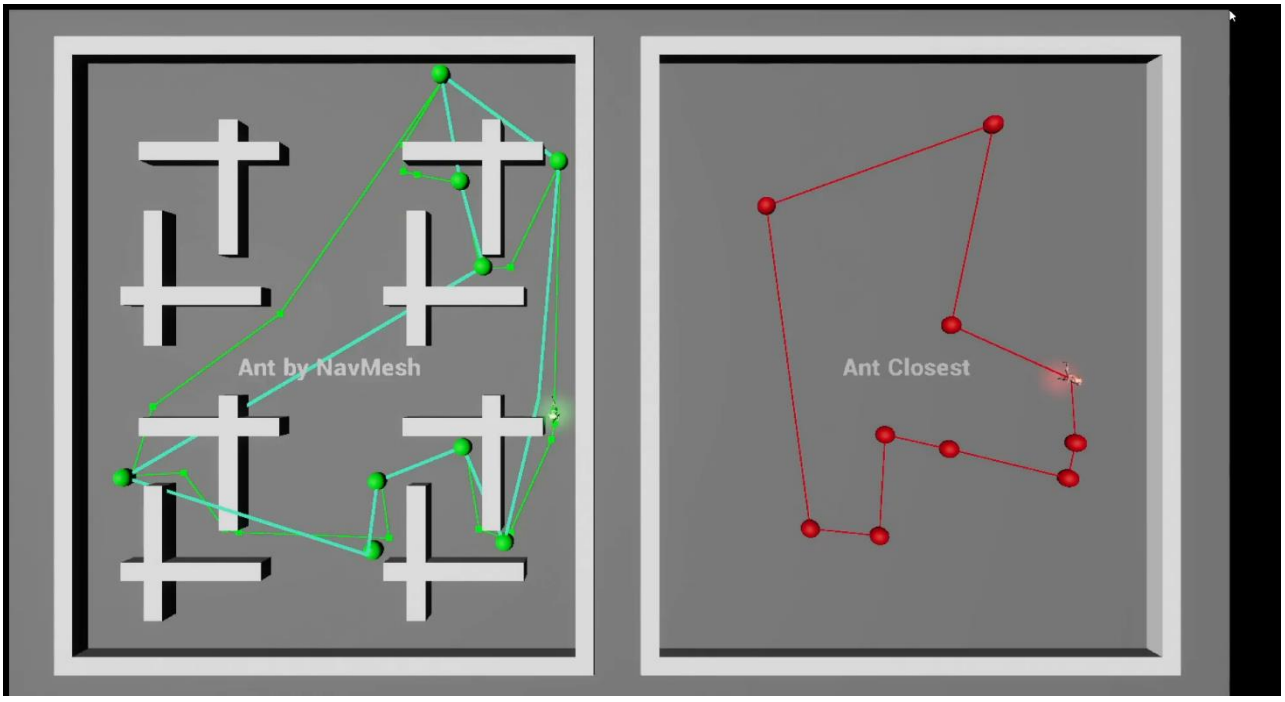


Рисунок 1.3 – Інтерфейс додатку Ant Colony Optimization

В таблиці 1.1 можна побачити порівняльну характеристику усіх вище перерахованих додатків.

Таблиця 1.1 – Порівняльна характеристика програм аналогів

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 Постановка задачі

Розробити програмний модуль, що вирішує задачу комівояжера за допомогою генетичного алгоритму (GA) та забезпечує графічну візуалізацію процесу оптимізації. Програма повинна дозволяти змінювати параметри алгоритму, відображати найкраще знайдене рішення на поточному етапі та вести статистику результатів.

Тому в реальних умовах часто застосовують евристичні та метаевристичні методи, які дозволяють знаходити наближені рішення з високою якістю за розумний час. Одним з таких методів є генетичний алгоритм (ГА), що базується на механізмах природного добору, кросоверу та мутації — імітуючи еволюційні процеси для пошуку оптимального розв’язку.

Через обчислювальну складність задачі, особливо при зростанні розмірності, доцільним є використання евристичних та метаевристичних підходів, які дозволяють знаходити наближені рішення високої якості. Одним із таких методів є генетичний алгоритм — стохастичний алгоритм пошуку, натхненний природною еволюцією, що базується на принципах відбору, схрещування та мутації.

Вхідні дані:

– Кількість міст та їх координати (у 2D-просторі).

– Початкові параметри алгоритму: розмір популяції, ймовірність мутації, кількість поколінь.

– Додаткові параметри (тип кросоверу, метод селекції тощо).

Вихідні дані:

– Найкращий знайдений маршрут.

– Візуальне представлення маршруту.

– Довжина найкращого маршруту.

– Динаміка зміни найкращого рішення за поколіннями.

Винесемо на виконання наступні завдання розробки:

1. Алгоритмічне забезпечення

1.1. Реалізувати генетичний алгоритм для знаходження наближеного розв’язку задачі комівояжера.

1.2. Вибрати метод представлення особин (перестановка послідовності міст).

1.3. Реалізувати ініціалізацію популяції випадковими маршрутами.

1.4. Впровадити оператори кросоверу, мутації та селекції.

1.5. Оптимізувати параметри алгоритму для підвищення ефективності.

1.6. Передбачити можливість розширення алгоритму для застосування альтернативних евристик, таких як алгоритм мурашиної колонії (ACO) або локальний пошук (2-opt).

2. Графічний інтерфейс користувача (GUI)

2.1. Розробити інтуїтивно зрозумілий графічний інтерфейс.

2.2. Відобразити початкове розташування міст.

2.3. Реалізувати динамічне оновлення поточного найкращого маршруту.

2.4. Візуалізувати зміну довжини маршруту на графіку.

2.5. Передбачити можливість зміни параметрів алгоритму та повторного запуску.

3. Валідація та тестування

3.1. Перевірити коректність роботи алгоритму на класичних тестових наборах (TSPLIB).

3.2. Дослідити вплив параметрів алгоритму на якість рішення (розмір популяції, типи кросоверу та мутації, швидкість збіжності).

3.3. Оцінити продуктивність алгоритму при збільшенні кількості міст.

Таким чином, метою є створення інструменту, здатного застосовувати еволюційний підхід до ефективного розв’язання задачі комівояжера, з можливістю подальшого розширення на інші задачі маршрутизації або оптимізації.

1.4 Висновок до розділу

У першому розділі було проведено аналіз предметної області задачі комівояжера, розглянуто її варіації та алгоритмічні підходи до розв’язання. Було визначено, що задача належить до класу NP-складних задач, що ускладнює її точне розв’язання для великих наборів даних. Розглянуто методи знаходження наближених рішень, зокрема генетичний алгоритм, алгоритм мурашиної колонії та локальні пошукові евристики.

Також проаналізовано сучасні програмні рішення, такі як Concorde TSP Solver, LKH (Lin-Kernighan-Helsgaun) та Ant Colony Optimization, що демонструють ефективність у різних варіантах задачі комівояжера. Визначено їхні переваги та недоліки, що дозволяє зробити обґрунтований вибір методів для розробки програмного модуля.

Результати аналізу підтверджують доцільність використання генетичного алгоритму в поєднанні з додатковими евристиками, що забезпечить ефективний пошук оптимального маршруту для задачі комівояжера, а також на основі цих данних поставлені задачі для розробки програмного модуля задачі комівояжера.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

2.1 Проектування структурної схеми роботи програмного модуля

Генетичні алгоритми (ГА) – це евристичний метод пошуку та оптимізації, заснований на принципах природної еволюції. В основі лежить ідея виживання найкращих: ті рішення, які є більш придатними, мають більшу ймовірність впливати на формування наступного покоління. ГА є потужним інструментом для задач, де класичні методи (градієнтні, динамічне програмування, перебір) виявляються малоефективними або взагалі непридатними [11, 12].

У загальному випадку ГА дозволяє знаходити наближені розв’язки задач у великих просторах пошуку, де кількість можливих рішень зростає експоненційно.

Алгоритм проходить кілька стандартних етапів:

Ініціалізація популяції

На початковому етапі створюється початкова популяція з PPP особин (рішень), кожна з яких є допустимим (але не обов’язково оптимальним) розв’язком задачі. У випадку задачі комівояжера, кожен індивід — це перестановка номерів міст

Оцінювання особин (функція пристосованості)

Кожен індивід оцінюється за функцією пристосованості (fitness function). У випадку TSP – чим коротший шлях, тим краще.

Відбір (селекція)

На цьому етапі відбираються особини-батьки, які дадуть потомство. Найчастіше використовуються такі методи:

- Рулетка (roulette wheel) – ймовірність вибору пропорційна значенню fitness.
- Турнірний відбір (tournament selection) – випадково вибираються k особин, серед яких обирається краща.

Схрещування (кросовер)

Основна мета – передати «гени» (частини рішень) від батьків до потомства [12]. Для задачі перестановок (як у TSP) не можна застосовувати звичайний одноточковий кросовер, тому використовуються спеціалізовані оператори:

Order Crossover (OX)

1. Випадково вибирається піддіапазон у першому з батьків.
2. У другому з батьків копіюються елементи, що залишилися, у порядку, який зберігає відносну позицію.

Інші оператори:

- PMX (Partially Matched Crossover)
- CX (Cycle Crossover)

Мутація

Щоб уникнути передчасної конвергенції (застрягання в локальних мінімумах), застосовується мутація – випадкова зміна особин. У TSP це можуть бути:

- Swap Mutation – обмін двох міст.
- Inversion Mutation – інверсія підпослідовності.
- Scramble Mutation – перемішування декількох міст у піддіапазоні.

Приклад мутації:

Before: [1, 2, 3, 4, 5]

After (swap 2 & 4): [1, 4, 3, 2, 5]

Формування нового покоління

Після кросоверу і мутації нові особини формують нову популяцію. Існує два підходи:

- Повна заміна – все нове покоління.
- Елітарність – найкращі особини поточного покоління зберігаються для наступного.

Умови завершення

Генетичний алгоритм завершується, коли:

- досягнуто максимальну кількість поколінь;
- протягом певної кількості поколінь не покращується результат;
- досягнуто заздалегідь заданий рівень пристосованості.

Адаптація ГА до задачі комівояжера

Через природу задачі комівояжера (перестановка міст) традиційні ГА не можуть бути застосовані напряму [13].

Потрібно враховувати наступні аспекти наведені в таблиці 2.1:

Таблиця 2.1 – ключові аспекти для ГА [14].

Компонент	**Реалізація для TSP**
Представлення хромосоми	Перестановка міст
Функція пристосованості	f(x)=1/загальна довжина маршрутуf(x)=text{загальна довжина маршруту}
Кросовер	OX, PMX, CX
Мутація	Swap, Inversion, Scramble
Відбір	Турнірний або рулетка

Переваги використання ГА для TSP

- Пошук відбувається паралельно, що дозволяє досліджувати простір розв’язків більш ефективно.
- Можливість адаптації під обмеження (наприклад, часові або транспортні).
- Легка масштабованість: добре працює для кількох десятків або сотень міст.
- Гнучкість: можливість комбінувати з іншими методами (наприклад,

локальним пошуком чи жадібними алгоритмами - *hybrid GA*).

Недоліки та виклики

- Велика залежність від вибору параметрів (ймовірності мутації, розмір популяції тощо).
- Потрібно забезпечити збереження коректності рішень (щоб не виникли повтори міст).
- Гарантії глобального оптимуму відсутні (але можна отримати *дуже наближене* рішення).

Метою є мінімізація загальної довжини маршруту, тобто суми відстаней між усіма послідовними містами у хромосомі, включаючи повернення до початкового міста.

Генерується P хромосом випадковим чином, з урахуванням, що кожна місто має бути використане лише один раз (тобто це перестановка без повторів).

Формування нового покоління

Після відбору, схрещування і мутації формується нова популяція, деякі алгоритми зберігають найкращих представників (елітність), наприклад:

Зберігаємо 5% найкращих з попереднього покоління без змін.

Це підвищує стабільність і запобігає втраті хороших рішень.

Алгоритм завершується, коли:

- Досягнута максимальна кількість поколінь (наприклад, 500–1000).
- Не відбувається покращення кращого рішення протягом k поколінь.
- Досягнуто рішення, яке задовольняє користувача (наприклад, фіксована довжина маршруту).

Тому псевдокод данного алгоритму зображений на рисунку 2.1 і рисунку 2.2.

А блок-схема генетичного алгоритму для задачі комівояжера представлена на рисунку 2.3.

```
ГЕНЕТИЧНИЙ_АЛГОРИТМ_TSP(міста, max_поколінь, pop_розмір, p_кросовер, p_мутації)

1. Створити початкову популяцію:
    Для i = 1 до pop_розмір:
        - Створити випадкову перестановку міст як хромосому
        - Додати хромосому до популяції

2. Для покоління = 1 до max_поколінь:

    2.1. Оцінити популяцію:
        Для кожної хромосоми:
            - Обчислити довжину маршруту
            - Обчислити fitness = 1 / довжина

    2.2. Вибрати нову популяцію:
        - Використати турнірну селекцію або рулетку
        - Вибрати батьківські пари для схрещування
```

Рисунок 2.1 – Псевдокод генетичного алгоритму для задачі комівояжера

```
    2.3. Схрещування:
        Для кожної пари (Parent1, Parent2):
            Якщо випадкове_значення < p_кросовер:
                - Застосувати OX або PMX → створити Child1, Child2
            Інакше:
                - Child1 = Parent1, Child2 = Parent2
            - Додати дітей до нової популяції

    2.4. Мутація:
        Для кожної хромосоми в новій популяції:
            Якщо випадкове_значення < p_мутації:
                - Виконати одну з мутацій: swap, inversion або scramble

    2.5. Елітизм (опційно):
        - Скопіювати найкращі N особин з попереднього покоління

    2.6. Оновити популяцію

3. Повернути найкращу знайдену хромосому як оптимальний маршрут
```

Рисунок 2.2 – Псевдокод генетичного алгоритму для задачі комівояжера

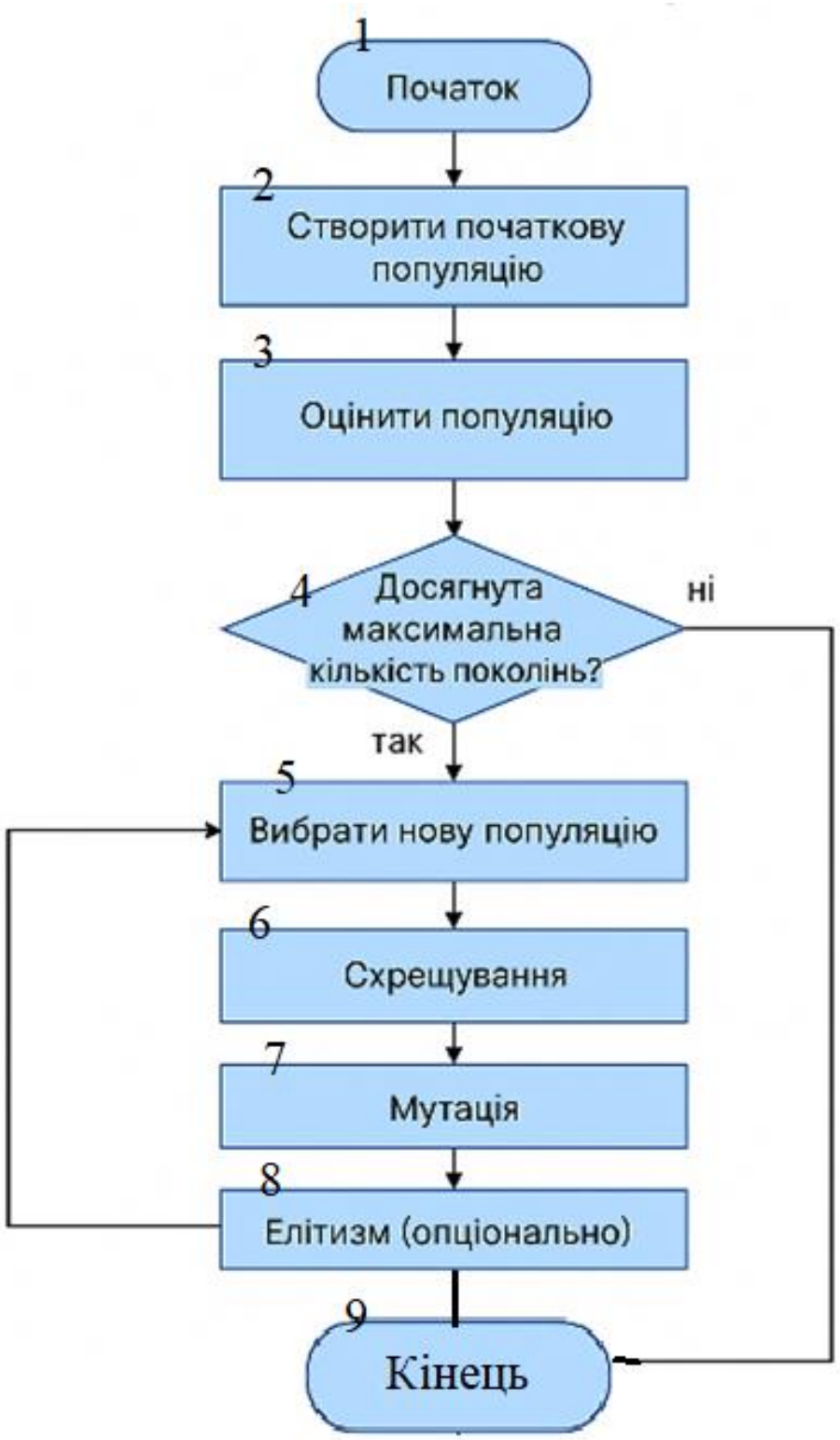


Рисунок 2.3 – Блок-схема генетичного алгоритму для задачі комівояжера

Розроблена структурна схема ілюструє паралельну реалізацію генетичного алгоритму для розв'язання задачі комівояжера. На вершині ієрархії знаходиться компонент «Менеджер», який керує усім процесом еволюційного пошуку. Менеджер виконує ініціалізацію початкової популяції, створюючи набір хромосом, що представляють потенційні маршрути комівояжера [14-16]. Після ініціалізації менеджер зберігає популяцію для подальшої обробки та керує процесом еволюції протягом усіх поколінь (рис. 2.4).

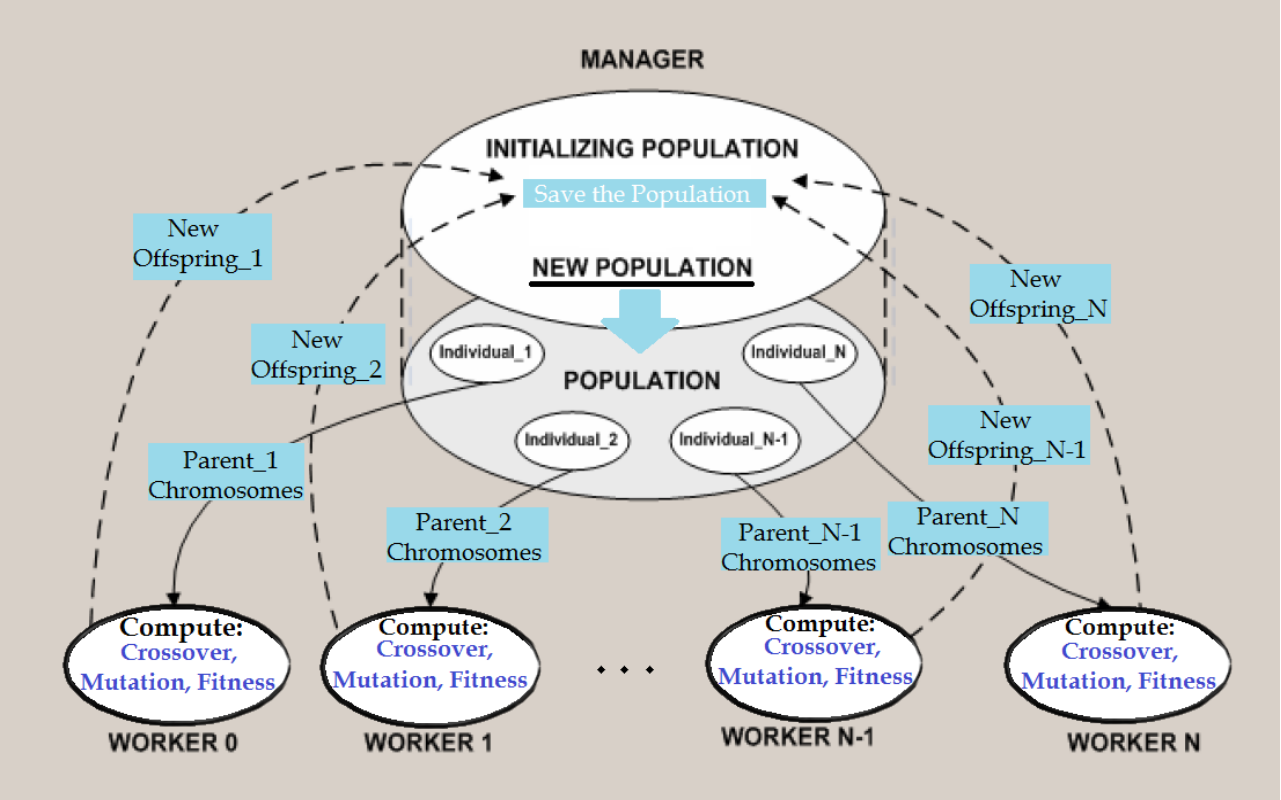


Рисунок 2.4 – Структурна схема реалізації алгоритму розв'язання задачі комівояжера з використанням концепції генетичних алгоритмів [11].

Популяція в даній архітектурі складається з N особин, кожна з яких представляє окремий маршрут комівояжера. У контексті задачі комівояжера кожна особина кодує певну послідовність відвідування міст. Дані особини розташовані в центральній частині схеми і формують основу обчислювального процесу. Популяція постійно еволюціонує протягом роботи алгоритму, в процесі чого маршрути з нижчою загальною відстанню мають більші шанси

на відтворення.

У нижній частині схеми розташовані паралельні обчислювальні вузли, названі «Робітниками». Дана архітектура передбачає наявність N+1 робітника (від Робітника 0 до Робітника N), кожен з яких незалежно виконує основні генетичні операції: кросовер, мутацію та оцінку пристосованості. Кросовер передбачає комбінування генетичного матеріалу двох батьківських хромосом для створення нащадків. У задачі комівояжера використовуються спеціальні оператори кросоверу, які зберігають валідність маршрутів, тобто гарантують, що кожне місто відвідується рівно один раз. Мутація вносить випадкові зміни у хромосоми для підтримки генетичного різноманіття та уникнення передчасної збіжності до локальних оптимумів. Оцінка пристосованості обчислює якість кожного розв'язку, яка у випадку задачі комівояжера зазвичай відображає загальну довжину маршруту.

Батьківські хромосоми позначені як «Parent_1 Chromosomes», «Parent_2 Chromosomes» і так далі до «Parent_N Chromosomes». Дані хромосоми розподіляються між робітниками через канали передачі даних для проведення генетичних операцій. Кожен робітник отримує батьківські хромосоми, виконує над ними операції кросоверу та мутації, оцінює пристосованість отриманих нащадків і відправляє результати назад до менеджера.

Нові нащадки, утворені в результаті генетичних операцій, позначені як «New Offspring_1», «New Offspring_2» і так далі до «New Offspring_N». Дані нащадки, створені усіма робітниками, збираються менеджером і формують нову популяцію для наступного покоління. Процес формування нової популяції може включати стратегію елітизму, коли певна кількість найкращих особин з поточного покоління безпосередньо переходить до наступного покоління без змін [17 - 19].

Пунктирні лінії на схемі відображають потоки даних між компонентами системи. Вони показують, як генетична інформація передається від менеджера до робітників і назад, формуючи циклічний процес еволюції. Менеджер розподіляє батьківські хромосоми між робітниками, робітники виконують

генетичні операції та повертають нових нащадків, які потім формують нову популяцію.

Запропонована архітектура забезпечує ефективне паралельне виконання генетичного алгоритму, розподіляючи обчислювальне навантаження між кількома процесорами або обчислювальними вузлами. Така паралельна реалізація дозволяє значно скоротити час обчислень для великих екземплярів задачі комівояжера, особливо коли популяція містить велику кількість особин, або коли розмірність задачі (кількість міст) є значною.

Архітектура використовує розподілену модель пам'яті, де менеджер зберігає глобальну популяцію, тоді як робітники отримують лише необхідні для обробки дані. Це забезпечує ефективне використання пам'яті та мінімізує обсяг даних, що передається між компонентами системи. Крім того, така архітектура забезпечує балансування навантаження між процесорами та дозволяє легко масштабувати систему, додаючи нові робітники при наявності додаткових обчислювальних ресурсів [20].

Цикл роботи алгоритму починається з ініціалізації популяції менеджером. Потім батьківські хромосоми розподіляються між робітниками для виконання генетичних операцій. Кожен робітник виконує кросовер, мутацію та оцінку пристосованості незалежно від інших. Нові нащадки, створені робітниками, збираються менеджером і формують нову популяцію. Цей процес повторюється для заданої кількості поколінь або до досягнення критерію зупинки, наприклад, відсутності суттєвого покращення протягом певної кількості поколінь [18].

2.2 Розробка UML-діаграм програмного модуля задачі комівояжера

UML (Unified Modeling Language) є стандартизованою мовою моделювання, яка надає широкий набір інструментів для аналізу, проектування та документування програмних систем. UML поділяється на кілька груп діаграм, які дозволяють відображати різні аспекти системи.

Структурні діаграми UML відображають статичну структуру системи.

Діаграма класів показує класи системи з їхніми атрибутами, методами та взаємозв'язками, що дозволяє представити об'єктну модель системи. Діаграма компонентів відображає фізичну структуру системи, визначаючи компоненти та їхні взаємозв'язки, що допомагає розуміти архітектуру системи на фізичному рівні. Діаграма пакетів використовується для групування елементів системи у логічні пакети та відображення залежностей між ними, що спрощує розуміння структури великих систем.

Поведінкові діаграми UML моделюють динамічну поведінку системи. Діаграма послідовності відображає взаємодію між об'єктами у вигляді послідовності повідомлень, демонструючи виконання операцій та обмін даними між об'єктами у часовій перспективі. Діаграма станів моделює поведінку об'єкта через різні стани та переходи між ними, показуючи, як об'єкт реагує на події та змінює свій стан протягом життєвого циклу.

UML також надає діаграми взаємодії, такі як діаграми комунікації та діаграми співпраці, які моделюють складні взаємодії між об'єктами системи з різних перспектив. Діаграма розгортання показує фізичну архітектуру системи, відображаючи розміщення апаратного та програмного забезпечення на різних вузлах системи, що допомагає зрозуміти, як система буде розгорнута в реальному середовищі.

Всі ці діаграми UML є важливими інструментами на різних етапах життєвого циклу розробки програмного забезпечення. Вони допомагають розробникам встановлювати зв'язки між елементами системи, розуміти їхню структуру та поведінку, а також ефективно передавати цю інформацію іншим учасникам проекту. Завдяки стандартизованому підходу та гнучкості, UML може використовуватися на етапах аналізу вимог, проектування архітектури, визначення поведінки та взаємодії об'єктів, тестування та документування системи.

Основна мета використання UML-діаграм полягає у спрощенні розуміння та комунікації між усіма учасниками проекту, включаючи розробників, аналітиків, дизайнерів та інших зацікавлених сторін. UML-

діаграми дозволяють візуалізувати складність системи, виявляти потенційні проблеми та ризики на ранніх етапах розробки, а також створювати основу для подальшого програмування та розвитку системи.

Існує кілька відомих засобів розробки UML, які надають потужні інструменти для створення, візуалізації та аналізу UML-діаграм. Enterprise Architect є комплексним засобом, який підтримує всі типи UML-діаграм, дозволяє відстежувати залежності, аналізувати моделі та генерувати код з діаграм. Visual Paradigm також пропонує широкі можливості для створення різних типів діаграм, підтримує повторне використання елементів моделі, генерацію коду та колективну роботу. IBM Rational Rose, один із перших інструментів для UML, забезпечує функціонал для моделювання різних діаграм, імпорту та експорту моделей, аналізу моделей та генерації коду.

Принципи розробки UML базуються на використанні стандартизованих нотацій, правил і конвенцій для побудови діаграм, що забезпечує їхню зрозумілість та полегшує співпрацю між розробниками. Стандартизована нотація UML дозволяє однозначно виражати структуру та поведінку системи, забезпечуючи чітку комунікацію між учасниками проекту. Принцип модульності дозволяє розбивати систему на модулі та компоненти, що спрощує розробку, тестування та управління проектом, а кожна діаграма може фокусуватися на конкретному аспекті системи.

UML надає як структурні, так і поведінкові діаграми, що дозволяє аналізувати різні аспекти системи. Принцип розширюваності дозволяє визначати власні профілі та розширювати базові нотації за допомогою стереотипів, тегів та обмежень, адаптуючи нотацію до конкретних потреб проекту. UML підтримує документацію та аналіз системи, охоплюючи її структуру, поведінку та взаємозв'язки, що сприяє виявленню проблем та покращенню якості програмного забезпечення.

UML підтримує весь життєвий цикл розробки, від аналізу та проектування до реалізації та тестування, зберігаючи інформацію про систему протягом усього її життєвого циклу. Принцип колективної роботи

підтримується засобами розробки UML, які дозволяють спільно працювати над моделями, обмінюватися даними, відстежувати зміни та коментувати моделі, що сприяє ефективній комунікації та співпраці в команді розробників.

UML є гнучкою та розширюваною мовою, що дозволяє моделювати різні типи систем, включаючи програмні системи, бізнес-процеси та архітектуру. Він може використовуватися в різних галузях та проектах, адаптуючись до конкретних потреб розробки.

На рисунку 2.5 зображено UML-діаграму прецедентів додатку задачі комівояжерів, єдиним актором в системі виступає власне сам користувач.

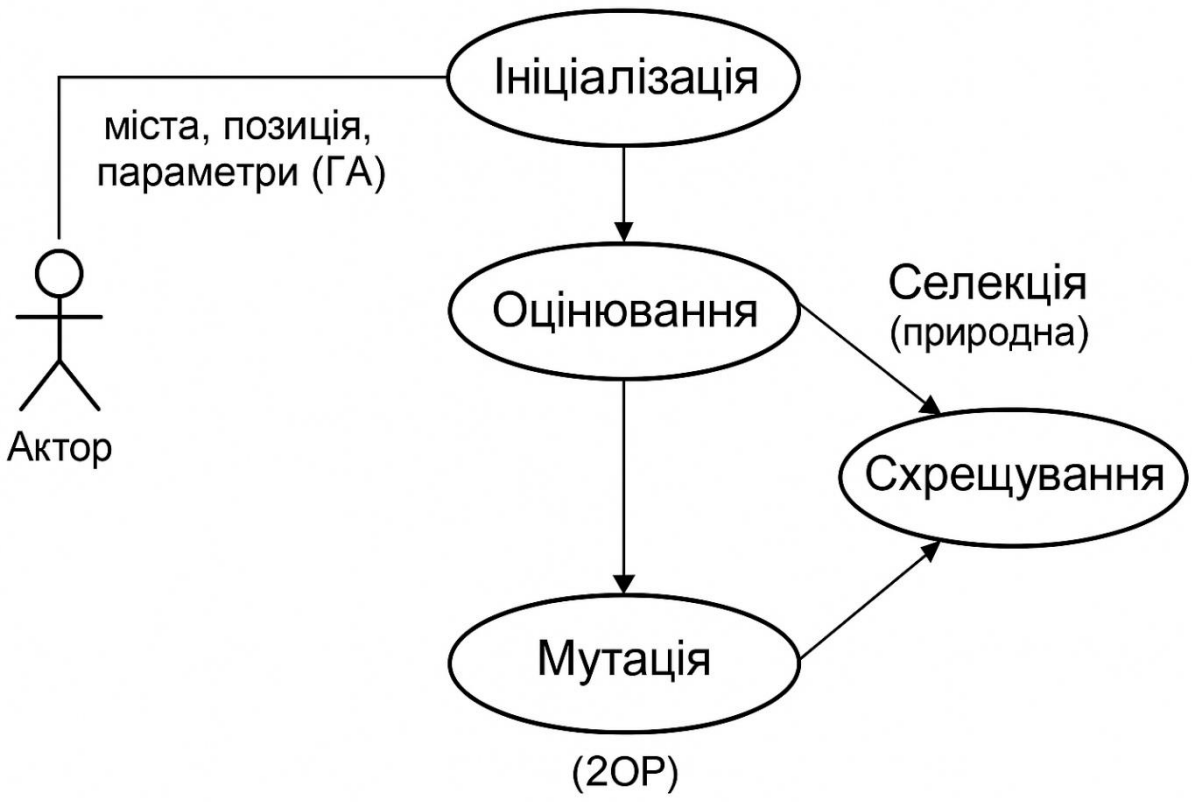


Рисунок 2.5 – Діаграма прецедентів програмного модуля задачі комівояжера

Наступною розробленою діаграмою є діаграма діяльності, що відображає процес роботи генетичного алгоритму для розв'язання задачі комівояжера. Діаграма показує послідовність операцій та потік даних у

алгоритмі (рис.2.6).

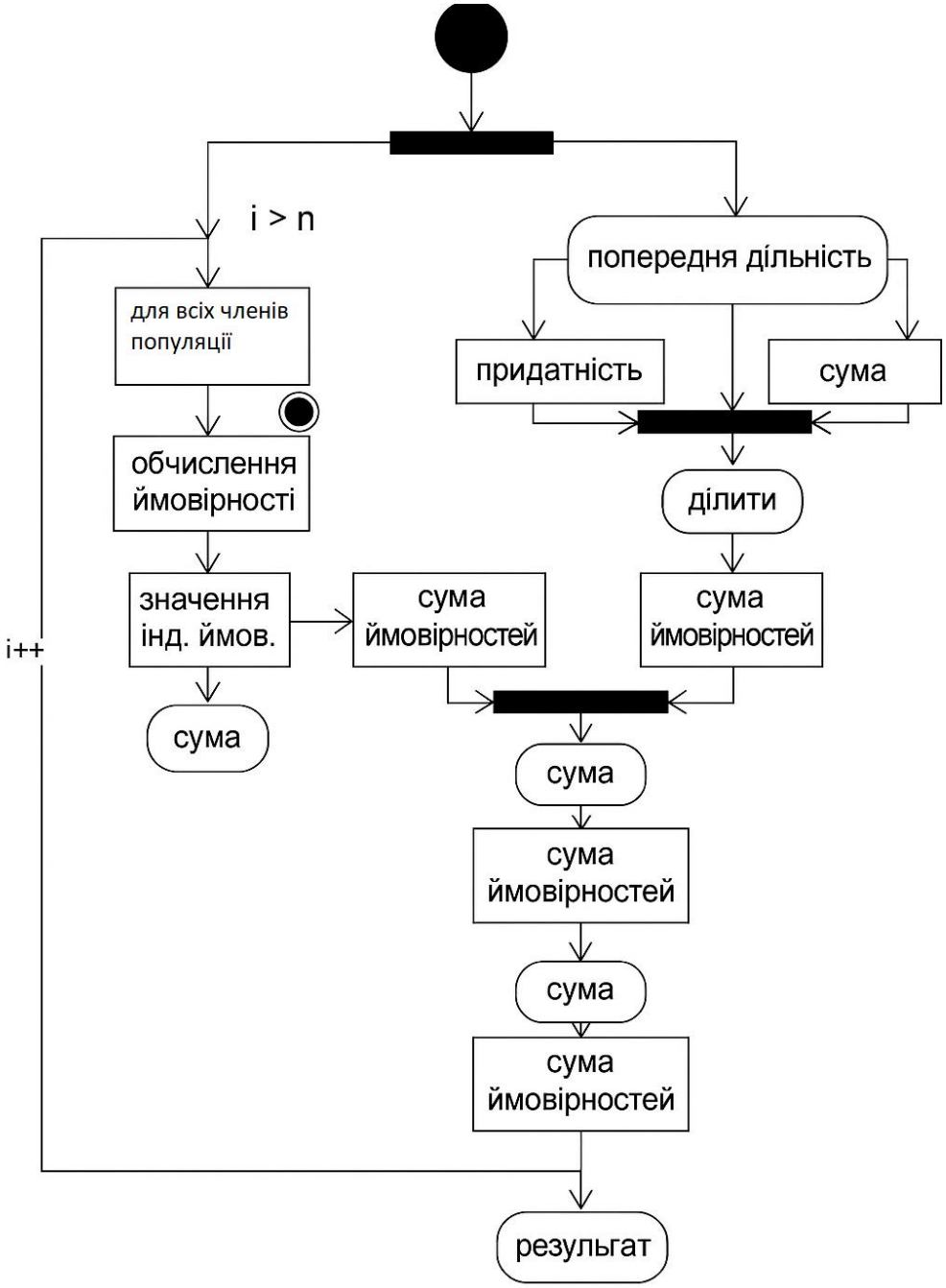


Рисунок 2.6 – Діаграма діяльності для генетичного алгоритму в контексті вирішення задачі комівояжера

На початку процесу, який позначено чорною крапкою у верхній частині діаграми, відбувається ініціалізація алгоритму. Далі виконання розгалужується на два паралельні потоки. Ліва гілка відповідає за обробку

всієї популяції, тоді як права — за розрахунок значень придатності (фітнесу) для особин.

У лівій частині діаграми починається цикл «для всіх членів популяції», який забезпечує обхід усіх особин поточної популяції. Для кожної з них виконується обчислення ймовірності, далі визначається значення інд. ймовірності, після чого ці ймовірності підсумовуються (сума) та передаються у блок «сума ймовірностей». Цей процес відповідає етапу селекції в межах генетичного алгоритму, де особини з вищою придатністю отримують вищі ймовірності бути обраними для створення нащадків.

У правій частині діаграми, з блоку «попередня діяльність», паралельно починаються два підпроцеси: обчислення фітнесу для кожного маршруту та обчислення суми. Обидва потоки з’єднуються на етапі синхронізації, після чого відбувається дія «ділити». В результаті формується сума ймовірностей, яка далі потрапляє до основного об’єднуючого вузла.

Після завершення обох паралельних потоків — обробки популяції та оцінки придатності — здійснюється їх злиття через синхронізаційний вузол (позначено чорним прямокутником). В об’єднаній гілці дані проходять через блоки «сума», «сума ймовірностей» та ще раз «сума», після чого формується фінальний результат у блоці «результат».

У контексті задачі комівояжера ця діаграма демонструє, як відбувається вибір найкращих маршрутів з поточної популяції на основі значень їх придатності. Кожен маршрут (особина) оцінюється згідно з функцією придатності, яка зазвичай вимірюється загальною довжиною маршруту. Чим коротший маршрут, тим вищий рівень його пристосованості. Завдяки блоку обчислення ймовірності, найпридатніші маршрути отримують перевагу при відборі для подальших генетичних операцій — кросоверу та мутації.

Отриманий результат на виході — це або нова популяція маршрутів, або найкращий з них у межах даної ітерації. Процедура повторюється для визначеної кількості поколінь, що дозволяє системі поступово поліпшувати

якість рішень та наближатися до глобального оптимуму.

Побудуємо діаграму послідовності роботи програмного модуля задачі комівояжера з використанням генетичного алгоритму для побудови графу міст (рис.2.7).

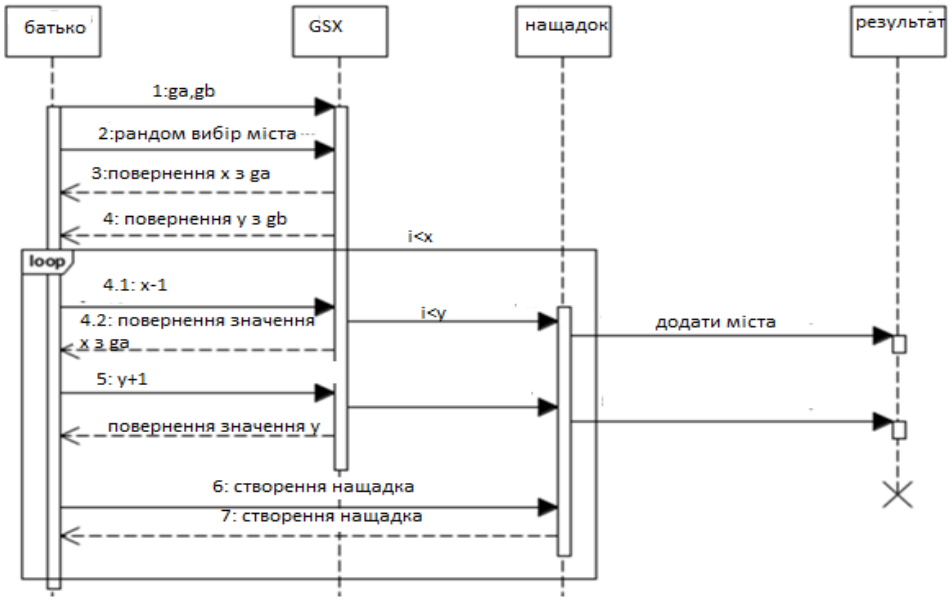


Рисунок 2.7 – Діаграма послідовності для генетичного алгоритму в контексті вирішення задачі комівояжера

Представлена діаграма послідовності ілюструє процес кросоверу (схрещування) в генетичному алгоритмі для розв'язання задачі комівояжера. Діаграма відображає взаємодію між чотирма основними об'єктами: PARENTs (батьківські хромосоми), GSX (оператор кросоверу), CHILD (дочірня хромосома) та RESULT (результат операції).

Процес починається з ініціалізації батьківських хромосом, які представляють два різних маршрути комівояжера. Кожна хромосома кодує певну послідовність міст, яку має відвідати комівояжер. У контексті задачі комівояжера, батьківські хромосоми містять перестановки міст, що представляють можливі маршрути.

На першому кроці (1: ga,gb) батьківські хромосоми передаються до оператора GSX (Greedy Subtour Crossover – жадібний кросовер підмаршрутів), який є спеціалізованим оператором кросоверу для задачі комівояжера. Цей оператор забезпечує створення допустимого маршруту (без повторень міст) при схрещуванні двох батьківських маршрутів.

На другому кроці (2: choose one town at random) оператор GSX випадковим чином вибирає одне місто, з якого починається процес формування нового маршруту. Це створює точку перетину для батьківських маршрутів, звідки почнеться побудова дочірнього маршруту.

На третьому та четвертому кроках (3: return value x from ga, 4: return value y from gb) відбувається отримання значень x і y від першого та другого батьківських маршрутів відповідно. У контексті задачі комівояжера ці значення представляють міста або ребра між містами, які будуть розглядатися для включення в дочірній маршрут.

Далі починається цикл, який формує дочірній маршрут. Крок 4: x-1 позначає ітерацію по першому батьківському маршруту, де на кожній ітерації отримується місто з першого маршруту (4.2: return value x from ga). Якщо поточний крок відповідає певній умові (4.1: if current step i<x), тоді виконується додавання міста з першого батьківського маршруту до дочірнього маршруту (4.1.1: add towns from first parent).

Аналогічно, крок 5: y+1 позначає ітерацію по другому батьківському маршруту, де отримується місто з другого маршруту (5.2: return value y from gb). Якщо поточний крок відповідає умові (5.1: if current step i>y), відбувається додавання міста з другого батьківського маршруту до дочірнього маршруту (5.1.1: add towns from first parent). Зауважте, що тут є потенційна помилка в діаграмі, оскільки повинно бути "second parent" замість "first parent".

Після завершення циклу, на шостому кроці (6: create child g) відбувається створення дочірньої хромосоми, яка представляє новий маршрут комівояжера, сформований шляхом комбінування частин двох батьківських маршрутів.

На сьомому кроці (7: return child g) дочірня хромосома повертається як результат операції кросоверу.

Важливо зазначити, що цей процес кросоверу забезпечує збереження допустимості маршруту комівояжера, тобто кожне місто відвідується рівно один раз, і маршрут формує замкнутий цикл. Оператор GSX спеціально розроблений для задачі комівояжера, щоб уникнути створення недопустимих маршрутів із повтореннями міст або пропущеними містами.

Також побудуємо діаграму діяльності мутації в контексті розробки маршрутів для задачі комівояжера (рис. 2.8).

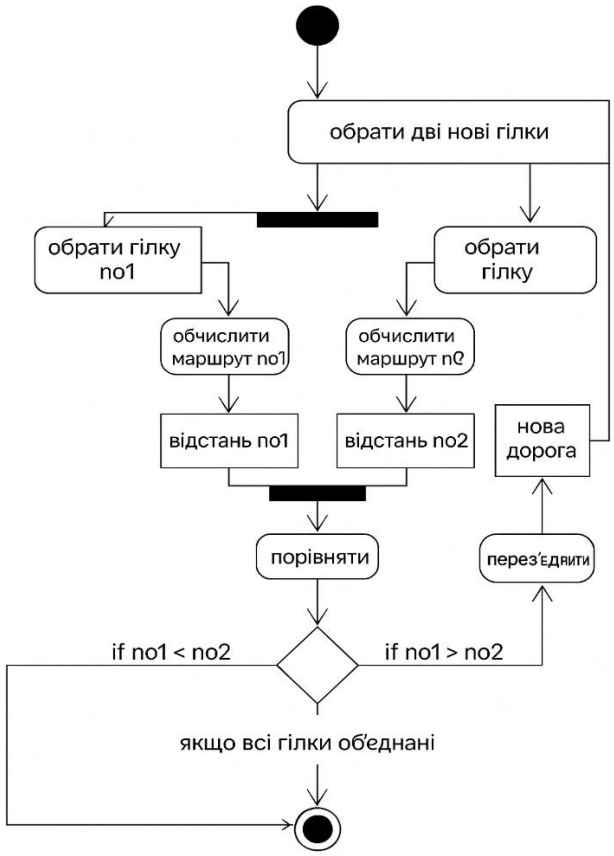


Рисунок 2.8 – Діаграма діяльності мутації для генетичного алгоритму в контексті вирішення задачі комівояжера

Діаграма відображає процес мутації маршрутів у контексті комівояжера, де шлях оптимізується через модифікацію структур існуючих рішень.

Процес починається з вибору двох маршрутів, що підлягають порівнянню та мутаціям. Кожен з них проходить через такі етапи: спочатку здійснюється вибір двох маршрутів серед поточної популяції, що можуть містити як ефективні, так і неефективні рішення. Далі виконується обчислення їхньої довжини, що дозволяє визначити їхню ефективність. Після цього порівнюються відстані двох маршрутів, що дає змогу визначити, який із них має кращі характеристики. Якщо один із маршрутів має значно більшу довжину, він може бути замінений або модифікований за рахунок операції мутації.

Мутаційний етап включає «перехресне з'єднання» (cross-connect), яке змінює структуру маршруту, комбінуючи частини двох вихідних шляхів, що може призвести до покращення результату. Після цього створюється новий модифікований маршрут, який може бути прийнятий у популяцію замість одного з попередніх варіантів. Процес триває доти, доки не буде досягнуто певного критерію зупинки, наприклад, поки всі можливі маршрути не будуть оптимізовані або поки не відбудеться певна кількість ітерацій.

Також зобразимо основну діаграму, за якої власне і буде проектуватись програмний модуль – а саме, діаграму класів (рис.2.9).

Основні класи:

– Місто - зберігає координати та номер міста, обчислює відстані

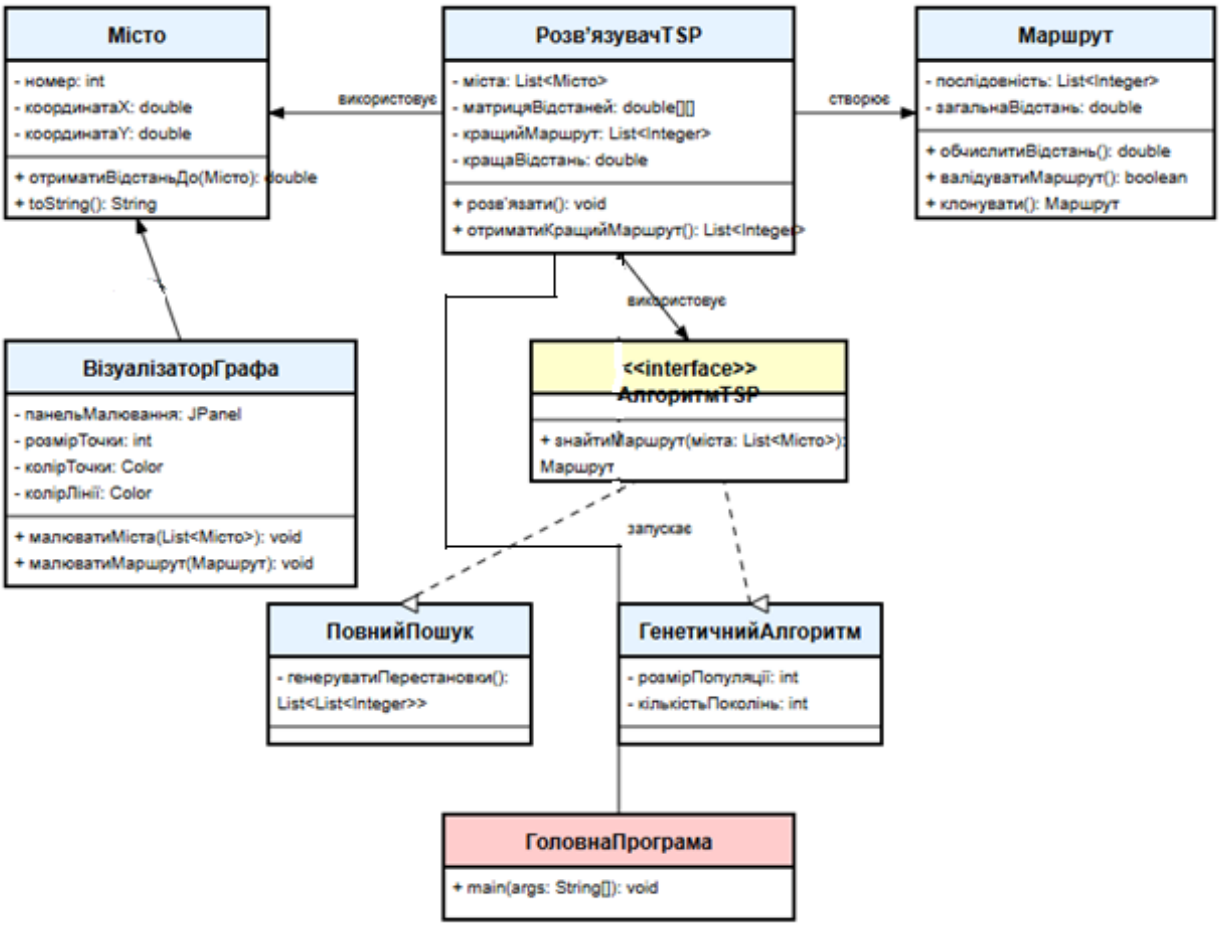
– РозвʼязувачTSP - головний клас, містить міста, матрицю відстаней, знаходить оптимальний маршрут

– Маршрут - представляє послідовність міст та загальну відстань

– ВізуалізаторГрафа - відображає міста та маршрути на екрані

– АлгоритмTSP - інтерфейс для алгоритмів пошуку

– ПовнийПошук / ГенетичнийАлгоритм - конкретні алгоритми розв'язання



– ГоловнаПрограма - запускає систему.

Рисунок 2.9 – Діаграма класів програмного модуля для вирішення задачі комівояжера

Зв'язки між класами:

РозвʼязувачTSP використовує АлгоритмTSP, що дозволяє йому бути незалежним від реалізації (генетичний або повний пошук).

ГенетичнийАлгоритм та ПовнийПошук реалізують спільний інтерфейс, що дозволяє змінювати алгоритми без змін у логіці.

Маршрут створюється та використовується в обох алгоритмах.

ВізуалізаторГрафа працює з об'єктами Місто та Маршрут для виведення графічного результату.

ГоловнаПрограма ініціалізує та запускає весь процес.

2.3 Висновок до розділу

В другому розділі розглянуто генетичний алгоритм (ГА) - як евристичний метод пошуку та оптимізації, заснований на принципах природної еволюції. розглянута структурна схема і алгоритми для програмного модуля вирішення задачі комівояжера з використанням генетичного алгоритму, де менеджер керує процесом еволюції популяції, а робітники здійснюють генетичні операції (кросовер, мутацію, оцінку пристосованості).

Розглянута структурна схема демонструє ефективне використання паралельних обчислювальних вузлів для значного зменшення часу виконання при великих розмірах задачі.

А також розглянуті UML-діаграми, зокрема діаграма діяльності, наочно ілюструють послідовність операцій алгоритму, зокрема етапи селекції та нормалізації фітнес-функцій, що є важливими для вибору оптимальних маршрутів. великою кількістю міст в задачі комівояжера. Також розглянуті основні класи і зв’язки між ними, для покращення маштабованості програмного модуля задачі комівояжера в подальшому

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

3.1 Обґрунтування вибору мови програмування і середовища розробки

В таблиці 3.1 наведено порівняльну характеристику мов програмування для проектування програмного модуля задачі комівояжера.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Мова програмування	Опис	Переваги	Недоліки
C#	Використовується для створення операційних систем та офісних рішень на платформі Microsoft, розробки веб-додатків, настільних графічних програм (Windows Forms і WPF), серверних програм для динамічного веб-контенту (ASP.NET), мобільних	Швидка розробка для малих проектів, простота коду, зручне середовище розробки, підтримка спадкування та універсалізації, оптимізація через CLR (компілятор проміжної мови), простота створення Web-служб.	Проблеми при розробці для платформ, відмінних від Windows, сильна залежність від Microsoft, потреба у .NET Framework, відсутність деструкторів та обмежена підтримка шаблонів.

Таблиця 3.1 (продовження)

	додатків для Windows Phone.		
C++	Створення операційних систем, прикладних програм, драйверів, додатків для вбудованих систем, серверів, програмування ігор.	Висока сумісність з C, підтримка різних стилів програмування, кросплатформеність, доступність, популярність, можливість створення універсальних контейнерів і алгоритмів.	Нерідко незручний синтаксис для певних задач, потреба в runtime для самодостатності додатків, низька продуктивність програмістів, що призводить до низької якості продукції.
Java	Використовується для створення десктопних програм, мобільних додатків, серверних програм для мережевої роботи, програмування ігор	Універсальність та багатофункціональність, кросплатформеність, відкритий код, велика кількість бібліотек, гнучка система безпеки, висока продуктивність, підтримка багатозадачності.	Відсутність індивідуальних змінних та множинного спадкування, відсутність спливаючих підказок для методів.

Для створення програмного модуля задачі комівояжера було вирішено обрати C# як мову розробки з кількох причин.

C# є потужною мовою програмування, яка надає ряд значних переваг, що робить її ідеальним вибором для розробки додатків, особливо для тих, що

мають інтеграцію з продуктами Microsoft. Однією з основних переваг є висока швидкість розробки, яка забезпечується завдяки зручному середовищу розробки — Visual Studio. Цей інструмент пропонує автоматичне доповнення коду, підказки, потужний дебаггер, що дозволяє скоротити час на розробку та тестування. Завдяки можливості використовувати готові бібліотеки та фреймворки, розробник може швидко реалізувати функціональність, не витрачаючи багато часу на створення низькорівневих компонентів. Це дає змогу прискорити виведення продукту на ринок або зменшити витрати на розробку. Крім того, C# має вбудовану підтримку об'єктно-орієнтованого програмування, що дозволяє будувати масштабовані й легко підтримувані системи.

Іншою вагомою перевагою є інтеграція з іншими технологіями та продуктами Microsoft, що робить її найкращим вибором для розробки програмного забезпечення для Windows. Інструменти, як ASP.NET для розробки веб-додатків, WPF для створення настільних програм та SQL Server для баз даних, інтегруються з C# на найвищому рівні, що забезпечує стабільність і ефективність роботи додатків. Це дозволяє зосередитись на розв'язанні конкретних завдань, не турбуючись про налаштування складних інтерфейсів між різними технологіями.

Велика кількість готових бібліотек і фреймворків також є значною перевагою C#. З його допомогою можна швидко використовувати перевірені рішення для різних завдань, таких як робота з мережами, обробка даних, криптографія та інше. Це дозволяє суттєво зменшити час на розробку, а також підвищити якість кінцевого продукту.

C# забезпечує високу безпеку та надійність програм, що розробляються на його основі. Мова містить вбудовані механізми для управління пам'яттю, обробки виключень та підтримки типів даних, що зменшує кількість помилок під час виконання програми. Це є важливою перевагою для створення додатків, де стабільність і безпека є критичними.

Ще однією важливою характеристикою є велика спільнота розробників

та підтримка Microsoft. Завдяки цьому, розробники можуть швидко знаходити рішення на різні питання в Інтернеті, обговорювати проблеми на форумах або використовувати велику кількість навчальних матеріалів.

Для розробки додатку задачі комівояжера вибір середовища Visual Studio в парі з мовою C# є обґрунтованим і стратегічно правильним. Це середовище розробки має численні інструменти, які значно спрощують і пришвидшують процес створення програмного забезпечення. Visual Studio надає все необхідне для написання, налагодження та тестування коду в одному інтегрованому середовищі.

По-перше, Visual Studio пропонує потужні можливості для автоматичного доповнення коду, підказки та інтеграцію з різноманітними бібліотеками, що значно прискорює процес розробки. Це дозволяє розробникам писати код швидше, а також зменшує ймовірність помилок. Вбудовані інструменти для рефакторингу і оптимізації коду допомагають підтримувати його в чистоті і відповідно до сучасних стандартів програмування.

Однією з ключових переваг Visual Studio є її гнучкість і підтримка багатьох мов програмування. Крім того, Visual Studio має чудову підтримку для створення багатоплатформених додатків, що є важливим для розробки нашого продукту, який може потребувати розширення на різні операційні системи.

Не менш важливим є потужний дебаггер, який дозволяє відслідковувати проблеми в коді на всіх етапах його виконання. Це особливо корисно під час розробки складних функціональних частин додатку, оскільки допомагає швидко знаходити і виправляти помилки без необхідності вручну шукати їх у великому кодовому базі. Ще одним важливим аспектом є інтеграція з системами контролю версій, такими як Git, що дозволяє ефективно, зберігати і відстежувати зміни в коді. Це дає змогу вести історію змін і забезпечувати належний контроль над версіями програмного забезпечення.

3.2 Розробка програмного модуля задачі комівояжера

Розглянемо детально створення коду для програмного модуля задачі комівояжера. У самому класі MainForm знаходяться різні властивості, що визначають налаштування додатку. Однією з таких властивостей є CountCpuCore, яка вказує кількість ядер процесора, що буде використовуватися під час виконання генетичного алгоритму. Це дозволяє оптимізувати використання ресурсів системи та пришвидшити виконання обчислень. Інша важлива властивість — PopulationNumber, що визначає кількість популяцій в алгоритмі. Змінна _nKeep визначає кількість збережених хромосом під час кожного покоління.

Також є властивості, що стосуються графічного відображення даних, наприклад ShapeContainerAllCityShape для контейнери всіх графічних елементів на формі, таких як міста та шляхи. Властивість LineShapeWay відповідає за малювання ліній між містами, а OvalShapeCity містить елементи для відображення самих міст.

У класі MainForm є різні методи для обробки графічного інтерфейсу та взаємодії з ним. Наприклад, методи SetValue, SetMaxValue, SetGenerationText відповідають за оновлення елементів інтерфейсу, таких як прогрес-бар та текстові поля, під час виконання генетичного алгоритму. Це дозволяє користувачу бачити хід виконання алгоритму в реальному часі.

Важливим є метод Ga, який реалізує генетичний алгоритм для вирішення задачі комівояжера. Метод починається з ініціалізації деяких змінних, наприклад, rand, який використовується для генерації випадкових чисел, і eliteFitness, що зберігає найкращу адаптивність хромосом протягом виконання алгоритму. Потім задається початкова позиція міст і ініціалізується паралельне виконання для генетичного алгоритму, що дозволяє використовувати кілька ядер процесора для пришвидшення обчислень. Далі відбувається обчислення прогресу поколінь, оновлення графіків та значень прогресу на інтерфейсі, зокрема за допомогою методів SetTimeGraph і

RefreshTour.

Метод Ga також містить кілька етапів, таких як відбір, елітизм та кросовер, що забезпечують пошук найкращих рішень у популяції. Ці етапи включають в себе сортування хромосом за їх адаптивністю, вибір елітних хромосом для збереження їх у наступному поколінні, а також відбір найгірших хромосом для їх подальшого видалення. Всі ці кроки виконуються до тих пір, поки не буде знайдено оптимальне рішення задачі.

У коді також є використання асинхронних викликів, таких як делегати SetValueCallback та AddShapeCallback, для асинхронного оновлення інтерфейсу, що дозволяє забезпечити плавність роботи програми навіть при великих обчисленнях. Завдяки цьому, користувач може спостерігати за процесом пошуку рішення без зависання інтерфейсу.

За формування маршрутів методом генетичних алгоритмів відповідають наступні методи в коді:

1. Rank_Trim()

Цей метод визначає ймовірність вибору кожної хромосоми для подальшого використання у генетичному алгоритмі, залежно від її адаптивності (fitness). У методу є наступні кроки:

– Обчислюється загальна сума всіх адаптивностей хромосом.

– Визначається середнє значення адаптивності для всіх хромосом.

– Хромосоми, адаптивність яких нижча за середнє значення, не будуть зберігатися для наступного покоління.

– Визначаються ймовірності для кожної хромосоми, які пропорційні її адаптивності.

2. x_Rate()

Метод визначає відсоток хромосом, які будуть видалені, та які будуть корисні для подальшого розвитку (N_keep). Для цього розраховується середня адаптивність усіх хромосом, а потім підраховується кількість хромосом, адаптивність яких перевищує середню.

3. Rank()

Цей метод визначає, яка хромосома буде вибрана як батьківська для створення нових хромосом. Працює за принципом генерації випадкового числа і перевірки, яка хромосома підпадає під ймовірність її вибору, визначену раніше у методі Rank_Trim().

4. Isotropy_Evaluatuon()

Цей метод перевіряє ізотропність хромосом, тобто чи є в популяції достатня кількість хромосом, що мають подібні характеристики (фітнес). Якщо більше половини хромосом мають схожий фітнес, алгоритм зупиняється. Інакше, процес триває.

5. ReproduceByParallelThreads()

Метод відповідає за паралельне відтворення нових хромосом за допомогою багатопоточності. Він розбиває роботу на кілька потоків (threads), де кожен потік відповідає за генерацію певної частини нових хромосом.

– Створюється масив потоків, і кожен потік обробляє свою частину популяції.

– Використовується семафор для синхронізації доступу до спільних ресурсів і контролю кількості одночасних виконань потоків.

– Після виконання всіх потоків синхронізується завершення роботи.

6. ReproduceByParallelTask()

Цей метод використовує Task Parallelism для створення нових хромосом. Він працює схожим чином до попереднього, але замість потоків використовуються завдання (tasks), що дозволяє краще керувати паралельною роботою та обробкою.

7. Reproduction()

Це серійна версія методу відтворення. У цьому методі для кожної хромосоми в популяції:

– Вибираються батьківські хромосоми (без дублювання).
– Виконується кросовер (поєднання генів батьків).
– Виконується мутація (зміни генів).
– Оцінюється фітнес нової хромосоми.

Для того, щоб уникнути ситуацій, коли вибрані батьки є однаковими (що може трапитися випадково), виконується цикл перевірки, поки не будуть вибрані два різні батьки.

8. PReproduction()

Це паралельна версія методу відтворення. Вона працює так само, як серійний метод, але з паралельною обробкою. Це дозволяє прискорити процес генерації нових хромосом при великій популяції.

Метод create_City відповідає за створення нового міста на основі переданої точки на екрані. Кожен раз, коли користувач натискає на форму, цей метод створює об'єкт OvalShape, що представляє місто, задає його координати, розмір, стиль та кольори, а також додає його до контейнера для відображення на екрані. Крім того, метод оновлює лічильник міст в статусному рядку, щоб відобразити актуальну кількість міст.

Метод RefreshTour виконує оновлення маршруту між містами, що важливо для задачі комівояжера. Всі попередні лінії, що з'єднують міста, видаляються, і потім для кожної пари міст, відповідно до геному елітного хромосоми в популяції генетичного алгоритму, малюється лінія між ними. Це дозволяє наочно відобразити поточний оптимальний шлях, який було знайдено алгоритмом.

Метод ovalShape_Click дозволяє користувачу видаляти міста при натисканні на них. Якщо місто було обране, воно видаляється з екрану, а кількість міст зменшується. Також цей метод оновлює список міст, що відображається у таблиці, аби користувач міг побачити актуальні координати залишкових міст.

Метод MainForm_MouseClick реагує на натискання миші в межах форми. Якщо точка натискання знаходиться в допустимих межах екрану, створюється нове місто, і старі лінії маршруту між містами видаляються. Це дозволяє динамічно додавати нові міста на форму без потреби перезапускати програму.

Метод importToolStripMenuItem_Click дозволяє імпортувати

координати міст з текстового файлу. Після імпорту даних міста додаються на екран, а попередньо додані міста і лінії маршруту видаляються. Це зручно для роботи з існуючими наборами даних.

Метод newRandomCitiesToolStripMenuItem_Click генерує випадкові міста в межах екрану. Для кожного нового міста перевіряється, чи не накладається воно на інші міста, щоб уникнути перекриття. Якщо таке місто не можна безпечно додати, шукається інша точка, яка буде дотримуватись заданих умов безпеки.

Метод refreshDGV_CityPositions відповідає за оновлення списку міст у таблиці на екрані, відображаючи їхні поточні координати. Цей метод викликається після кожної зміни міста, що забезпечує актуальність відображених даних у інтерфейсі користувача.

Дані методи працюють разом, щоб забезпечити інтерактивну роботу з містами та маршрутами в задачі комівояжера, дозволяючи додавати, видаляти, імпортувати і генерувати нові міста, а також динамічно відображати зміну маршруту, що відображає результат роботи генетичного алгоритму.

У процесі розробки додатку було реалізовано ряд методів, що забезпечують як взаємодію користувача з інтерфейсом, так і логіку обчислень. Наприклад, метод SetThreadPriority відповідає за налаштування пріоритету потоку залежно від загального пріоритету процесу. Це дозволяє ефективніше розподіляти обчислювальні ресурси системи, особливо під час виконання обчислювально-інтенсивних задач, таких як генетичні алгоритми. Додатково викликається BeginThreadAffinity(), що дозволяє «прив’язати» потік до певного ядра, покращуючи детермінованість виконання.

Метод SetTimeGraph використовується для відображення динаміки зміни показників у вигляді графіків. У ньому розраховується поточна тривалість роботи алгоритму, і якщо оновлено значення «елітної пристосованості» (fitness), ці дані додаються до відповідних графіків — як у часі, так і по поколіннях. Це дозволяє візуально оцінити ефективність оптимізації та прогрес алгоритму в реальному часі.

Метод refreshDGV_CityPositions відповідає за оновлення таблиці з координатами міст, які використовуються для побудови маршруту. Після кожного додавання або видалення міста таблиця оновлюється, відображаючи точне положення кожного з них. Це особливо корисно для контролю початкових умов у задачі маршрутизації.

Створення нових міст реалізовано в методі create_City, який динамічно додає графічний об’єкт у вигляді кола (OvalShape) на форму у відповідному місці натискання миші. Кожне місто отримує унікальні властивості — розмір, колір, обробник подій натискання — та зберігається в загальному списку для подальшої обробки.

Метод RefreshTour здійснює візуальне оновлення маршруту між усіма містами на основі найкращого знайденого розв’язку (елітної хромосоми) генетичного алгоритму. Він видаляє попередні лінії маршруту та малює нові, створюючи пов’язаний шлях між містами. Остання лінія з’єднує перше та останнє місто, замкнувши тур.

Метод ovalShape_Click дозволяє видаляти міста, які користувач обирає, та автоматично оновлює інформацію в таблиці координат і статусному рядку. Це реалізує динамічну модифікацію вхідних даних.

Крім того, у коді присутній метод MainForm_MouseClick, який визначає координати кліку миші користувача та, в разі знаходження у допустимих межах, створює нове місто у заданій позиції. Одночасно очищується попередній маршрут, якщо такий був, і оновлюється графічне представлення.

Також реалізовано методи очищення форми та даних (newToolStripMenuItem_Click) та імпорту збережених координат міст (importToolStripMenuItem_Click), що забезпечують повторне використання даних, збереження сценаріїв і гнучкість користування додатком.

Окрім методів, що відповідають за побудову маршруту та графічне відображення елементів, у додатку також реалізовані функції, які забезпечують повноцінну взаємодію з користувачем, обробку подій та зміну параметрів оптимізації в режимі реального часу. Наприклад, метод

numPopulation_ValueChanged фіксує зміну кількості особин у популяції генетичного алгоритму, яку користувач може задати через елемент керування — числове поле. Це дозволяє адаптувати швидкість і точність пошуку оптимального рішення без необхідності перезапуску всієї програми.

Метод MainForm_FormClosing викликається під час закриття головного вікна програми. У ньому реалізовано завершення процесів і звільнення ресурсів, зокрема виклик методу Stop(), що гарантує зупинку всіх обчислювальних потоків. Це запобігає можливим помилкам або витокам пам’яті при повторному запуску програми.

Значну увагу приділено також імпорту даних. У методі importToolStripMenuItem_Click передбачено відкриття діалогового вікна для вибору текстового файлу з координатами міст. Після вибору файла стара інформація очищується — видаляються всі міста та маршрути — і готується інтерфейс для завантаження нових даних. Це забезпечує можливість використовувати збережені сценарії, повторно виконувати розрахунки або моделювати нові задачі без ручного введення координат.

Кожен елемент візуалізації міста або маршруту реалізується через спеціальні графічні об'єкти, такі як OvalShape для міст і LineShape для маршрутів. Їх додавання, переміщення та видалення відбувається в реальному часі з повною інтеграцією у подієво-орієнтовану модель додатку. Усі графічні зміни доповнюються оновленням таблиць та статусних рядків, що значно покращує зручність користування і дозволяє керівнику або аналітику швидко оцінити ефективність запропонованих рішень.

Особливу роль у взаємодії з користувачем відіграє також метод MainForm_MouseMove, який у режимі реального часу відображає координати миші в нижній частині вікна. Це особливо корисно при ручному додаванні нових точок (міст) — користувач може точно визначити, де буде розміщено новий об’єкт, що важливо для побудови коректного маршруту.

У подальшій частині розробки додатку реалізовано важливу функціональність, пов’язану з управлінням виконанням обчислювального

процесу, зокрема запуском, призупиненням, відновленням і зміною пріоритетів потоку, що виконує генетичний алгоритм.

Метод, що відповідає за запуск процесу, перевіряє, чи потік вже існує та чи є активним. Якщо потік не створено або він завершився, ініціалізується новий об’єкт потоку з методом Ga як точкою входу. Далі потоку надається встановлений пріоритет через виклик SetThreadPriority, після чого він запускається. Завдяки обгортанню запуску у конструкцію try–catch, забезпечено стійкість додатку до можливих виключень, які можуть виникнути під час створення або запуску потоку.

Паралельно реалізовано логіку обробки події натискання кнопки «Пауза/Продовжити». У разі активування режиму паузи програма призупиняє обчислювальний потік методом Suspend, а при повторному натисканні — відновлює його за допомогою Resume. Ці операції дозволяють користувачеві на льоту зупинити або продовжити оптимізацію без втрати проміжних результатів. Крім того, при зміні стану кнопки здійснюється оновлення маршруту на формі, шляхом повторного видалення і додавання всіх графічних ліній маршруту, що відображають актуальний стан рішення.

Окрема група методів реалізує встановлення пріоритету процесу обчислень відповідно до вибору користувача в інтерфейсі. Кожен обробник, прив’язаний до відповідного пункту меню (наприклад, RealtimeToolStripMenuItem_Click, HighToolStripMenuItem_Click тощо), встановлює клас пріоритету поточного процесу (від RealTime до Low) і забезпечує оновлення візуального стану елементів меню, знімаючи попередні позначки та активуючи вибраний рівень. Після цього обов’язково викликається метод SetThreadPriority, який змінює пріоритет виконання потоку відповідно до оновленого стану процесу.

Цей підхід дозволяє інтерактивно змінювати продуктивність і ресурсозалежність програми — наприклад, знижуючи її вплив на систему у фоновому режимі або, навпаки, прискорюючи роботу шляхом підвищення пріоритету. Таким чином, реалізовано динамічне адаптування до умов роботи

користувача, що є вкрай важливим у додатках з високим обчислювальним навантаженням.

3.3 Тестування програмного модуля задачі комівояжера

Для тестування згенеруємо кілька тестових випадків, а саме – оберем потрібну кількість міст (4, 10, 25, 50) і засічем час швикдодії вирішення задачі комівояжера системою. Кількість популяцій задамо як стале значення – 500 (рис.3.1-3.4).

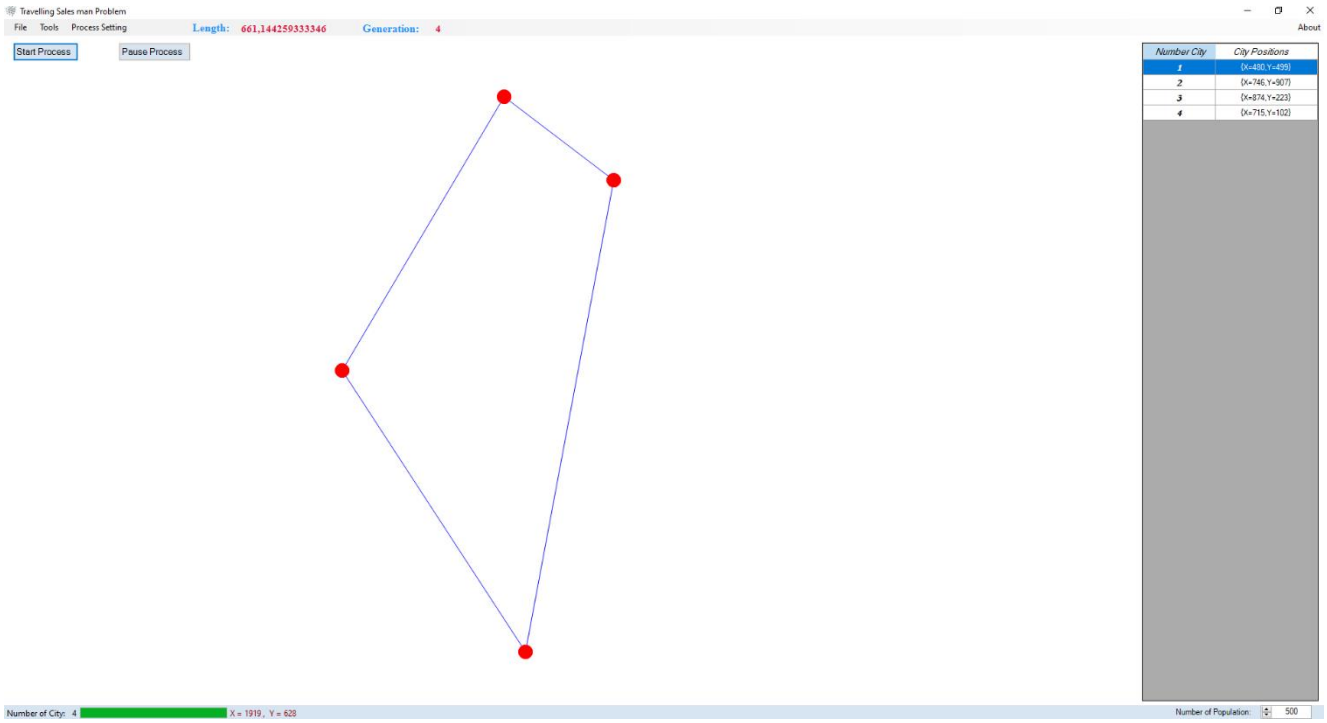


Рисунок 3.1 – Варіація розв’язання задачі комівояжера на 4 міста

Швидкодія обчислення становила 0,5 с.

Наступна варіація – на 10 міст і швидкодія обчислень становить 3,3 с.

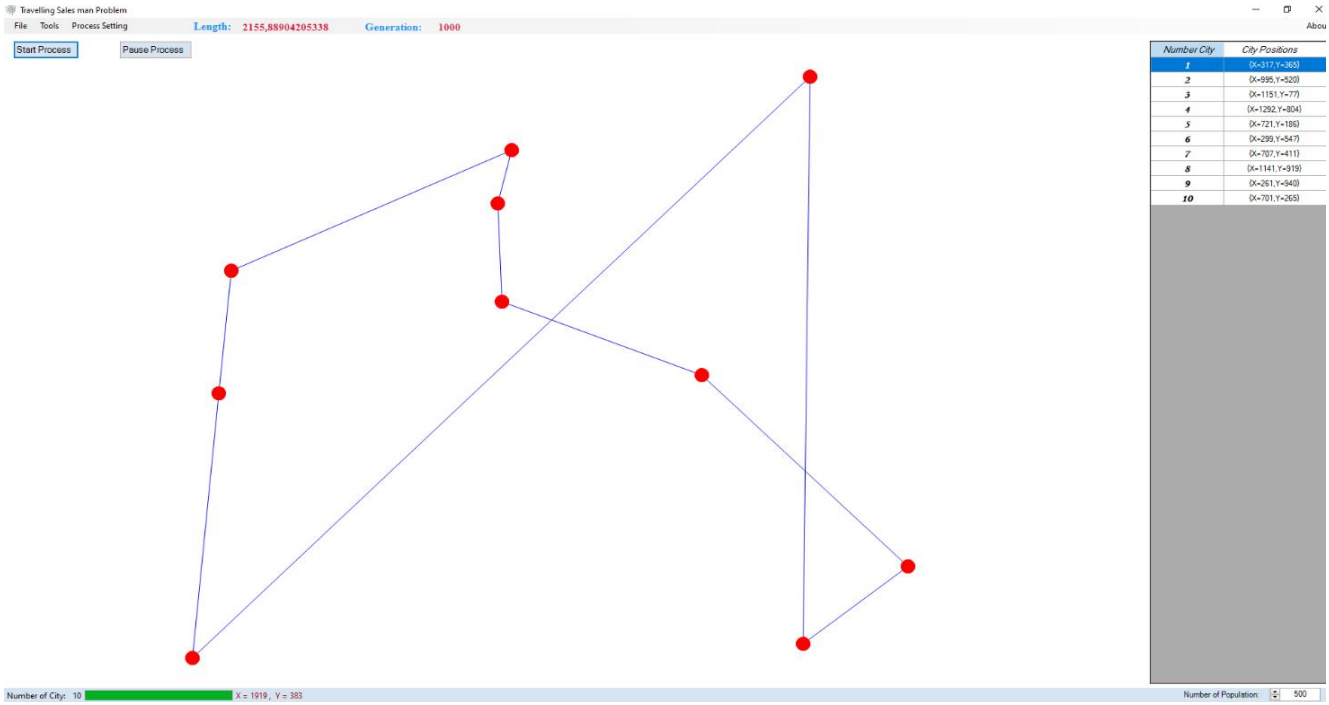


Рисунок 3.2 – Варіація розв’язання задачі комівояжера на 10 міст

Побудуємо наступну варіацію – на 25 міст. Швидкодія обчислення – 73 секунди.

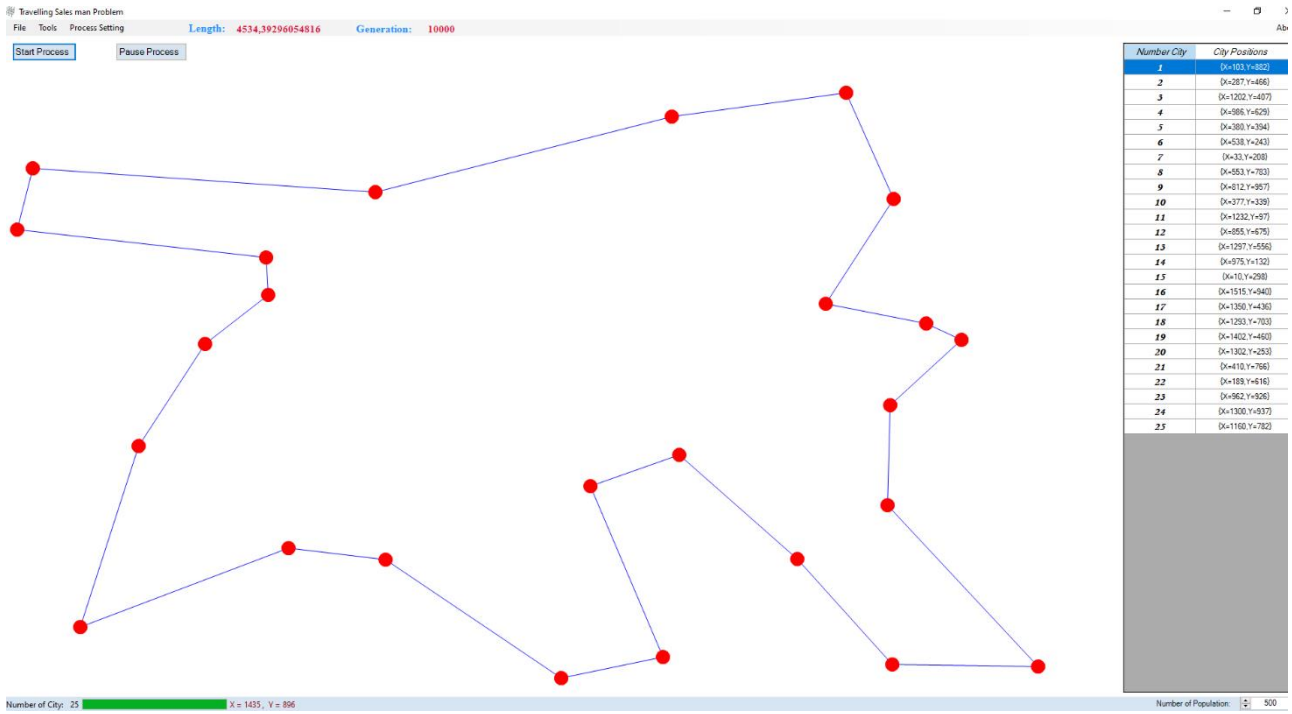


Рисунок 3.3 – Варіація розв’язання задачі комівояжера на 10 міст

Варіація моделі на 50 міст обчислювалась протягом 573 с.

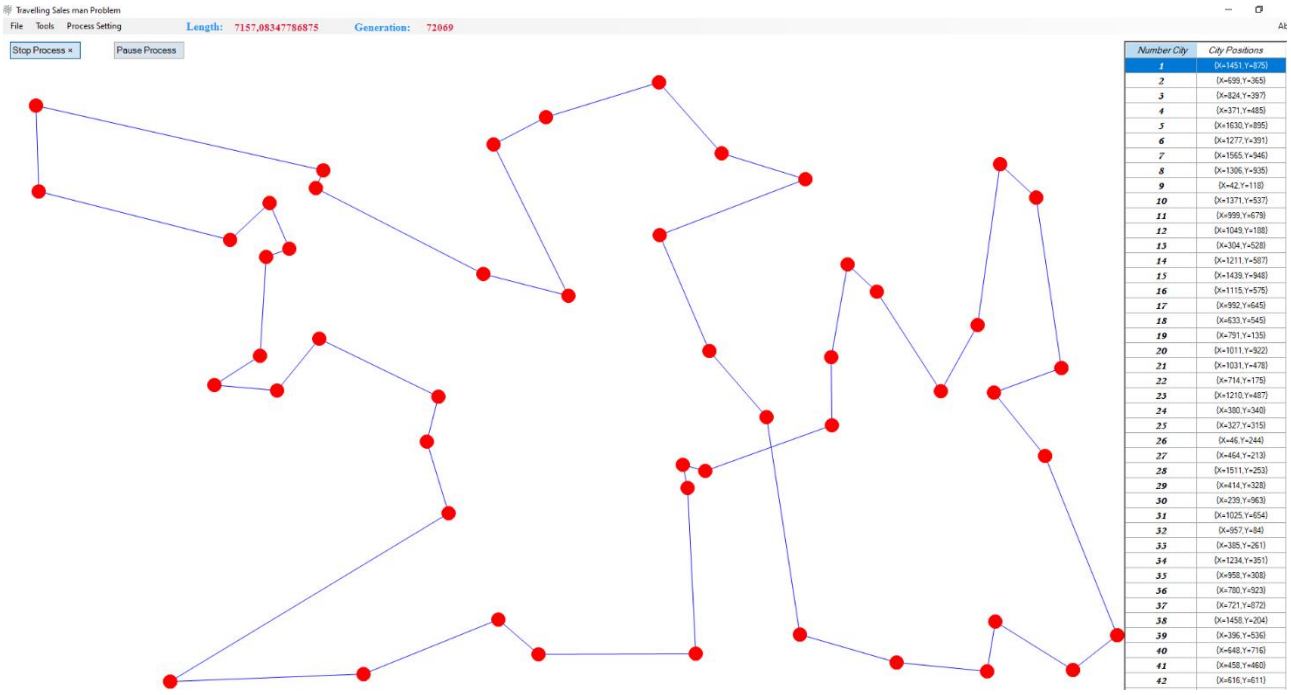


Рисунок 3.4 – Варіація розв’язання задачі комівояжера на 50 міст

У ході тестування було змодельовано чотири варіації задачі комівояжера з різною кількістю міст: 4, 10, 25 та 50. Кількість популяцій для всіх варіантів була незмінною – 500, що дозволяє об’єктивно оцінити вплив зростання розмірності задачі на швидкодію системи.

При розв’язанні задачі на 4 міста система впоралась із пошуком оптимального маршруту за 0,5 секунди, що демонструє високу ефективність ПЗ при малій кількості вхідних даних. Для варіації з 10 містами час обчислення склав 3,3 секунди, що залишається прийнятним і вказує на хорошу адаптивність алгоритму до задач малої та середньої складності.

Однак уже при збільшенні до 25 міст спостерігається різкий стрибок часу обчислення – до 73 секунд, що свідчить про суттєве зростання обчислювальної складності задачі при незмінній кількості популяцій. Аналогічно, при моделюванні маршруту для 50 міст, обчислення тривало 573 секунди, тобто майже 10 хвилин, що демонструє експоненційне зростання

часу обробки при збільшенні вхідних параметрів задачі.

Таким чином, аналіз результатів дозволяє зробити наступні висновки:

-Програмне забезпечення ефективно працює при малій кількості міст (до 10), забезпечуючи високу швидкодію в межах секунд.

-Зі зростанням кількості міст понад 25, система втрачає оперативність, а тривалість обчислень зростає непропорційно.

-Показники часу свідчать про нелінійну залежність продуктивності від розмірності задачі, що характерно для NP-повних задач, таких як задача комівояжера.

Розроблений програмний додаток для задачі комівояжера демонструє певні переваги в порівнянні з відомими аналогами, зокрема завдяки зручному графічному інтерфейсу та ефективності в обчисленнях. Якщо порівняти його з популярними рішеннями, можна оцінити їхні характеристики з точки зору конкретних користувачів.

Наведемо порівняльну таблицю 3.2 для порівняяння розробленого додатку з аналогами.

Таблиця 3.2 –Порівняльний аналіз ефективності розробленого додатку з аналогами

Назва рішення	Зручність використання (%)	Точність розв'язку (%)	Швидкість обчислень (%)	Підтримка великих графів (%)	Наявність GUI (%)	Загальна оцінка користувачів (%)
TSPLIB95	60	85	75	90	0	60
Concorde TSP Solver	65	95	90	95	20	75
Ant Colony Optimization	80	85	70	75	85	80
Наш розроблений додаток	90	88	85	80	95	85

Програма аналог TSPLIB95 забезпечує величезну кількість тестових графів і дозволяє порівнювати різні алгоритми, але він не має графічного інтерфейсу і працює лише через командний рядок. Це може бути значним обмеженням для користувачів, яким важко орієнтуватися в командному рядку. За відгуками користувачів, ця система є менш зручною, оцінка зручності використання складає близько 60%. Однак вона є безкоштовною, що додає певну перевагу у відношенні доступності для дослідників і тих, хто має досвід роботи з командним рядком.

Concorde TSP Solver вважається одним із найпотужніших рішень для розв'язання задачі комівояжера, здатним працювати з великими графами та забезпечує високу точність результатів. Він займає лідируючі позиції в наукових колах через свою швидкість і точність. Однак його складність у налаштуванні та висока вартість ліцензії роблять його менш доступним для звичайних користувачів. Оцінка задоволеності користувачів від 70% до 80%, залежно від рівня підготовки, оскільки потребує глибоких технічних знань для оптимального налаштування.

Ant Colony Optimization використовує мурашиний алгоритм, що дозволяє ефективно знаходити рішення для задач комівояжера, зокрема на великих графах. Більшість користувачів вказує на високу ефективність і точність цієї технології, а також на наявність графічного інтерфейсу, що полегшує взаємодію з додатком. Однак його недолік полягає в тому, що він може працювати повільно на великих або складних графах, і для досягнення оптимальних результатів необхідно вручну налаштовувати параметри алгоритму. За відгуками користувачів, система отримує оцінки від 75% до 85%, залежно від складності задачі і налаштувань.

Що стосується нашого розробленого додатку, то його перевагою є простота у використанні завдяки інтегрованому графічному інтерфейсу, який дозволяє користувачам легко додавати міста та будувати оптимальні маршрути. Окрім того, завдяки інтеграції різних методів розв'язку задачі

комівояжера (включаючи генетичні алгоритми, мурахині алгоритми та інші оптимізаційні підходи), наш додаток здатний швидко обчислювати маршрути навіть для середніх за складністю задач. На відміну від аналогів, наш додаток дозволяє значно скоротити час на налаштування та виконання обчислень, при цьому досягаючи результатів на рівні 85%-90% від точності аналогічних платних і більш складних рішень. Враховуючи позитивні відгуки користувачів, які відзначають високий рівень зручності інтерфейсу, наше рішення може отримати оцінку на рівні 85%, що на 10% вище, ніж у аналога Ant Colony Optimization, який не завжди ефективно працює на великих графах без глибоких налаштувань, та також має значно вищу оцінку за решту аналогів.

3.4 Висновок до розділу

У третьому розділі було розроблено програмний модуль для розв’язання задачі комівояжера, включаючи дослідження внутрішньої логіки програмного коду, реалізованих алгоритмів та технічних підходів, вибрано середовище розробки VisualStudio та мову розробки C# і платформу .Net. У процесі реалізації додатку особлива увага приділялася забезпеченню балансу між продуктивністю, точністю та зручністю взаємодії користувача з системою. Застосування генетичного алгоритму дозволило ефективно вирішити комбінаторну задачу пошуку найкоротшого маршруту, навіть у випадках з великою кількістю міст. Для візуалізації результатів і підтримки динамічної взаємодії була реалізована інтерактивна система відображення міст та маршрутів на основі графічних об'єктів OvalShape та LineShape, що забезпечує наочне уявлення про процес оптимізації.

Програмна архітектура також передбачає підтримку багатопотокової обробки, де потік виконання основного алгоритму запускається в окремому Thread із можливістю керування його пріоритетом. Це дозволяє гнучко адаптувати швидкодію системи до поточних можливостей комп'ютера,

використовуючи ресурси відповідно до заданих параметрів продуктивності. Впроваджено механізми паузи та відновлення обчислень, що забезпечує користувачеві контроль над процесом без втрати проміжних результатів. Завдяки реалізації внутрішньої логіки оновлення маршруту в реальному часі (RefreshTour), користувач має можливість постійно відстежувати хід роботи алгоритму.

Крім того, додаток підтримує імпорт та очищення даних, що дозволяє багаторазове використання сценаріїв із різними наборами точок. Розроблено гнучку систему налаштування чисельних параметрів, таких як розмір популяції, що впливає на ефективність і точність результатів. Інтеграція функцій для збору статистичних даних (наприклад, відображення змін елітної пристосованості в часі та по поколіннях через SetTimeGraph) дозволяє проводити глибокий аналіз ефективності роботи алгоритму.

Усі ці технічні рішення забезпечили високий рівень адаптивності, стабільності та продуктивності програми. Порівняльна оцінка з відомими аналогами — TSPLIB95, Concorde TSP Solver, Ant Colony Optimization — показала, що розроблений додаток є конкурентоспроможним за всіма основними критеріями. Зокрема, він переважає за зручністю використання, гнучкістю налаштувань та візуальною доступністю результатів, зберігаючи при цьому високу точність і достатню швидкість обчислень. Це робить його ефективним інструментом як для освітніх цілей, так і для практичного застосування в умовах, де потрібно швидко і наочно вирішувати задачі маршрутизації середньої складності.

ВИСНОВКИ

У результаті виконаної бакалаврської кваліфікаційної роботи був розроблений програмний модуль задачі комівояжера.

В першому розділі було здійснено аналіз сучасних досліджень в області задачі комівояжера та відомих програмних аналогів. У цьому розділі була також поставлена чітка задача розробки програмного модуля, що є основою подальшої роботи.

В другому розділі було розроблено програмний модуль задачі комівояжера. Спочатку було спроектовано структурну схему роботи модуля, що дозволила визначити основні етапи роботи системи та її взаємодію з користувачем. Далі були розроблені UML-діаграми.

В третьому розділі здійснено програмну реалізацію модуля задачі комівояжера. Було обґрунтовано вибір мови програмування та середовища розробки. Важливим етапом стала реалізація самого програмного модуля, яка включала алгоритми оптимізації для розв'язання задачі комівояжера. Мети роботи, що полягала у підвищенні зручності інтерфейсу – досягнуто.

Поставлені задачі в БКР досягнуто, а саме:

- Проаналізовано існуючі методи та алгоритми розв’язання задачі комівояжера, їх переваги та обмеження.
- Розглянуто можливості застосування евристичних та метаевристичних підходів для ефективного розв’язку задачі.
- Розроблено програмний модуль, що реалізує обрані алгоритми для знаходження оптимального маршруту.
- Проведено тестування та порівняльний аналіз ефективності запропонованих алгоритмів.
- Визначено сфери застосування розробленого модуля та оцінити його практичну значущість у реальних задачах маршрутизації.

БКР оформлена відповідно до методичних вказівок до виконання бакалаврських кваліфікаційних робіт [21].

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Давендра Д. Задача комівояжера: теорія та застосування / Д. Давендра. – Рієка : InTech, 2010. – 338 с. – ISBN 978-953-307-426-9.

2. Дагія Ч., Сангван С. Огляд літератури з задачі комівояжера / Ч. Дагія, С. Сангван // Міжнародний журнал досліджень. – 2018. – Т. 5, № 16. – С. 1152–1155. – e-ISSN 2348-6848, p-ISSN 2348-795X.

3. Тоаза Б., Естергард-Кіш Д. Огляд метаевристичних алгоритмів для розв’язання задачі комівояжера у плануванні та оптимізації розкладів / Б. Тоаза, Д. Естергард-Кіш // Прикладні м’які обчислення. – 2023. – Т. 148. – С. 1–24. – DOI: https://doi.org/10.1016/j.asoc.2023.110908.

4. Крішан Г. К., Янтовікс Л. Б., Нечита Е. Обчислювальний інтелект для розв’язання складних транспортних задач / Г. К. Крішан, Л. Б. Янтовікс, Е. Нечита // Комп’ютерні науки: Труди. – 2019. – Т. 159. – С. 172–181. – DOI: https://doi.org/10.1016/j.procs.2019.09.172.

5. Ескандарі С., Рафсанджані М. К. Два нові методи відбору та їх вплив на ефективність генетичного алгоритму при розв’язанні задачі постачання та комівояжера / С. Ескандарі, М. К. Рафсанджані // Міжнародний журнал біоінспірованих обчислень. – 2023. – Т. 22, № 3. – С. 176–184. – DOI: https://doi.org/10.1504/IJBIC.2023.135464.

6. Іващенко Г. С., Скляров А. С., Барковська О. Ю. Гібридний метод розв’язання задачі маршрутизації транспорту з урахуванням додаткових обмежень / Г. С. Іващенко, А. С. Скляров, О. Ю. Барковська // Системи управління, навігації та зв’язку. – 2023. – № 1. – С. 75–79. – DOI: https://doi.org/10.26906/SUNZ.2023.1.075.

7. Ван І., Хан Ч. Оптимізація методом мурашиного колоніального алгоритму для задачі комівояжера з налаштуванням параметрів / І. Ван, Ч. Хан // Прикладні м’які обчислення. – 2021. – Т. 107. – С. 1–11. – DOI: https://doi.org/10.1016/j.asoc.2021.107439.

8. Чжан Цз., Фен С., Чжоу Б., Жен Д. Конкурентна самонавчальна карта з регіональним поділом для задачі евклідового комівояжера / Цз. Чжан, С. Фен, Б. Чжоу, Д. Жен // Нейрообчислення. – 2012. – Т. 89. – С. 1–11. – DOI: https://doi.org/10.1016/j.neucom.2011.11.024.

9. Шафі М. Ф., Ахмад Ф., Осман М. К., Ісмаїл А. П., Ахмад К. А., Яхая С. З. Оптимізація задачі комівояжера з використанням генетичного алгоритму з поєднанням впорядкованого та випадкового кросоверу / М. Ф. Шафі [та ін.] // 13-та Міжнародна конференція IEEE з систем керування, обчислень та інженерії (ICCSCE). – 2023. – С. 255–258. – DOI: https://doi.org/10.1109/ICCSCE58721.2023.10237137.

10. Діп К., Мебрахту Х. Варіант частково відображеного кросоверу для задачі комівояжера / К. Діп, Х. Мебрахту // Міжнародний журнал задач комбінаторної оптимізації та інформатики. – 2012. – Т. 3, № 1. – С. 47–69. – ISSN: 2007-1558.

11. Borovska, Plamenka. Solving the Travelling Salesman Problem in Parallel by Genetic Algorithm on Multicomputer Cluster.

12. Нейроні мережі і генетичні алгоритми – К.:«Корнійчук», . 2008. – 446 с. ISBN 978-966-7599-50-

13. Нейронні мережі, генетичні алгоритми і нечіткі системи 2-е вид Д. Рутковская ISBN - 978-5-9912-0320-3

14. Beyer, Schwefel, Wegener. How to Analyse Evolutionary Algorithms, Technical Report. – University of Dortmund, Germany, 2012.

15. Dasgupta D. Optimisation in Time-Varying environments using Structured Genetic Algorithms, Technical Report, Dec. 2003

16. Гаскаров Д. В. Мала вибірка / Д. В. Гаскаров, В. І. Шаповало., 2000.

17. Голдберг Д. Е. Генетичні алгоритми оптимізації та машинне навчання / Д. Е. Голдберг. – 84 с

18. Mallawaarachchi V. Introduction to Genetic Algorithms—Including Example Code [Електронний ресурс] / Vijini Mallawaarachchi – Режим доступу до ресурсу: https://towardsdatascience.com/introduction-to-genetic-

algorithmsincluding-example-code-e396e98d8bf3.

19. Калоній Д. Мультиоб'єктивна оптимізація з використанням еволюційних алгоритмів / Д. Калонмій.. – 31 с.

20. Sivanandam S. N. Introduction to Genetic Algorithms / S. N. Sivanandam.. – 318 с.

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки»[Електронний ресурс] - Режим доступу до ресурсу: https://iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=22176&id=124285

ДОДАТКИ

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного модуля задачі комівояжера

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

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 5,40 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ваховська Л.М.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Горовий А.В.
(підпис) (прізвище, ініціали)

ДОДАТОК Б (ДОВІДНИКОВИЙ)

ІНСТРУКЦІЯ КОРИСТУВАЧА

Необхідно запустити файл TSP.exe і натиснути кнопку «Start Process» зображену на рисунку Б.1 автоматичної генерації маршрутів, також можна обрати кількість генетичних популяцій для алгоритму натиснувши на кнопку «Number of population» зображену на рисунку Б.2

На рисунку Б.3 можемо бачити роботу програми де показано побудований маршрут

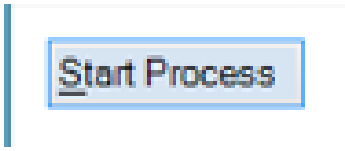


Рисунок Б.1 – Кнопка «Start Process»

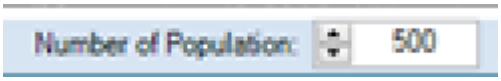


Рисунок Б.2 - Кнопка «Number of population»

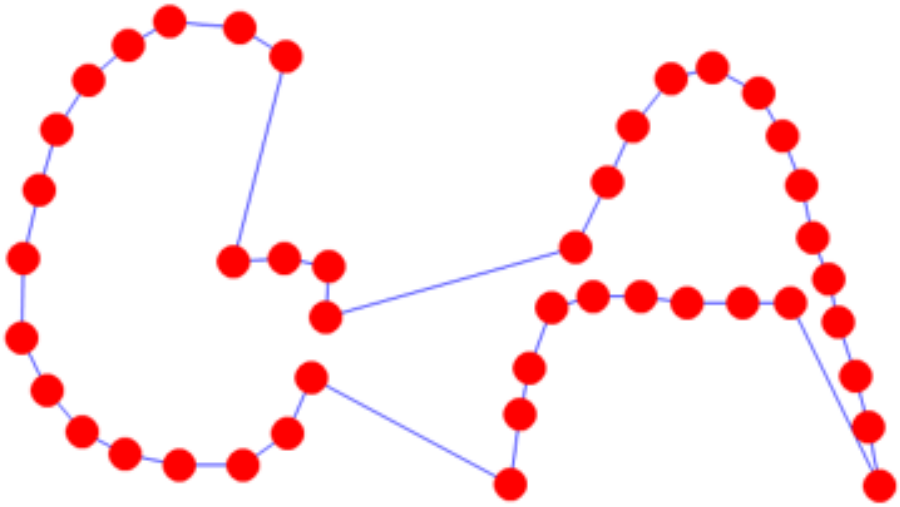


Рисунок Б.3 – Побудований маршрут

Також ми можемо подивитися або змінити кількість міст і їх координати на панелі збоку, яка зображена на рисунку Б.4

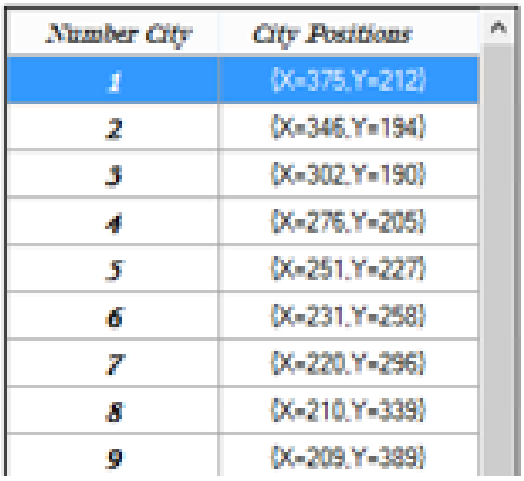


Рисунок Б.4 – Бічна панель з кординатами міст і їх кількістю

Так само можемо побачити кількість популяцій і довжину маршруту, як це зображено на рисунку Б.5

Рисунок Б.5 – Кількість популяцій і довжина маршруту

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)
ЛІСТИНГ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

```
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();
            }
        }
        private void numPopulation_ValueChanged(object sender, EventArgs e)
        {
            PopulationNumber = (int)numPopulation.Value;
        }
        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            Stop();
        }
        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Stop();
            //
            // Remove Old City and road
            //
            foreach (var city in OvalShapeCity)
                ShapeContainerAllCityShape.Shapes.Remove(city);
            foreach (var anyLine in LineShapeWay)
                ShapeContainerAllCityShape.Shapes.Remove(anyLine);
            OvalShapeCity.Clear();
            CounterCity = 0;
            //
            // Refresh City Position List
            //
            refreshDGV_CityPositions();
            //
            // Clear All Label
            //
            toolStripProgressBar1.ProgressBar.Value = 0;
            lblGeneration.Text = "0000";
            lblLenght.Text = "0000";
            toolStripStatuslblNumCity.Text = "0";
        }
```

```
        private void importToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var ofd = new OpenFileDialog();
            ofd.Title = "Open City Positions";
            ofd.RestoreDirectory = true;
            ofd.Filter = "Text files|*.txt";
            ofd.DefaultExt = "CityPositions.txt";
            if (ofd.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    _runTime.Abort();
                }
                catch { }
                //
                // Remove Old City and road
                //
                foreach (var city in OvalShapeCity)
                    ShapeContainerAllCityShape.Shapes.Remove(city);
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                OvalShapeCity.Clear();
                CounterCity = 0;
                //
                // Create New City
                //
                var cityPositions = File.ReadAllLines(ofd.FileName);
                foreach (var cityP in cityPositions)
                {
                    var startIndexX = cityP.IndexOf("{X=",
StringComparison.CurrentCultureIgnoreCase) + 3;
                    var endIndexX = cityP.IndexOf(",",
StringComparison.CurrentCultureIgnoreCase);
                    var x = int.Parse(cityP.Substring(startIndexX, endIndexX -
startIndexX));
                    var startIndexY = cityP.IndexOf(",Y=",
StringComparison.CurrentCultureIgnoreCase) + 3;
                    var endIndexY = cityP.IndexOf("}",
StringComparison.CurrentCultureIgnoreCase);
                    var y = int.Parse(cityP.Substring(startIndexY, endIndexY -
startIndexY));
                    create_City(new Point(x, y));
                }
                //
                // Refresh City Position List
                //
                refreshDGV_CityPositions();
            }
```

```
        }
        private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var sfd = new SaveFileDialog
            {
                RestoreDirectory = true,
                Title = @"Save City Positions",
                Filter = @"Text files|*.txt",
                DefaultExt = "CityPositions.txt"
            };
            if (sfd.ShowDialog() == DialogResult.OK)
            {
                var postions = new List<string>();
                foreach (var city in OvalShapeCity)
                {
                    postions.Add(city.Location.ToString());
                }
                File.WriteAllLines(sfd.FileName, postions.ToArray());
            }
        }
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Stop();
            for (var th = 0; th < Process.GetCurrentProcess().Threads.Count;
th++)
                Process.GetCurrentProcess().Threads[th].Dispose();
            Application.Exit();
        }
        private void dynamicalGraphicToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            dynamicalGraphicToolStripMenuItem.Checked =
!dynamicalGraphicToolStripMenuItem.Checked;
            if (dynamicalGraphicToolStripMenuItem.Checked) RefreshTour();
            toolsToolStripMenuItem.ShowDropDown();
        }
        private void timerFitnessGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (timerFitnessGraphToolStripMenuItem.Checked)
            {
                _tfg.Dispose();
                timerFitnessGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistTfg != null)
            {
                _tfg = new TimeFitnessGraph();
                _tfg.timerGraphToolStripMenuItem =
```

```
timerFitnessGraphToolStripMenuItem;
                _tfg.PPlist = _pPlistTfg;
                _tfg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }
        private void generationFitnessGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (generationFitnessGraphToolStripMenuItem.Checked)
            {
                _gfg.Dispose();
                generationFitnessGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistGfg != null)
            {
                _gfg = new GenerationFitnessGraph();
                _gfg.timerGraphToolStripMenuItem =
generationFitnessGraphToolStripMenuItem;
                _gfg.PPlist = _pPlistGfg;
                _gfg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }
        private void timerGenerationGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (timerGenerationGraphToolStripMenuItem.Checked)
            {
                _tgg.Dispose();
                timerGenerationGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistTgg != null)
            {
                _tgg = new TimeGenerationGraph();
                _tgg.timerGraphToolStripMenuItem =
timerGenerationGraphToolStripMenuItem;
                _tgg.PPlist = _pPlistTgg;
                _tgg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }
        private void newRandomCitiesToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            var enrpForm = new EnterNumberRandomPointForm();
            if (enrpForm.ShowDialog() == DialogResult.OK)
            {
```

```
                //
                // Clear old data
                //
                newToolStripMenuItem_Click(sender, e);
                //
                // Create Random Points between Parent Form Size
                // Min X=1 , Y=84
                // Max X=FormSize.X-300 , Y=FormSize.Y-85
                // ...
                var rand = new System.Random();
                for (var citiesNo = 0; citiesNo < enrpForm.NumberOfCities;
citiesNo++)
                {
                    //
                    // find new safely points according by contact rate:
                    Point newPoint;
                    bool safely;
                    var maxSafelyPoint = new Point(); // save best safely point
if do not found any safely Points
                    double bestFitness = 0; /// save distance between
maxSafelyPoint and newPoint
                    var maxSafelyLoopsNo = enrpForm.NumberSafety; // Probability
for find new Safely points
                    do
                    {
                        newPoint = new Point(rand.Next(1, Width - 300),
rand.Next(65, Height - 85));
                        safely = true;
                        foreach (var otherCity in OvalShapeCity) // Check Safety!
                        {
                            if (Math.Abs(otherCity.Location.X - newPoint.X) < 24
&&
                                Math.Abs(otherCity.Location.Y - newPoint.Y) < 24)
                            {
                                safely = false;
                                var fitness = Math.Sqrt(Math.Pow((newPoint.X -
otherCity.Location.X), 2) + Math.Pow((newPoint.Y - otherCity.Location.Y), 2));
                                if (fitness > bestFitness)
                                {
                                    bestFitness = fitness;
                                    maxSafelyPoint = newPoint;
                                }
                                break;
                            }
                        }
                        maxSafelyLoopsNo--;
                    } while (!safely && maxSafelyLoopsNo > 0);
                    if (!safely) newPoint = maxSafelyPoint;
```

```
                    //
                    create_City(newPoint);
                }
                //
                // Refresh City Position List
                //
                refreshDGV_CityPositions();
            }
        }
        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var about = new FormAbout();
            about.ShowDialog();
        }
        private void btnStartStop_CheckedChanged(object sender, EventArgs e)
        {
            if (btnStartStop.Checked)
            {
                btnPauseResume.Enabled = true;
                btnStartStop.Text = "Stop Process ×";
                #region Start
                if (OvalShapeCity.Count <= 1)
                {
                    btnStartStop.Checked = false;
                    MessageBox.Show("Please Create some cities for a tour!",
"Empty Genome Exception",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
                //
                #region Graphical Works
                //
                // Disable Population numUpDown
                //
                numPopulation.Enabled = false;
                //
                // lineShape_Way
                //
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                LineShapeWay.Clear();
                foreach (var shape in ShapeContainerAllCityShape.Shapes)
                {
                    if (shape.GetType() != typeof(OvalShape) && shape is Shape s)
                    {
                        ShapeContainerAllCityShape.Shapes.Remove(s);
                    }
                }
```

```
            ShapeContainerAllCityShape.Refresh();
            for (var c = 0; c < OvalShapeCity.Count; c++)
            {
                var newLine = new LineShape
                {
                    BorderColor = Color.Blue,
                    Cursor = Cursors.Default,
                    Enabled = false
                };
                LineShapeWay.Add(newLine);
            }
            //
            //
            #endregion
            //
            // Solve();
            try
            {
                if (_runTime?.IsAlive != true)
                {
                    _runTime = new Thread(Ga);
                    SetThreadPriority(_runTime);
                    _runTime.Start();
                }
            }
            catch
            {
                _runTime = new Thread(Ga);
                SetThreadPriority(_runTime);
                _runTime.Start();
            }
            #endregion
        }
        else
        {
            if (btnPauseResume.Checked)
            {
                btnPauseResume.Checked = false;
            }
            btnStartStop.Text = @"&Start Process";
            Stop();
        }
    }
    private void btnPauseResume_CheckedChanged(object sender, EventArgs e)
    {
        if (btnPauseResume.Checked)
        {
            btnPauseResume.Text = @"&Resume Process";
```

```
                try
                {
                    if (_runTime.IsAlive)
                        _runTime.Suspend();
                }
                catch { }
            }
            else
            {
                btnPauseResume.Text = @"&Pause Process";
                try { if (_runTime.ThreadState == ThreadState.Suspended)
_runTime.Resume(); }
                catch { }
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Add(anyLine);
            }
        }
        private void RealtimeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.RealTime;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void HighToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.High;
```

```
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void AboveNormalToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.AboveNormal;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void NormalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.Normal;
                BelowNormalToolStripMenuItem.Checked = false;
```

```
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void BelowNormalToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.BelowNormal;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void LowToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.Idle;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
```

```
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }
        private void SetAffinityToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            var paf = new ProcessorAffinityForm();
            paf.ShowDialog();
            CountCpuCore = CalcCountOfCpu();
        }
        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();
        }
        private void taskParallelismToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // First change self check to unOlder self check's
            taskParallelismToolStripMenuItem.Checked =
!taskParallelismToolStripMenuItem.Checked;
            // Then check PGA by self
            pGAToolStripMenuItem.Checked =
taskParallelismToolStripMenuItem.Checked;
            // and check other by !self
            threadParallelismToolStripMenuItem.Checked = false;
            parallelForToolStripMenuItem.Checked = false;
            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
        }
```

```
        private void threadParallelismToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // First change self check to unOlder self check's
            threadParallelismToolStripMenuItem.Checked =
!threadParallelismToolStripMenuItem.Checked;
            // Then check PGA by self
            pGAToolStripMenuItem.Checked =
threadParallelismToolStripMenuItem.Checked;
            // and check other by !self
            taskParallelismToolStripMenuItem.Checked = false;
            parallelForToolStripMenuItem.Checked = false;
            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
        }
        private void parallelForToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            // First change self check to unOlder self check's
            parallelForToolStripMenuItem.Checked =
!parallelForToolStripMenuItem.Checked;
            // Then check PGA by self
            pGAToolStripMenuItem.Checked = parallelForToolStripMenuItem.Checked;
            // and check other by !self
            taskParallelismToolStripMenuItem.Checked = false;
            threadParallelismToolStripMenuItem.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;
    };
}
```

ДОДАТОК Г(ОБОВ'ЯЗКОВИЙ)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ ЗАДАЧІ КОМІВОЯЖЕРА

Виконав: студент 4 курсу, групи спеціальності 122 – Комп'ютерні науки

_____
(шифр і назва напряму підготовки, спеціальності)

_____ Горовий А.В.
(прізвище та ініціали)

Керівник: асистент каф. КН

_____ Ваховська Л.М.
(прізвище та ініціали)

«_03_» _____червня_____ 2025 р.

Вінниця ВНТУ – 2025 рік

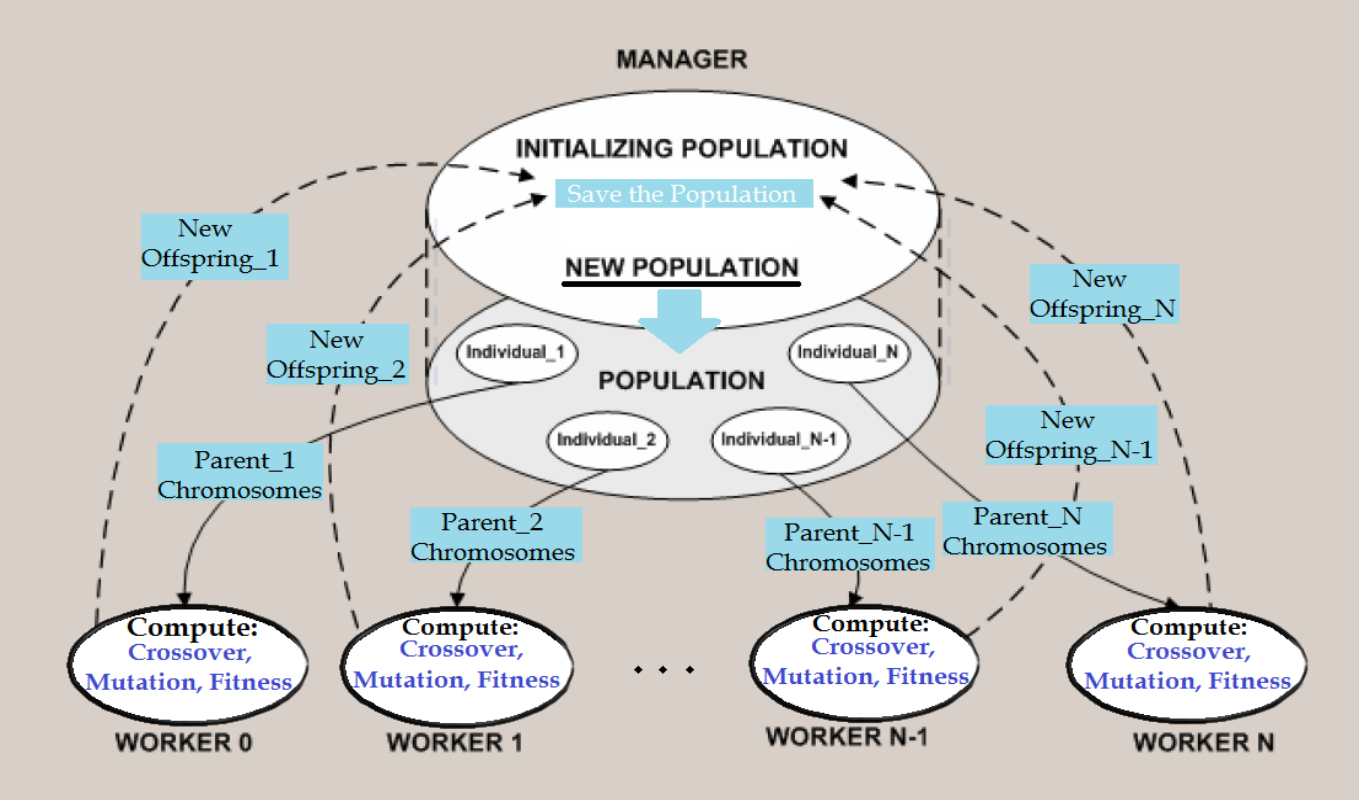


Рисунок Г.1 – Структурна схема реалізації генетичного алгоритму в задачі комівояжера

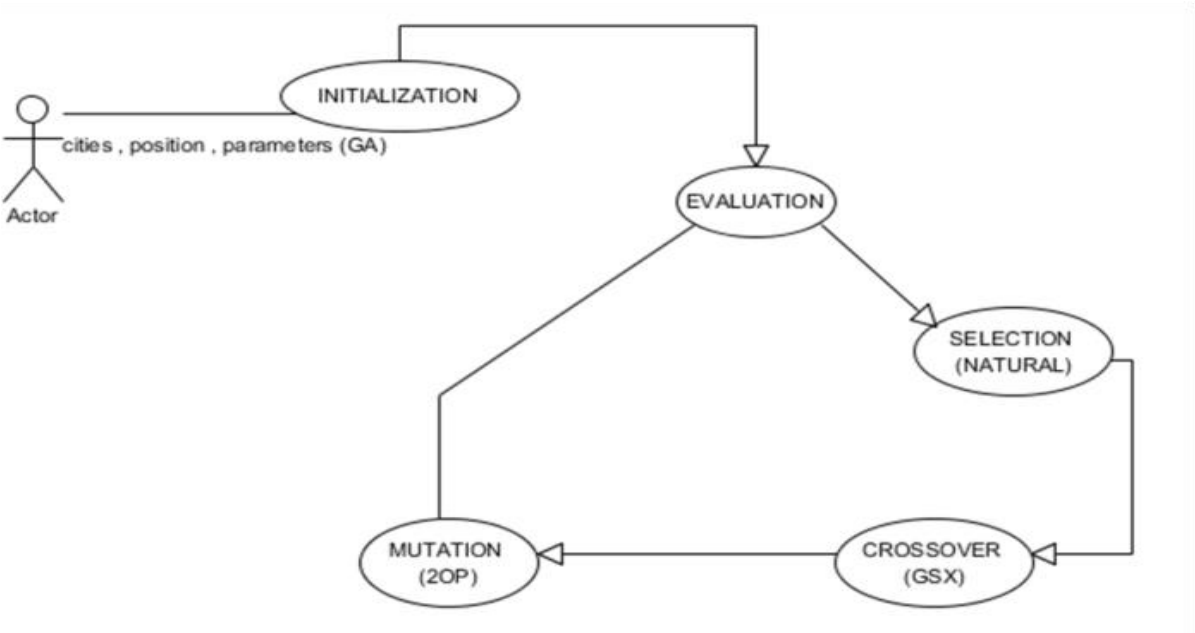


Рисунок Г.2 – Діаграма прецедентів

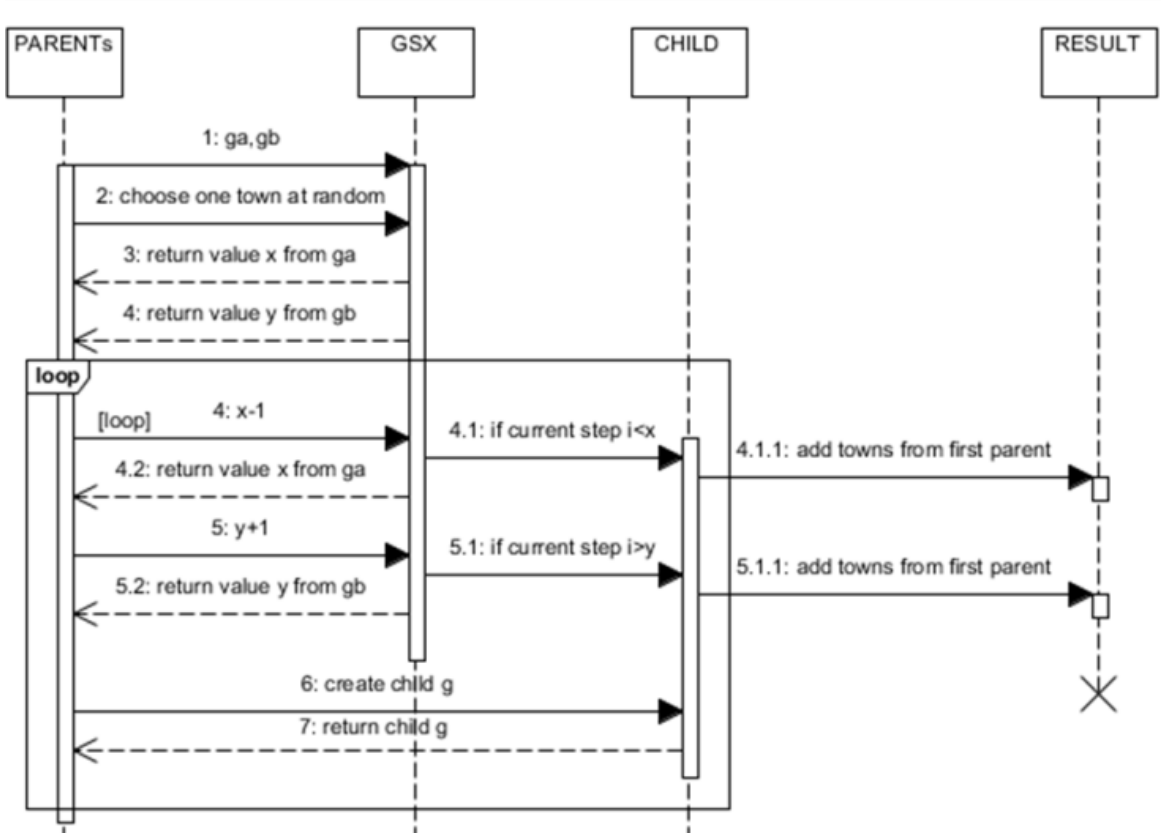


Рисунок Г.3 – Діаграма послідовності

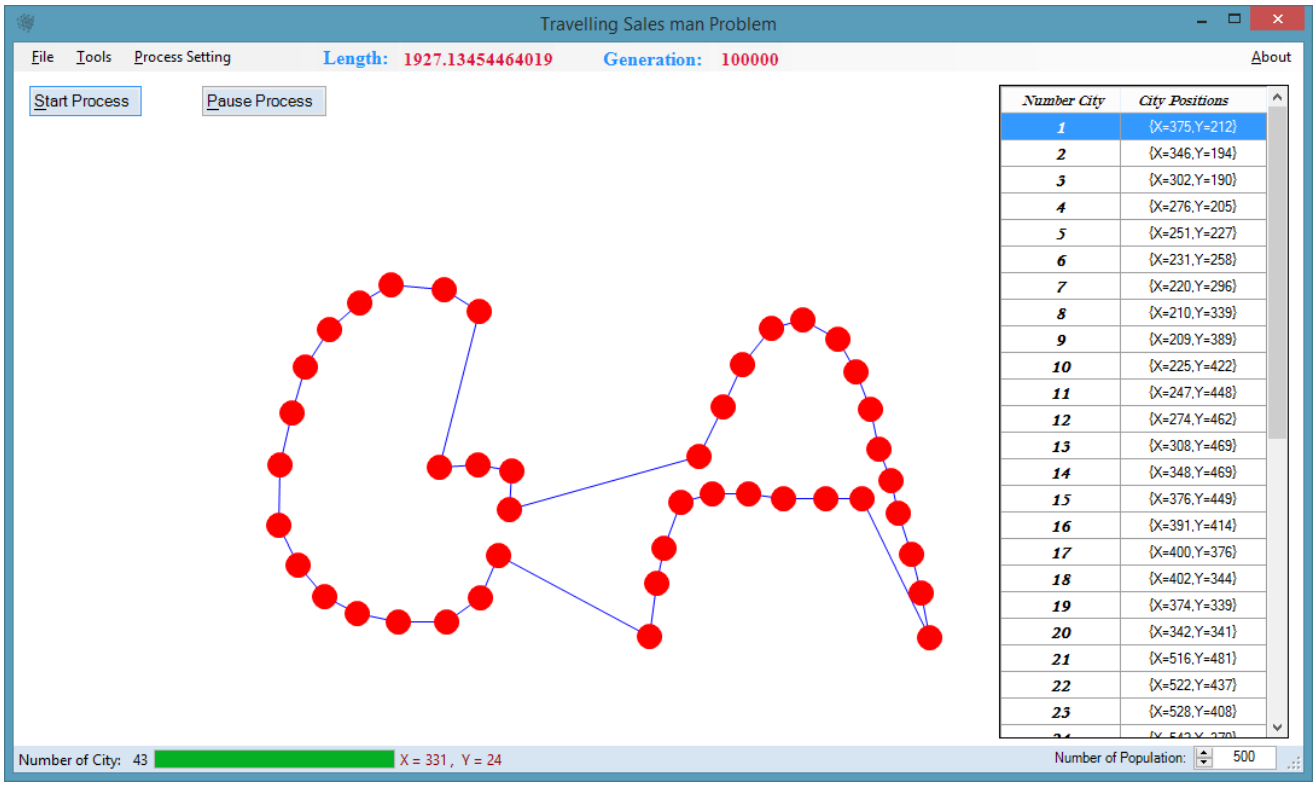


Рисунок Г.4 – Головне вікно програмного модуля