

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультет)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ГЕНЕТИЧНИХ
МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАВДАНЬ КОМБІНАТОРНОЇ ОПТИМІЗАЦІЇ
RESEARCH AND SOFTWARE IMPLEMENTATION OF GENETIC METHODS
FOR SOLVING THE COMBINATORIAL OPTIMIZATION PROBLEMS

Виконав: студент(ка) 2 курсу, групи КНТ-110м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Короткий Б.В.

(прізвище та ініціали)

Керівник Гофман Є.О.

(прізвище та ініціали)

Рецензент Коцур М.І.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
« » 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Короткого Богдана Володимировича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація генетичних методів розв'язання завдань комбінаторної оптимізації. Research and Software Implementation of Genetic Methods for Solving the Combinatorial Optimization Problems

керівник проєкту (роботи) Гофман Євгеній Олександрович, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «05» листопада 2021 року № 435

2. Строк подання студентом проєкту (роботи) 01 грудня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	Гофман Є.О., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання “3” вересня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз та дослідження предметної області	2-5 тижні	Розділи 1, 2
3	Розробка структури програми	6-7 тиждень	Розділ 2
4	Розробка програми	8-13 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	14-15 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	16 тиждень	Додатки
7	Захист роботи	16 тиждень	

Студент(ка)

(підпис)

Короткий Б.В.

(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Гофман Є.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра: 114 с., 32 рис., 27 табл., 4 дод., 22 джерела.

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ДИСКРЕТНА ОПТИМІЗАЦІЯ, ЗАДАЧА КОМІВОЯЖЕРА, МОВА ПРОГРАМУВАННЯ C#.

Об'єкт дослідження – процес комбінаторної оптимізації.

Предмет дослідження – генетичні методи розв'язання завдань комбінаторної оптимізації.

Метою роботи є дослідження та реалізація генетичних методів розв'язання завдань комбінаторної оптимізації.

Матеріали, методи та технічні засоби: мова програмування C#, середовище розробки Microsoft Visual Studio.

Результати. Розроблено програмне забезпечення, призначене для розв'язання задачі комбінаторної оптимізації з використанням методів генетичного пошуку.

Практична цінність роботи полягає в розробці програмного забезпечення для розв'язання практичної задачі комівояжера на основі генетичної оптимізації.

Висновки. Розроблено програмне забезпечення, що використовує генетичні алгоритми та може застосовуватися для розв'язання завдань дискретної оптимізації.

Галузь використання – інтелектуальні системи для логістики, розпізнавання образів, діагностування, прогнозування.

ABSTRACT

Explanatory note to the qualifying work of the master: 114 pages, 32 figures, 27 tables, 4 appendixes, 22 sources.

SOFTWARE, GENETIC ALGORITHM, DISCRETE OPTIMIZATION, TRAVELER'S TASK, C# PROGRAMMING LANGUAGE.

The object of research is the process of combinatorial optimization.

The subject of the research is genetic methods for solving combinatorial optimization problems.

The purpose of this work is to study genetic methods for solving combinatorial optimization problems.

Materials, methods and technical tools: C# programming language, Microsoft Visual Studio development environment.

Results. The software to solve the problem of combinatorial optimization using genetic search methods has been developed.

The practical value of the work lies in the development of software to solve the practical problem of the salesman on the basis of genetic optimization.

Conclusions. Software that implements that genetic algorithms for solving discrete optimization problems has been developed.

Scope of use – intelligent systems for logistics, pattern recognition, diagnosis, forecasting.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	10
1.1 Завдання комбінаторної оптимізації	10
1.2 Задача комівояжера	12
1.3 Аналіз методів розв’язання завдань комбінаторної оптимізації	13
1.3.1 Алгоритм повного перебору	14
1.3.2 Алгоритм «жадібного» пошуку	15
1.3.3 Метод гілок і меж	15
1.3.4 BV метод	16
1.3.5 Генетичний алгоритм	17
1.3.6 Порівняння відомих методів розв’язання завдань дискретної оптимізації	20
1.4 Висновки за розділом 1	22
2 Матеріали і методи	24
2.1 Вибір мови програмування	24
2.2 Вибір середовища розробки	26
2.3 Розроблення модифікованого генетичного методу для розв’язання задачі комбінаторної оптимізації	29
2.4 Структурна схема програми	33
2.5 Висновки за розділом 2	34
3 Опис програми	36
3.1 Схема функціонування програми	36
3.2 Модуль оптимізації маршруту	39
3.3 Проектування інтерфейсу	49
3.4 Висновки за розділом 3	
4 Експлуатація, тестування та експериментальне дослідження програми	52
4.1 Призначення й умови застосування програми	52

4.2	Характеристики програми	53
4.3	Інструкція по експлуатації програми.....	54
4.3.1	Звернення до програми.....	54
4.3.2	Вхідні й вихідні дані	54
4.3.3	Повідомлення.....	56
4.4	Приклад роботи з програмою	57
4.5	Експерименти та результати.....	60
4.6	Висновки за розділом 4	67
	Висновки	69
	Перелік джерел посилання	72
	Додаток А Технічне завдання	75
	Додаток Б Опис програми	80
	Додаток В Текст програми	85
	Додаток Д Слайди презентації	106

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

АКО	–	алгоритм комбінаторної оптимізації;
АПП	–	алгоритм повного перебору;
ГА	–	генетичний алгоритм;
ЗК	–	задача комівояжера;
ЗКО	–	задача комбінаторної оптимізації;
КО	–	комбінаторна оптимізація;
МГіМ	–	метод гілок і меж;
ПЗ	–	програмне забезпечення;
ПК	–	персональний комп'ютер;
ТЗ	–	технічне завдання.

ВСТУП

У наш час важливим завданням будь-якого суб'єкту (фізичної особи, підприємства або організації) є пошук резервів зниження витрат стосовно здійснюваної ним діяльності і, як наслідок цього, підвищення конкурентоспроможності та рентабельності [1]. В умовах сучасного світу пошук таких резервів часто заснований на логістичному підході, що передбачає зменшення витрат на транспортування товарів та продукції та є важливим інструментом забезпечення господарської діяльності. Важливим практичним завданням логістичного підходу є оптимізація перевезень, вирішення якого пов'язано з необхідністю розв'язання завдання дискретної оптимізації (мінімізації пройденого шляху, а значить і логістичних витрат, за умови необхідності відвідування певних транспортних вузлів, точок, складів тощо) [1].

Існують різні методи та алгоритми оптимізації транспортних витрат, але вони, як правило, є досить трудомісткі та мають високу часову залежність відповідно до розмірності задачі [1]-[11].

Виходячи з цього, актуальним є питання створення універсального інструменту для вирішення широкого кола задач комбінаторної оптимізації, впровадження якого дозволить користувачам отримати під час рішення задачі комівояжера наступні переваги:

- зазначити будь-яку кількість міст туру, що не вплине на час розв'язку задачі;
- побудувати найкоротший маршрут серед міст туру (отримати найкраще значення цільової функції пристосованості) за допомогою використання модифікованого алгоритму розв'язку, що засновується на еволюційному алгоритмі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Завдання комбінаторної оптимізації

Стрімке розширення сфер застосування методів математичного моделювання й оптимізації у сучасному світі призвело до того, що задачі комбінаторної оптимізації знаходять в наш час широке застосування в інформатиці, техніці, економіці, плануванні та інших областях, що постійно розширюються останніми роками, відповідно до чого формується потреба в розв'язуванні нових класів задач підвищеної складності та розмірності. В даний час в комбінаторній оптимізації інтенсивно розвивається підхід, заснований на принципі моделювання еволюційних процесів адаптації в живій природі. Еволюційні методи успішно застосовуються в інформаційних технологіях проектування, планування, управління, розпізнавання образів і т.д. [4]-[22].

Незважаючи на велику кількість результатів, отриманих в області комбінаторної оптимізації, потреба в подальших дослідженнях не зменшується. Це пов'язано як з постійним припливом нових завдань, так і зі складністю їх вирішення. Багато задач комбінаторної оптимізації є NP-важкими, і побудова оптимального рішення вимагає значних витрат часу навіть при порівняно низьких розмірностях вхідних даних. У таких ситуаціях виникає необхідність в більш глибокому теоретичному аналізі, що дозволить оцінити перспективи розробки алгоритмів із заданими початковими характеристиками і в підсумку знайти підхід до вирішення завдання, що зумовлює необхідність подальшого розвитку методів комбінаторної оптимізації та розробки нових математичних моделей оптимізаційних проблем, які можуть використовуватися в сучасних інформаційних технологіях [2]-[11].

Задачу дискретної оптимізації у загальному випадку можна подати за допомогою кортежу (1.1) [2]-[4]:

$$\langle f, X, \Pi, D, ext \rangle, \quad (1.1)$$

де $f: X \rightarrow R^1$ – цільова функція практичного завдання;

R^1 – множина можливих значень;

X – простір розв’язків практичного завдання (простір пошуку);

Π – предикат, що визначає підмножину $D \subseteq X$ допустимих розв’язків практичного завдання відповідно до заданих обмежень;

$ext \in \{min, max\}$ – тип завдання (напрямку) оптимізації.

У цих позначеннях завдання дискретної оптимізації можна подати у такому вигляді [2]-[4]: необхідно знайти $x_* \in D \subseteq X$ таке, що (1.2):

$$x_* = \arg f(x), \quad (1.2)$$

Під простором розв’язків будемо розуміти множину X , у якій введено деякі співвідношення між елементами, а предикат $\Pi(x)$ визначає множину припустимих розв’язків (1.3) [2]-[4]:

$$\Pi(x) = \{0, x \in D; 1, x \notin D, \quad (1.3)$$

Таким чином, при розв’язанні задач оптимізації необхідно знайти екстремальне значення цільової функції вигляду (1.4):

$$f(x) \rightarrow min, \quad (1.4)$$

Вирази (1.2) та (1.4) представляють собою різні задачі, проте вираз (1.4) може мати на увазі вирішення проблеми пошуку аргументу екстремуму (1.2), а не тільки значення цільової функції [2]-[4].

Для подання задачі оптимізації також може вживатися вираз (1.5):

$$f(x) \rightarrow ext. \quad (1.5)$$

Умови-обмеження задачі можуть подаватися як системи рівнянь абонерівностей.

Цільова функція задачі може визначатися аналітичним або іншим способом та конкретним набором вхідних даних $f(x) = f(x|c)$, де c – набір даних задачі (вхід задачі) [2].

1.2 Задача комівояжера

Задача комівояжера (ЗК) – найбільш відома задача комбінаторної оптимізації, яка є широко застосовуваною в областях управлінської діяльності для вирішення проблеми зниження логістичних витрат, пов'язаних з транспортуванням [2]-[11]. У зв'язку з актуальністю даної проблеми, виникла велика кількість задач, які можна об'єднати в загальний клас «транспортних». Особливе місце в цьому класі займає задача комівояжера, яка полягає в знаходженні найоптимальнішого маршруту, що проходить через задані пункти. Крім зазначеної актуальності практичного застосування задача комівояжера має серйозний теоретичний зміст: вона використовується в якості моделі для розробки евристичних та пошукових алгоритмів різних оптимізаційних задач [3].

Задачі комівояжера розрізняються за типом графів, які лежать в їх основі. Так, існують симетричні і асиметричні задачі комівояжера. Також, у відповідності до можливості додавання даних в процесі роботи алгоритму, ЗК діляться на статичні і динамічні [2]-[11].

Постановка задачі комівояжера передбачає, що дана множина з n міст, відстані між якими відомі. Необхідно знайти оптимальний маршрут, що проходить через задані міста рівно по одному разу. Математична постановка задачі є такою: дано граф $G = (V, E)$, де V – множина вершин, які ототожнюються з містами, E – множина ребер, d_{ij} – вага ребра $e \in E$. Необхідно знайти гамільтонів цикл мінімальної ваги, тобто замкнений цикл у

графі, що містить усі вершини та передбачає відвідування кожної вершини лише один раз [2]-[11].

Нехай $p = (p_1, \dots, p_n)$ – довільна перестановка всіх n елементів множини $\{1, \dots, n\}$; будемо вважати, що елемент p_i , $i = 1, \dots, n$, задає номер міста, яке відвідується на i -му етапі, а $p_{n+1} = p_1$ [2]-[11].

У комбінаторній постановці шукаємо таку перестановку, для якої досягається мінімум цільової функції (1.6) [2]-[11]:

$$\sum_{i=1}^n d_{p_i p_{i+1}} \rightarrow \min, \quad (1.6)$$

ЗК є класичним прикладом моделі для тестування методів штучного інтелекту й комп'ютерних наук взагалі [2].

1.3 Аналіз методів розв'язання завдань комбінаторної оптимізації

Існують різні підходи до класифікації методів комбінаторної оптимізації (АКО). Схему класифікації методів АКО наведено на рисунку 1.1 [2]-[11].



Рисунок 1.1 – Класифікація АКО за точністю

Точні АКО – це такі методи, які знаходять глобальний розв'язок, тобто для них $X_* \subseteq \text{Arg ext}$ [2]-[11].

Наближені АКО діляться на алгоритми з апіорною та апостеріорною оцінками точності [2]-[11].

Евристичні алгоритми будуються на основі правдоподібних міркувань, але не в змозі оцінити точність знайденого розв'язку[2]-[11].

Часто наближеними називають як алгоритми з оцінкою точності, так і евристичні [2]-[11].

Точні алгоритми можна поділити на загальні методи та спеціальні алгоритми. Загальні методи [2]-[11]:

- повний перебір (вичерпний пошук);
- метод гілок і меж (МГіМ);
- метод гілок і відтинань;
- послідовний аналіз варіантів;
- динамічне програмування (метод Беллмана).

Спеціальні алгоритми будуються на основі врахування специфіки конкретної задачі оптимізації, що розв'язується, тому мають вузький спектр застосування [4].

1.3.1 Алгоритм повного перебору

Алгоритм повного перебору (АПП) здійснює пошук у просторі $N!$ рішень шляхом перегляду усіх варіантів [5]. Для найпростішого варіанту симетричної задачі з n містами, навіть у випадку коли одне з міст фіксоване, існує $(N-1)!$ можливих варіантів. Половина з них є зверненням шляху (циклічною перестановкою), тому існує $\frac{(n-1)!}{2}$ можливих маршрутів. Факторіал виразу швидко зростає і вже при $n > 15$ перегляд всіх можливих рішень стає дуже трудомісткою проблемою [2]-[11].

1.3.2 Алгоритм «жадібного» пошуку

Ідея побудови «жадібного» алгоритму полягає в тому, що на кожному кроці робиться локально найкращий вибір, тобто приймаються рішення, що поліпшують цільову функцію, причому таким чином, щоб не порушувати діючих обмежень [2]-[11].

Алгоритм найближчого сусіда відноситься до категорії «жадібних» алгоритмів і є одним з найпростіших евристичних методів рішення ЗК. Згідно з ним, пункти обходу плану послідовно включаються в маршрут, причому кожен черговий включається пункт повинен бути найближчим до останнього вибраного пункту серед всіх інших, ще не включених до складу маршруту [2]-[11].

Алгоритм простий в реалізації, швидко виконується, але, як і інші «жадібні» алгоритми, може видавати неоптимальні рішення, тобто не в змозі оцінити точність знайденого розв'язку [2]-[11].

Варто зазначити, що для будь-якої кількості міст, що більша трьох, в ЗК можна підібрати таке розташування міст (значення відстаней між вершинами графа і вказівка початкової вершини), що алгоритм найближчого сусіда видаватиме найгірше рішення [2]-[11].

1.3.3 Метод гілок і меж

Метод гілок і меж (МГіМ) полягає в тому, що множина всіх рішень задачі послідовно розбивається на все дедалі менші за потужністю підмножини. Для отриманих підмножин перевіряються правила відсіву і відсіяні підмножини виключаються з подальшого розгляду. Раннє отримання значень рекорду, близьких до оптимуму, дозволяє швидше відсіювати підмножини, що заздалегідь не містять оптимального рішення. Для отримання початкового значення рекорду, використовуються один з евристичних методів (у даній роботі, для отримання мінімального початкового рішення

використовується рішення, отримане за допомогою «жадібного» пошуку, що наведено у пункті 1.2.2) [6].

Різні варіанти методу гілок і меж характеризуються в першу чергу застосовуваними процедурами розгалуження. Наприклад, при використанні дихотомічного розгалуження, множина рішень розбивається на дві підмножини, в одній з яких робиться припущення, що оптимальний маршрут містить деякий перехід (k, l) , в іншій підмножині перехід (k, l) забороняється. Послідовно застосовуючи дану процедуру, отримується допустиме рішення, що складається з дозволених переходів. У якості процедури оцінки використовується операція приведення [7].

1.3.4 BV метод

Даний метод заснований на аналізі наявного еталонного маршруту і його оптимізації. Алгоритм умовно складається з трьох етапів [8]:

- отримання початкового еталонного рішення;
- побудова BV матриці;
- оптимізація поточного рішення.

Початкове рішення є кращим рішенням з усіх, що отримані на основі «жадібного» методу, для чого будується N маршрутів, за кількістю міст. У кожному з цих N маршрутів, початкове місто дорівнює номеру маршруту, а всі наступні міста обираються за принципом «жадібного» алгоритму. Далі, серед них обирається найкращий маршрут, тобто з найменшою довжиною шляху і позначається як еталонний [8].

Другий етап – побудова BV матриці. Будується матриця оцінок розміром $N \times N$, що заповнюється нулями. Шляхом розглядання N побудованих «жадібних» маршрутів, для кожної пари сусідніх міст p та q , відповідний елемент матриці оцінок $a[p, q]$ збільшується на одиницю [8].

Третій етап алгоритму полягає в перетворенні еталонного маршруту за допомогою послідовного застосування BV модифікаторів до кожної пари міст,

яким відповідає ненульовий елемент матриці оцінок. Якщо перетворений маршрут має меншу довжину шляху, то він стає еталонним і в матриці оцінок відповідний елемент замінюється на 0. Прохід по BV матриці повторюється до тих пір, поки модифікований маршрут має перевагу над еталоном. Відсутність переваги модифікованого маршруту над еталонним, говорить про те, що отримане рішення є кінцевим [8].

1.3.5 Генетичний алгоритм

Генетичні алгоритми (ГА) виникли на основі спостереження за природними процесами еволюції та селекції популяцій живих істот – особин певного виду – і моделювання їх принципів [13]-[22].

Одне із основних застосувань ГА – розв’язання задач комбінаторної оптимізації (ЗКО), розпізнавання та прогнозування. Оскільки вся інформація про об’єкт міститься у хромосомних наборах, то для розв’язання оптимізаційних задач необхідно подати кожну ознаку об’єкта в закодованому вигляді. Найпоширенішим типом кодування є побітове кодування, при цьому кожній ознаці (алелі) у фенотипі відповідає один ген у генотипі [9].

Загальний підхід, що моделює процес еволюції в живій природі та використовується в еволюційних алгоритмах, подано на рис. 1.2 [13]-[17].

Розробка ГА для розв’язання певної ЗКО вимагає конкретизації таких аспектів [13]-[22]:

- принципів формування початкової популяції;
- механізмів відбору;
- вибору типу кросовера;
- вибору оператора мутації;
- умови завершення.

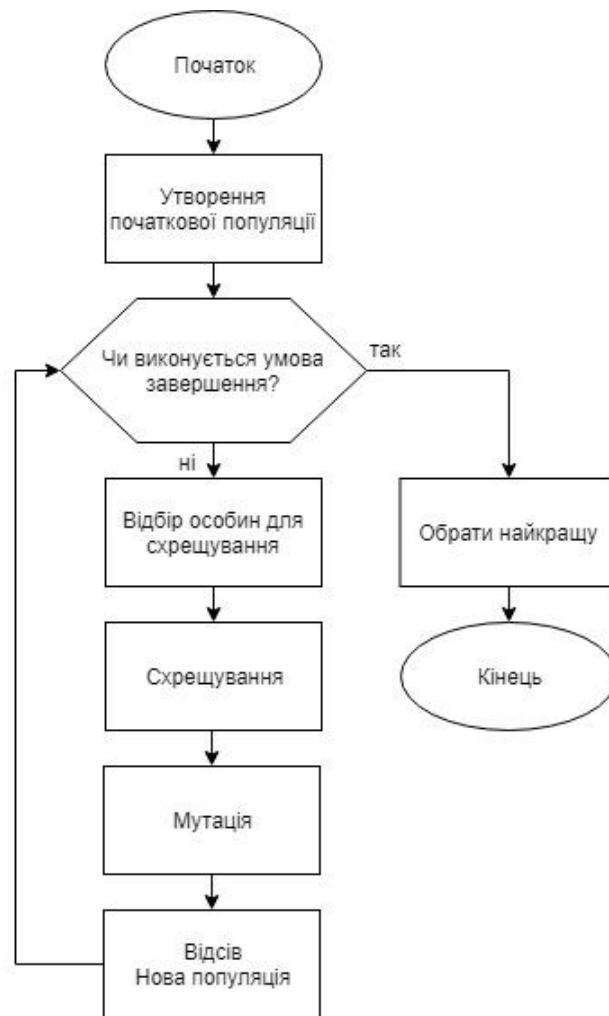


Рисунок 1.2 – Блок-схема еволюційних алгоритмів

При виборі початкової популяції часто застосовують процедури породження випадковим чином t варіантів розв'язку ЗКО, де t – параметр алгоритму, який задається [13]-[22].

У якості критеріїв завершення роботи ГА найчастіше використовуються такі умови [13]-[22]:

- перевищення заданої кількості ітерацій (популяцій);
- відсутність (або невелика зміна) поліпшення наявного рекорду чи середньої пристосованості популяції на кількох останніх ітераціях;
- вичерпання заданого часу на обчислення [10].

У ГА (як і в інших еволюційних алгоритмах) може бути виділено дві форми відбору (рис 1.2). Відбір для варіації, коли особини вибираються для рекомбінації та (або) мутації, та відбір популяції, під час якого особини

відбираються для нового покоління (інколи називається заміною, оскільки деякі або всі батьки замінюються деякими або всіма нащадками) [14].

Функція пристосованості ГА – дійсна або цілочисельна функція однієї або декількох змінних, що підлягає оптимізації в результаті роботи генетичного алгоритму, направляє еволюцію в бік оптимального рішення та виражає міру пристосованості особин у популяції. Ця функція дозволяє оцінити ступінь пристосованості конкретних особин у популяції й вибрати з них найбільш придатні відповідно до еволюційного принципу виживання найсильніших [13]. У задачах оптимізації функція пристосованості зазвичай формується на основі цільової функції й оптимізується [14].

Основну схему еволюційного відбору, покладену в основу розробки ГА, наведено на рис. 1.3 [14].

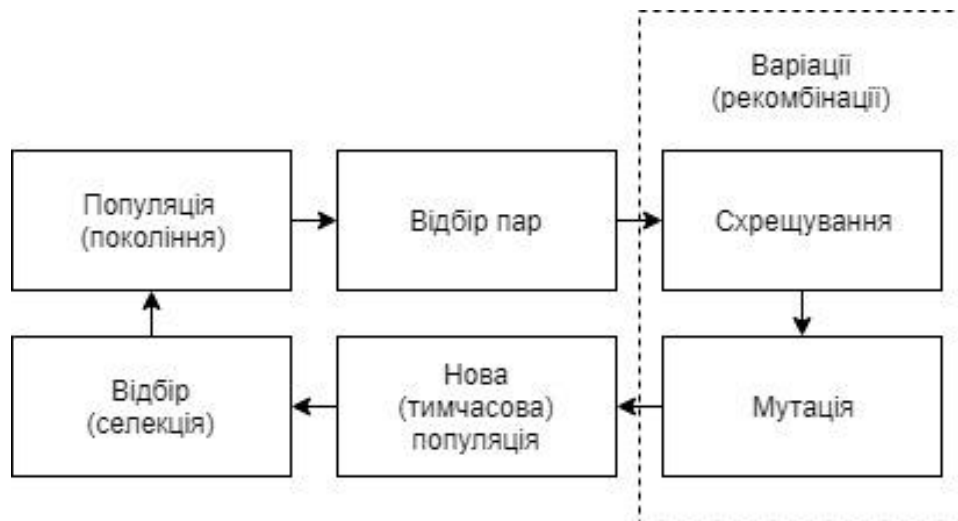


Рисунок 1.3 – Схема еволюційного відбору в ГА

Мета таких алгоритмів – підвищення загального рівня пристосованості популяції. Виділяють три основні класи еволюційних алгоритмів (рис. 1.4) [14].



Рисунок 1.4 – Класи еволюційних алгоритмів

Еволюційні стратегії – це алгоритми розв'язання задач оптимізації з неперервними змінними [13]-[22].

Еволюційне програмування об'єднує алгоритми, які спрямовані на оптимізацію структури програмних комплексів і систем [15]-[16].

Отже, генетичний алгоритм – це еволюційний алгоритм пошуку, що використовується для розв'язання задач оптимізації, моделювання чи підтримки прийняття рішень шляхом послідовного відбору, комбінування й варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію [17].

Одне із основних застосувань ГА – розв'язання задач комбінаторної оптимізації. Загальна ідея ГА – перенесення принципу еволюційного відбору на процес оптимізації [17]-[18].

1.3.6 Порівняння відомих методів розв'язання завдань дискретної оптимізації

Були розглянуті й проаналізовані різні методи вирішення зазначеної задачі, порівняльна таблиця яких наведена нижче (табл. 1.1).

Таблиця 1.1 – Порівняння методів розв’язку задачі дискретної оптимізації

Критерії	Алгоритм повного перебору	Метод гілок і меж	Метод гілок і меж + «жадібний» пошук	BV метод	Генетичний алгоритм
Простота реалізації	+	+	+	-	—
Висока часова залежність від розмірності задачі	+	+	+	—	—
Можливість розпаралелювання	+	+	+	Тільки на етапі оптимізації	+
Можливість досягнення оптимального рішення	+	+	+	—	—

АПП здійснює пошук в просторі $N!$ рішень за допомогою перебору всіх варіантів. Результатом роботи алгоритму є точне рішення. До переваг даного алгоритму можна віднести можливість розпаралелювання і точне рішення задачі. Недоліком АПП є його часова складність, оскільки розмір простору пошуку зростає експоненціально, внаслідок чого метод не може бути використаний для розв’язку практичних задач, тому коли N не є досить малим, використовують евристичні і пошукові алгоритми [11].

МГіМ є розвитком АПП. Його суть полягає в додаванні перевірки критерію обмежуючої функції, виходячи з того, що на певному рівні можна призупинити побудову даної гілки дерева перестановок. Метод зберігає всі позитивні властивості АПП, але все одно є мало придатним для задач, де N не є значно малим. До переваг даного алгоритму можна також віднести можливість розпаралелювання і точне рішення задачі. АПП, МГіМ та його модифікації відносять до точних алгоритмів [2]-[12].

BV метод є евристичним алгоритмом, що має квадратичну складність і не дозволяє отримати точне рішення, а дає лише приблизне, що обумовлено

неповнотою BV матриці, а також наявністю однакових відстаней між містами. Метод може бути розпаралеленим на 3-ому етапі [8].

Генетичний алгоритм відноситься до пошукових алгоритмів і є досить ефективним у порівнянні з іншими методами за відношенням результат-час, чийі недоліки тим не менш пов'язані з передчасною збіжністю (алгоритм не завжди дозволяє знайти вихід з локального мінімуму). Генетичний алгоритм дає наближене рішення і може бути розпаралеленим [13]-[22].

Тому внаслідок проведеного аналізу для розв'язання завдань комбінаторної оптимізації у цій роботі прийнято рішення використовувати генетичні алгоритми.

1.4 Висновки за розділом 1

Проаналізовано завдання комбінаторної оптимізації та задачу комівояжера. Проведено дослідження відомих методів штучного інтелекту, що використовуються для розв'язку задач комбінаторної оптимізації, зокрема розв'язку задачі комівояжера.

Виходячи з проведеного аналізу, зроблено висновок про актуальність дослідження та використання генетичних алгоритмів у якості інструмента для розв'язку задач комбінаторної оптимізації й створення програмного забезпечення для вирішення задачі комівояжера за допомогою генетичного алгоритму.

Визначено основний функціонал, що повинно надавати розроблюване програмне забезпечення:

- зазначення координат міст, обраних для побудови маршруту, за допомогою XML-файлу або вручну;
- введення розміру популяції;
- вибір ймовірності (у процентах) мутації;
- зазначення розміру робочої групи;
- введення максимального числа поколінь;

- введення ймовірності (у процентах) вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції;
- інтерпретація та відображення результатів роботи програми.

2 МАТЕРІАЛИ І МЕТОДИ

У цьому розділі наведено опис прийнятих рішень щодо проєктування та розробки програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації.

2.1 Вибір мови програмування

Для створення програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації доцільно обрати мову програмування C#, яка є універсальною об'єктно-орієнтованою мовою програмування, що забезпечує безпеку перетворення типів, метою створення якої було підвищення продуктивності праці програмістів. В мові збалансовані простота, виразність і ефективність [12].

При створенні C# були враховані досягнення C++, C, Java, SmallTalk, Delphi, Visual Basic і т.д. Були усунені незручні особливості, притаманні іншим мовам і існуючі, як правило, для забезпечення сумісності [12]. Результатом стала поява простої, зручної і сучасної мови, що за потужністю не поступається C++, але істотно підвищує продуктивність розробок [12].

Простежується зв'язок – чим більше мова захищена і стійка до помилок, тим менше продуктивність програм, написаних на ній. У C# існують характерні особливості для обходу можливих помилок [12]. Всі змінні автоматично не започатковані середовищем і мають типовий захист, що дозволяє уникнути невизначених ситуацій у разі, якщо програміст забуде ініціалізувати змінну в об'єкті або спробує провести неприпустиме перетворення типів [12]. Також в C# були вжиті заходи для виключення помилок при оновленні програмного забезпечення. Зміна коду, в такій ситуації, може непередбачувано змінити суть самої програми. Щоб допомогти розробникам боротися з цією проблемою, C# включає в себе підтримку

сумісності версій. Зокрема, на відміну від C++ і Java, якщо метод класу був змінений, це повинно бути спеціальним чином обумовлено [12]. Особливістю є native підтримка інтерфейсів та їх успадкування. Дані можливості дозволяють розробляти складні системи і розвивати їх з часом [12].

У C# була уніфікована система типів, що дозволяє розглядати кожен тип як об'єкт. Використовуючи клас, структуру, масив або вбудований тип, можна звертатися до нього як до об'єкта [12].

У C# немає покажчиків, нетривіальність яких відповідає їх корисності, бо важко уявити програмування без покажчиків на функції. Відповідно до цього в C# присутні Delegates – як прямий аналог покажчика на функцію, що відрізняється захищеністю типів і повною відповідністю концепції об'єктно-орієнтованого програмування [12].

Сучасність C# виявляється і в новому підході до процесу налагодження програми. Традиційним засобом для налагодження програм на стадії розробки в C++ є маркування великих частин коду директивами `#ifdef` і т.д. У C#, використовуючи атрибути, орієнтовані на умовні слова, ви можете куди швидше писати і налагоджувати код [12].

Архітектурою проекту можуть визначатися локальні атрибути, які будуть пов'язані з будь-якими елементами мови – класами, інтерфейсами і т.д. Розробник може програмно перевірити атрибути будь-якого елементу [12].

Наведемо порівняльну характеристику мов програмування (табл. 2.1), що можуть бути використані для розв'язання задачі комівояжера на основі генетичної оптимізації.

Таблиця 2.1 – Порівняння мов програмування

Критерій порівняння мов програмування	C#	Java	Python
Зручність використання для створення програмного забезпечення для розв'язання задачі комівояжера	+	±	–
Методи та засоби розробки інтерфейсу користувача	+	±	±
Підтримка засобів процедурного програмування	+	–	+
Засоби програмування з використанням об'єктно-орієнтованої технології	+	+	±
Можливість роботи з прототипами	+	–	–
Засоби роботи з базами даних	+	+	+

Виходячи з проведеного аналізу, мовою програмування для розробки програми було обрано C#.

2.2 Вибір середовища розробки

Для розробки програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації доцільно обрати середовище розробки Microsoft Visual Studio, яке представляє собою набір інструментів для створення програмного забезпечення: від планування до розробки інтерфейсу, написання коду, тестування, налагодження, аналізу якості коду і продуктивності, розгортання в середовищах клієнтів і збору даних по використанню [12]. Ці інструменти призначені для максимально ефективної

спільної роботи; всі вони доступні в інтегрованому середовищі розробки (IDE) Visual Studio [12].

Visual Studio можна використовувати для створення різних типів додатків, від простих додатків для магазину та ігор для мобільних клієнтів до великих і складних систем, які обслуговують підприємства та центри обробки даних. Visual Studio дозволяє створювати [12]:

- додатки та ігри, які виконуються не тільки на платформі Windows, але і на Android, і iOS [12];
- веб-сайти і веб-служби на основі ASP.NET, JQuery, AngularJS і інших популярних платформ [12];
- додатки для самих різних платформ і пристроїв, включаючи, але не обмежуючись: Office, Sharepoint, Hololens, Kinect [12];
- ігри і графічні додатки для різних пристроїв Windows, включаючи Xbox, з підтримкою DirectX [12].

За замовчуванням, Visual Studio забезпечує підтримку C#, C і C++, JavaScript, F# і Visual Basic. Visual Studio, добре працює і інтерпретується зі сторонніми додатками, наприклад, Unity і Apache Cordova, за допомогою розширень [12]. Користувач також може самостійно розширити Visual Studio, створивши власні інструменти для виконання спеціалізованих завдань [12].

Наведемо порівняльну характеристику середовищ розробки (табл. 2.2), що можуть бути використані для створення програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації.

Таблиця 2.2 – Порівняння середовищ розробки для створення програмного забезпечення для розв’язання задачі комівояжера на основі генетичної оптимізації

Критерій порівняння середовищ розробки	Microsoft Visual Studio	SharpDevelop	MonoDevelop
Зручність використання для створення програмного забезпечення для розв’язання задачі комівояжера	+	+	–
Можливість роботи з репозиторіями	+	+	–
Платформонезалежність	+	–	+
Наявність візуальних засобів розробки	+	+	+
Швидкість модифікації створюваного ПЗ	+	–	+

Виходячи з проведеного аналізу, середовищем розробки для створення програмного забезпечення для розв’язання задачі комівояжера на основі генетичної оптимізації було обрано Microsoft Visual Studio.

2.3 Розроблення модифікованого генетичного методу для розв’язання задачі комбінаторної оптимізації

Під час вирішення задачі комівояжера виникає дві складні проблеми, що пов’язані з використанням генетичного алгоритму для розв’язку задачі. Це кодування туру і алгоритм оператора схрещування, який використовується для створення дочірніх турів.

У стандартному генетичному алгоритмі кодування являє собою просту послідовність чисел, внаслідок чого схрещування виконується шляхом вибору випадкової точки в послідовності вектора батьків і зміні місцями чисел послідовності після неї [13]-[22].

Маршрут туру представлено у вигляді послідовності міст, які, один за іншим, відвідує комівояжер. Виділивши два батьківських тури і встановивши точку схрещування, наприклад, між третім і четвертим пунктами маршруту створюються дочірні маршрути. Для цього в першу частину дочірнього туру помістимо частину маршруту першого батька, яка знаходиться зліва від точки схрещування, а для другої виберемо маршрут туру другого з батьків, що знаходиться справа від точки схрещування. Для другого дочірнього маршруту поміняємо місцями відповідні параметри батьківських маршрутів. Результат представлено табл. 2.3 (міста туру позначені буквами, вертикальна риса – точка схрещування).

Таблиця 2.3 – Стандартний оператор схрещування

Тип туру	Маршрут туру
Батьківський тур 1	F1 A1 B1 E1 C1 G1 D1
Батьківський тур 2	D1 E1 A1 C1 G1 B1 F1
Дочірній тур 1	F1 A1 B1 C1 G1 B1 F1
Дочірній тур 2	D1 E1 A1 E1 C1 G1 D1

Проблема даного методу створення нащадків полягає в тому, що виникає ситуація, при якій перший дочірній маршрут двічі містить в собі міста F1 і B1, проте жодного – D1 чи E1.

Кодування хромосом не може бути подано простим переліком міст в порядку їх слідування. Хоча деякі методи і вирішують дану проблему, вони стикаються з нездатністю враховувати той факт, що послідовність «A1B1C1D1E1F1G1» збігається, наприклад, з «G1F1E1D1C1B1A1».

Для подолання описаних вище обмежень була запропонована модифікація кодування, яка дозволяє зберігати зв'язки для кожного окремого міста туру в обох напрямках. Таким чином перший батьківський тур буде записаний у такий спосіб (табл. 2.4).

Таблиця 2.4 – Структура зберігання списку міст туру та їх взаємозв'язків

Місто	Перший зв'язок	Другий зв'язок
A1	F1	B1
B1	A1	E1
C1	E1	G1
D1	G1	F1
E1	B1	C1
F1	D1	A1
G1	C1	D1

Під час операції схрещування, всі зв'язки, що існують одночасно в обох батьківських маршрутах, будуть взяті і розміщені всередині дочірніх маршрутів. Після чого, для дочірнього маршруту 1, визначення залишившихся зв'язків буде зроблено чергуванням отримання спочатку тих зв'язків, які з'являються в батьківському маршруті 1, а потім – в батьківському маршруті 2. Зв'язки всередині дочірніх маршрутів, поява яких буде приводити до створення неприпустимого маршруту, будуть виключатися з туру. Решта

зв'язків між містами, що не були встановлені в результаті виконання вищеописаних дій, будуть створені випадковим чином.

Оскільки такий оператор схрещування не є повністю випадковим, його можна віднести до «жадібних». В кінці-кінців, у будь-якому випадку, використання генетичного алгоритму призведе до того, що кожне з рішень буде виглядати ідентично, після чого алгоритм не зможе продовжувати пошук найкращого рішення. Для вирішення цієї проблеми використовується велика за кількістю екземплярів початкова популяція, через що ГА потрібно більше часу для приведення всіх рішень до ідентичного вигляду, а також оператор мутації, внаслідок чого деякі дочірні тури змінюються випадковим чином, утворюючи таким чином нові унікальні тури.

Розроблений алгоритм використовує «жадібний» принцип формування початкової популяції, внаслідок чого зв'язки між містами не є повністю випадковими. Генетичний алгоритм, переважно, створює зв'язки між сусідніми, найближче розташованими один до одного, містами. Однак, це не робиться в 100 відсотках випадків для уникнення отримання великої кількості схожих турів в початковій популяції.

Алгоритм вирішує задачу пошуку оптимального маршруту, що проходить через задану множину міст (n) із поверненням в початкове місто.

Алгоритм, що наведено нижче, описує основні принципи функціонування програми і складається з декількох основних етапів.

Етап 1. Завантаження до списку `cityList` координат всіх міст або з XML-файлу (процедура `openCityListButton_Click`), або за допомогою натискання мишею по графічній частині форми (процедура `tourDiagram_MouseDown`), або комбінуючи перший спосіб і другий. Номер міста є індексом списку `cityList`.

Етап 2. Знаходження і подальше запам'ятовування в списку `Distances` для кожного міста відстані до всіх інших міст (метод `CalculateCityDistances` класу `Cities`).

Етап 3. Знаходження і запам'ятовування в списку `CloseCities` для кожного міста номерів його міст сусідів, тобто найближче розташованих до

нього міст. Їх число дорівнює `numberOfCloseCities` (метод `CalculateCityDistances` класу `Cities` і метод `FindClosestCities` класу `City`).

Етап 4. Формування початкової популяції – списку маршрутів, які створюються на основі «жадібного» алгоритму (метод `CreateRandomPopulation` класу `Population`).

Етап 5. Розраховування для кожного маршруту його довжини, тобто значення фітнес-функції (`Fitness`).

Етап 6. Встановлення поточної кількості поколінь (`generation`), кількості схрещувань для генерації дочірніх маршрутів, рівною нулю.

Етап 7. Вибір випадковим чином з популяції (`population`) деякої кількості маршрутів, що дорівнює параметру `groupSize` і перенесення їх до робочої групи - масиву `tourGroup`.

Етап 8. Впорядковування `tourGroup` за зростанням довжини маршруту (застосовується бульбашкове сортування).

Етап 9. Взяття двох перших (найкращих) маршрутів масиву `tourGroup` у якості батьківських і формування, за допомогою використання операторів схрещування і мутації, дочірнього маршруту `child`, (ймовірність мутації дочірнього маршруту задається параметром `mutationChance`).

Етап 10. Заміна в популяції найгіршого маршруту `tourGroup` на маршрут `child`.

Етап 11. Обчислення довжини маршруту `child`, у разі коли вона менше довжини кращого маршруту, визнати `child`-тур кращим маршрутом. Відобразити новий кращий маршрут в графічній частині форми (процедура `displayTour`). Оновити інформацію для відображення відповідних компонентів головної форми.

Етап 12. Для одних і тих же двох батьківських турів, поміняти батьківські параметри місцями (батьківські тури беруться в зворотньому порядку) та повторити виконання дій схрещування і мутації п.9.

Етап 13. Замінення в популяції другого найгіршого маршруту `tourGroup` на отриманий маршрут `child`.

Етап 14. Виконати дії п.11 для отриманого у п. 12 дочірнього туру.

Етап 15. Збільшити значення лічильника поколінь на одиницю ($generation = generation + 1$).

Етап 16. Якщо не перевищено поріг заданої максимальної кількості поколінь ($generation < maxGenerations$), то перейти до п. 7 алгоритму, в протилежному випадку закінчити виконання алгоритму.

Таким чином, у роботі була розроблена модифікація методу розв'язку задачі комівояжера за допомогою генетичного алгоритму, яка полягає у використанні покращеного засобу кодування вхідних даних, що дозволяє окремо зберігати кожне місто туру та зв'язки для нього в обох напрямках, і відповідно до цього, поліпшеного оператора схрещування, що дозволило уникнути проблеми створення помилкових маршрутів і дублювання вже існуючих і призвело до підвищення ефективності роботи алгоритму та його швидкодії в цілому.

Таким чином, розроблено удосконалений генетичний метод розв'язання задачі комбінаторної оптимізації (задачі комівояжера), що, на відміну від існуючих, використовує модифікований спосіб кодування вхідних даних, що дозволяє окремо зберігати кожне місто туру та зв'язки для нього в обох напрямках, і відповідно до цього, розроблено поліпшений оператор схрещування, що дозволяє уникати проблеми створення помилкових маршрутів і дублювання вже існуючих, що, в свою чергу, дозволяє зменшити час розв'язання задачі комівояжера.

2.4 Структурна схема програми

Виходячи із технічного завдання необхідно розробити програму, що надасть можливість програмної реалізації створених алгоритмів, буде мати зручний для користувача інтерфейс та надасть можливості свого подальшого використання для побудови оптимальних схем руху.

Структурна схема програми представлена на рис. 2.1.



Рисунок 2.1 – Структурна схема програми

Програма складається з двох функціональних модулів:

- модуля оптимізації;
- модуля інтерфейсу користувача та візуалізації.

Кожен модуль вирішує окрему задачу.

Модуль оптимізації маршруту дозволяє виконувати пошук найкоротшого шляху між містами, які повинен відвідати комівояжер, список (та координати) яких задаються користувачем, іншими словами, модуль дозволяє знайти маршрут з мінімальною вартістю шляхом застосування генетичного алгоритму з урахуванням зазначених користувачем кількості міст та їх положення відносно один одного, розміру популяції, ймовірності мутації, розміру робочої групи, числа міст-сусідів та проценту ймовірності вибору наступного міста маршруту зі списку його міст-сусідів при формуванні початкової популяції.

Модуль інтерфейсу користувача та візуалізації призначений забезпечення можливостей введення-виведення даних розробленого програмного забезпечення.

2.5 Висновки за розділом 2

Для створення програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації було обрано мову

програмування C# та середовище розробки Microsoft Visual Studio, які є досить зручними для розробки відповідного програмного забезпечення.

Розроблено удосконалений генетичний метод розв'язання задачі комбінаторної оптимізації (задачі комівояжера), що, на відміну від існуючих, використовує модифікований спосіб кодування вхідних даних, що дозволяє окремо зберігати кожне місто туру та зв'язки для нього в обох напрямках, і відповідно до цього, розроблено поліпшений оператор схрещування, що дозволяє уникати проблеми створення помилкових маршрутів і дублювання вже існуючих, що, в свою чергу, дозволяє зменшити час розв'язання задачі комівояжера.

Для програмної реалізації функціональних можливостей розроблюваного програмного забезпечення, проведено вибір і обґрунтовано структуру проектованої системи. Визначено, що програма складається з двох функціональних модулів: модуля оптимізації та модуля інтерфейсу користувача та візуалізації.

3 ОПИС ПРОГРАМИ

У цьому розділі описано розроблене програмне забезпечення для розв’язання задачі комівояжера на основі генетичної оптимізації, зокрема надано схему функціонування програми та описано розроблені програмні модулі.

3.1 Схема функціонування програми

Схему роботи програми зображено на рисунках 3.1–3.2.

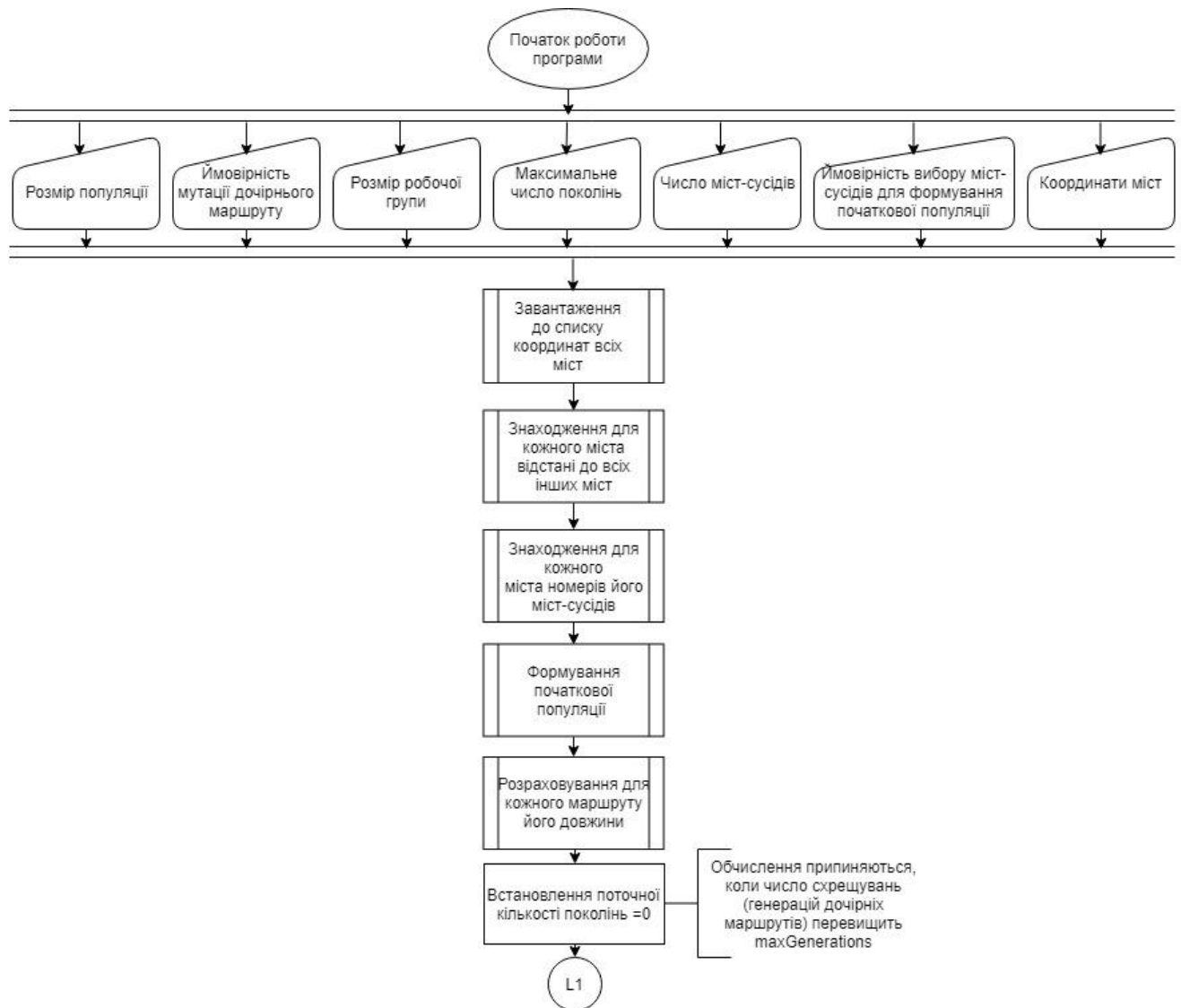


Рисунок 3.1 – Схема роботи програми (перша частина)

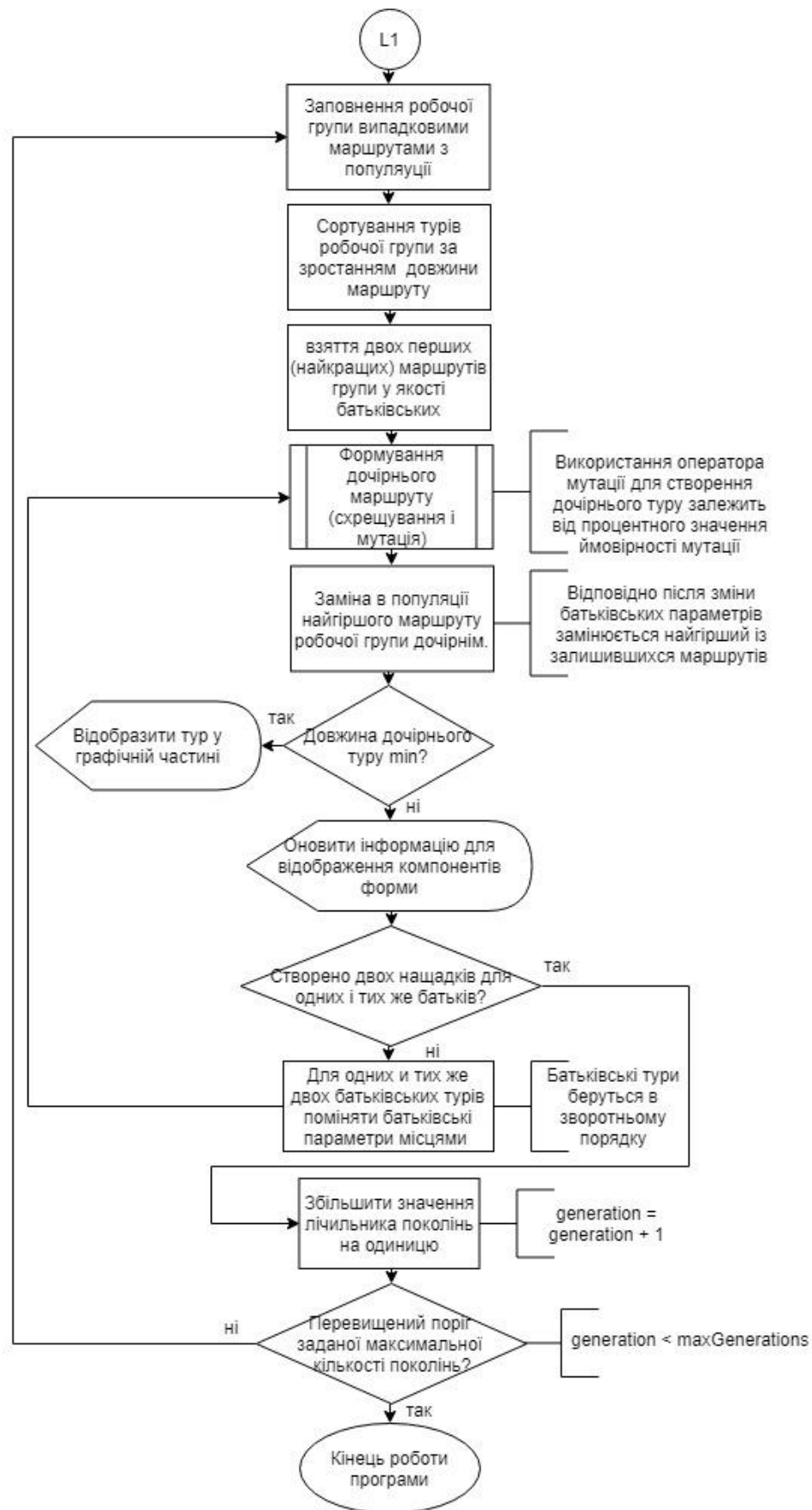


Рисунок 3.2 – Схема роботи програми (друга частина)

Вхідними даними до програми є числа, що описують параметри

алгоритму модуля оптимізації маршруту, значення яких вводяться користувачем з клавіатури у відповідні поля під час роботи з програмою, та при наявності, XML-файл з даними певного формату, що описують позиції міст туру (детальніше про вхідні дані у пункті 3.3).

На рисунку 3.3 показано структуру проекту, та його основні компоненти.



Рисунок 3.3 – Структура проекту та його основні компоненти

Рисунок 3.4 показує основні файлові залежності проєкту.

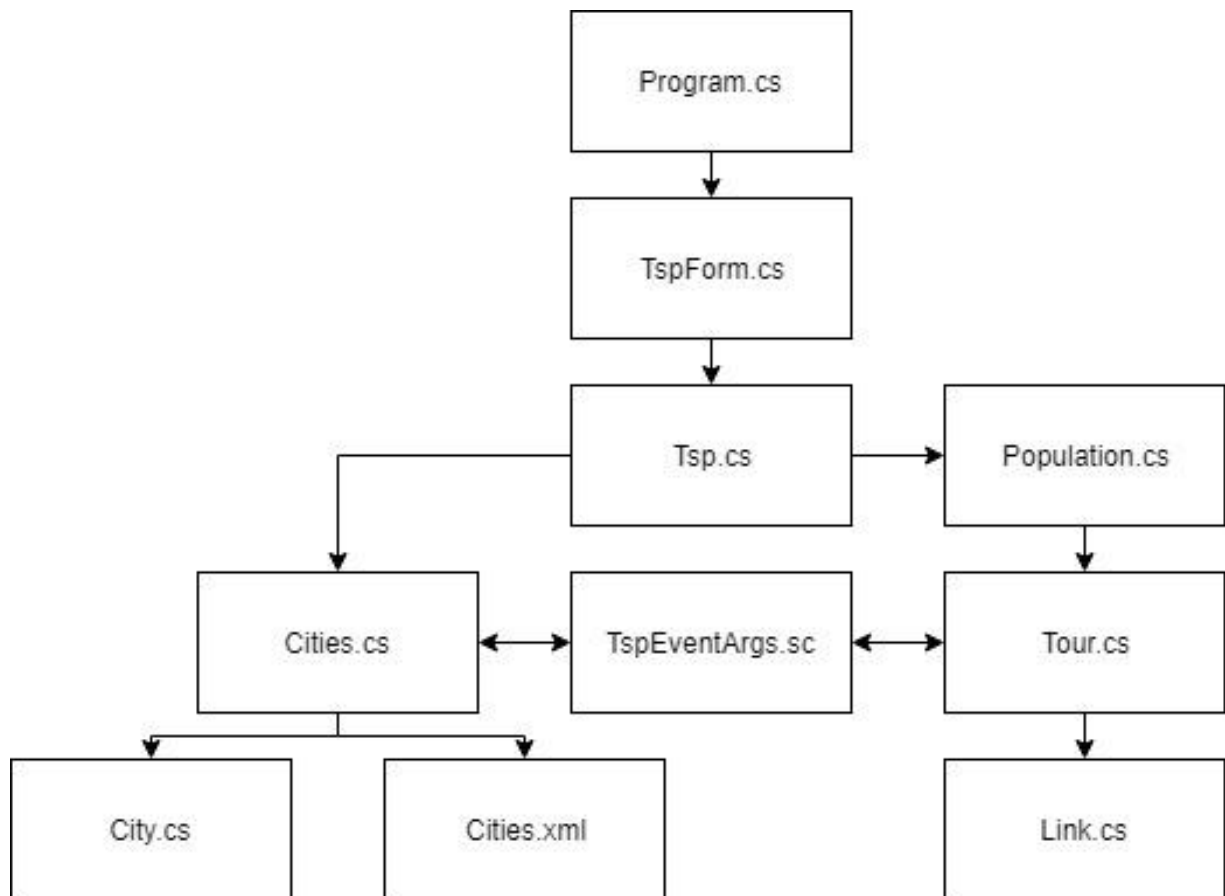


Рисунок 3.4 – Файлові залежності проєкту

3.2 Модуль оптимізації маршруту

Модуль надає програмну реалізацію генетичного алгоритму модифікації методу розв’язку задачі комівояжера.

Вхідними даними модуля є:

- cityList – список, що містить координати обраних для побудови маршруту міст (елемент списку має тип Point). Індекс списку є номером міста;
- populationSize – розмір популяції (тип int) маршрутів, що вдосконалюються генетичним алгоритмом;
- groupSize – розмір групи (тип int), яка зберігається у масиві tourGroup. До масиву випадковим чином обираються маршрути популяції, після чого він сортується за зростанням довжини маршруту. Перші два маршрути відсортованого масиву породжують дочірній маршрут. Для одних і тих же двох батьківських турів схрещування відбувається два рази (під час

другого батьківські параметри міняються місцями – батьківські тури беруться в зворотньому порядку), після чого два дочірніх маршрути замінюють у популяції найгірші тури групи. Розмір робочої групи задається у діапазоні $[2, (populationSize-1)]$;

- `mutation` – ймовірність мутації дочірнього маршруту (тип `int`).
Задається в процентах в діапазоні $[0, 100]$;

- `maxGenerations` – максимальне число поколінь (тип `int`).
Обчислення припиняються, коли число схрещувань (генерацій дочірніх маршрутів) перевищить `maxGenerations`;

- `numberOfCloseCitiies` – число міст-сусідів (тип `int`). Для кожного міста формується список `CloseCitiies` розміру `numberOfCloseCitiies` міст, що є найближчими до поточного міста. Задається в діапазоні від 3 до кількості міст;

- `chanceToUseCloseCity` – ймовірність вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції. Початкові маршрути формуються відповідно до принципу «жадібного» алгоритму. Генерується випадкове число з діапазону $[0, 100]$. Якщо число не перевищує `chanceUseCloseCity`, то наступне місто маршруту випадково вибирається зі списку міст-сусідів поточного міста. В іншому випадку місто випадково вибирається з підмножини міст, які не включені маршрут. Задається в процентах у діапазоні $[0, 100]$;

- `seed` – невід'ємне число «затравки» генератора випадкових чисел, що використовується для формування початкової популяції маршрутів та під час виконання операції схрещування та мутації для дублювання попередніх результатів з метою спрощення процесу пошуку помилок у роботі алгоритму.

Вихідними даними модуля є:

- `Population` – популяція зі зміненими маршрутами (елемент списку має тип `Tour`);

- `BestTour` – найкращий маршрут популяції (властивість класу `Population`, має тип `Tour`);

- lastIterationValue – номер ітерації (покоління) отримання останнього найкращого маршруту (тип int);
- lastFitnessValue – довжина останнього кращого маршруту (тип double);
- відображення останнього найкращого маршруту.

Проміжні дані:

- списки міст-сусідів closeCities (елемент списку має тип int).

Формуються окремо для кожного міста;

- списки відстаней Distances (елемент списку має тип double).

Формуються для кожного міста і містять всередині себе відстані до всіх інших міст;

- fitness – довжина маршруту (властивість маршруту Tour, має тип double). Обчислюється для кожного маршруту окремо;

- generation – номер поточного покоління (тип int);

– робоча група – масив tourGroup (тип int). Формується для кожного покоління зі всієї множини маршрутів. Містить номери маршрутів. Два перших маршрути відсортованого масиву (два його кращих маршруту) використовуються для створення нових дочірніх маршрутів;

- дочірній маршрут child (тип Tour).

Список файлів, які містять у собі основні однойменні класи, що входять до складу модуля наведено у табл. 3.1.

Таблиця 3.1 – Складові модуля оптимізації маршруту

Ім'я файлу	Опис файлу
TspForm.cs	Головна форма модуля
Cities.cs	Клас, що містить список міст, для кожного з яких вказується його координати, відстані до кожного іншого міста і список міст-сусідів.
City.cs	Клас забезпечення формування та зберігання позиції окремого міста
Link.cs	Клас відтворення індивідуального зв'язку між містами

Продовження таблиці 3.1

Ім'я файлу	Опис файлу
Population.cs	Клас, що забезпечує формування початкової популяції.
Program.cs	Містить Main-функцію запуску форми.
Tour.cs	Клас Tour представляє один екземпляр маршруту по всіх містах.
Tsp.cs	Клас, що реалізує алгоритм модифікованого за допомогою ГА метода розв'язку задачі комівояжера
TspEventArgs.cs	Клас установки аргументів події, при настанні якої відображається черговий кращий маршрут.

Основні складові класу tsppForm наведено у табл. 3.2 та табл. 3.3.

Таблиця 3.2 – Поля класу tsppForm

Поле	Тип	Опис
cityList	Cities cityList = new Cities()	Екземпляр класу Cities. Список міст.
tsp	Tsp tspp	Змінна, що відноситься до типу класу tspp та посилається на об'єкт однойменного класу. tspp – клас, який реалізує генетичний алгоритм розв'язання задачі комівояжера (tsp алгоритм)
cityImage	Image cityImage	Образ для відображення міст і маршруту
cityGraphics	Graphics cityGraphics	Графічний об'єкт для образу cityImage

Таблиця 3.3 – Функції-члени класу tsppForm

Метод	Синтаксис та сигнатура	Опис
DrawEventHandler	public delegate void DrawEventHandler(object sender, tsppEventArgs e);	Делегат для потоку, що запускає tspp алгоритм (для оновлення карти міст під час обчислень використовується окремий потік). sender – об'єкт що генерує подію; e – аргументи події
TsppForm	public tsppForm	Конструктор за замовчуванням
tsp_foundNewBestTour	private void tspp_foundNewBestTour(object sender, tsppEventArgs e)	tsp-алгоритм породжує подію foundNewBestTour при знаходженні поточного кращого маршруту. При цьому викликається GUI-потік, що дозволяє оновити карту міст. sender – об'єкт що генерує подію; e – аргументи події

Продовження таблиці 3.3

Метод	Синтаксис та сигнатура	Опис
DrawTour	public void DrawTor(object sendr, tsppEventAgs e)	Алгоритмом tspp отримано новий найкращий тур. Відображає останній краший маршрут. Оновлює відповідні поля (label) sender – об'єкт що генерує подію; e – аргументи події
DrawCityList	private void DrawCityList(Citiies cityList)	Відображає міста на растровій карті. cityList – список міст для відображення (малювання)
BeginTsp	private void BeginTsp(Object stateInfo)	Створює об'єкт класу tspp, виконується в потоці пулу потоків

Основні складові класу Citiies наведено у табл. 3.4.

Таблиця 3.4 – Функції-члени класу Citiies

Метод	Синтаксис та сигнатура	Опис
CalculateCityDistances	public void CalculateCityDistances(int numberOfCloseCitiies)	Знаходить і розміщує в список city.Distances відстані від обраного міста до всіх інших міст. Знаходить для кожного міста його numberOfCloseCitiies міст-сусідів і розміщує їх в список closeCitiies numberOfCloseCitiies – це кількість прилеглих міст, які будуть вважатися сусідами. При створенні початкової популяції турів, більше шансів того, що сусіднє місто буде обраний для зв'язку
OpenCityList	public void OpenCityList(string fileName)	Відкриття XML файлу який містить список міст. fileName – ім'я XML-файлу

Основні складові класу City наведено у табл. 3.5 та табл. 3.6.

Таблиця 3.5 – Поля класу City

Поле	Тип	Опис
Location	Point	Позиція міста
Distances	List<double>	Відстані від міста до кожного з інших міст. Індекс списку – це номер відповідного міста
CloseCitiies	List<int>	Список міст-сусідів

Таблиця 3.6 – Функції-члени класу City

Метод	Синтаксис та сигнатура	Опис
City	public City(int x, int y)	Конструктор, що забезпечує формування позиції міста. x – X координата міста; y – Y координата міста
FindClosestCities	public void FindClosestCities(int numberOfCloseCities)	Знаходження для поточного міста його міст-сусідів (їх число дорівнює numberOfCloseCities). Додавання їх у список closeCities. numberOfCloseCities – це кількість прилеглих міст, які будуть вважатися сусідами. При створенні початкової популяції турів більше шансів того, що сусіднє місто буде обраний для зв'язку.
Location	public Point Location	Методи доступу для визначення значень відповідних їм полів
Distances	public List<double> Distances	
CloseCities	public List<int> CloseCities	

Основні складові класу Link наведено у табл 3.7 та табл.3.8.

Таблиця 3.7 – Поля класу Link

Поле	Тип	Опис
Connection1	int	Зв'язок з першим містом
Connection2	int	Зв'язок з другим містом

Таблиця 3.8 – Функції-члени класу Link

Метод	Синтаксис та сигнатура	Опис
Connection1	public int Connection1	Методи доступу для визначення значень відповідних їм полів
Connection2	public int Connection2	

Клас Population містить поле BestTour (найкращий маршрут, що знайдено за допомогою ГА) типу Tour.

Функції-члени класу Population наведено у табл. 3.9.

Таблиця 3.9 – Функції-члени класу Population

Метод	Синтаксис та сигнатура	Опис
CreateRandomPopulation	public void CreateRandmPopulation(int populatinSize, Citis cityList, Random rnd, int chanceToUseClosCity)	Формує популяцію – початковий набір випадкових маршрутів. populationSize – число створюваних маршрутів (розмір популяції); cityList – список міст в цьому маршруті; rand – генератор випадкових чисел.
BestTour	public Tour BestTour	Метод доступу для визначення значення відповідного йому поля

Клас Tour містить поле Fitness (протяжність маршруту) типу Double.

Функції-члени класу Tour наведено у табл. 3.10.

Таблиця 3.10 – Функції-члени класу Tour

Метод	Синтаксис та сигнатура	Опис
Tour	public Tour(int capacity)	Конструктор класу. Capacity – розмір маршруту. Дорівнює числу міст + 1
resetTour	private void resetTor(int numberOfCitis)	Створює тур, втсанрвлює початкові з'єднання всіх = - 1. numberOfCitiies – кількість міст у турі
DetermineFitne ss	public void DetermineFitnss(Citiies citis)	Визначає загальну довжину маршруту Citiies – список міст маршруту. Використовується, щоб отримати відстань між містами.
joinCitiies	private static void joinCitis(Tour tor, int[] cityUsae, int city1, int cit2)	Створює зв'язок між 2 містами маршруту tour – неповний дочірній тур; cityUsage – скільки разів кожне місто використовувався в цьому турі. Оновлюється при приєднанні міст; city1 – перше місто у з'єднанні; city2 – друге місто в з'єднанні

Продовження таблиці 3.10

Метод	Синтаксис та сигнатура	Опис
findNextCity	private static int findNextCity(Tour parent, Tour child, Cities cityList, int[] cityUsage, int city)	Пошук зв'язку з даного міста в батьківському турі, який може бути розміщений в дочірньому турі. Якщо обидва зв'язки в батьківському турі недопустимі для дочірнього, повертає -1. Parent – батьківський тур для отримання зв'язку Child – дочірній тур в який буде поміщено зв'язок. cityList – список міст цього туру; cityUsage – скільки разів кожне місто використовувалося в дочірньому турі; City – місто від якого буде з'єднання; Returns – місто для зв'язку в дочірньому маршруті, або -1 якщо такого немає
testConnection Valid	private static bool testConnectonVal id(Tour tor, Cities cityList, int[] cityUsage, int city1, int city2)	Визначення, чи можна з'єднати 2 міста з урахуванням існуючих з'єднань в дочірньому турі. Якщо два міста вже можуть бути з'єднані (без повного туру) то зв'язок не є дійсним Tour – неповний дочірній тур. cityList – список міст цього туру; cityUsage – Масив, що містить кількість разів, коли кожне місто було з'єднане у турі; city1 – Перше місто у з'єднанні; city2 – друге місто в з'єднанні
Crossover	public static Tour Crossover(Tour parent1, Tour parent2, Cities cityList, Random rand)	Виконання операції схрещування двох батьківських турів, щоб створити новий дочірній тур. Ця функція повинна викликатися двічі для створення 2 дочірніх турів. При другому виклику батьківські параметри повинні бути поміняні місцями. parent1 – перший батьківський тур; parent2 – другий батьківський тур; cityList – список міст цього туру; rand – генератор випадкових чисел
Mutate	public void Mutate(Random rand)	Випадкова зміна одного з зв'язків у турі. rand – генератор випадкових чисел
Fitness	public double Fitness	Метод доступу для визначення значення відповідного йому поля

Основні складові класу tsprr наведено у табл. 3.11 та табл. 3.12.

Таблиця 3.11 – Поля класу tspp

Поле	Тип	Опис
rand	Random rand;	Змінна, що відноситься до типу класу Random та посилається на об'єкт генератора випадкових чисел
cityList	Citiies cityList	Змінна, що відноситься до типу класу Citiies та посилається на об'єкт визначеного класу. Список міст, що використовується для підрахунку відстані між містами
population	Population population	Змінна, що відноситься до типу класу Population та посилається на об'єкт однойменного класу. Повний список маршрутів (турів)
Halt	public bool Halt	Прапор завершення роботи алгоритму (пошуку нових поколінь) GUI встановлює цей прапор == true для зупинки алгоритму tspp і дозволяє функції Begin () повернути значення

Таблиця 3.12 – Функції-члени класу tspp

Метод	Синтаксис та сигнатура	Опис
NewBestTourEventHandler	public delegate void NewBestTourEventHandler(object sender, tsppEventArgs e);	Обробник події "Знайдений поточний найкращий маршрут" sender – об'єкт що генерує подію; e – аргументи події
NewBestTourEventHandler	public event NewBestTourEventHandler foundNewBstTour;	Подія спрацьовує, коли знайдений новий кращий тур
Tsp	public tspp()	Конструктор за замовчуванням
Begin	public void Begn(int populationize, int maxGeneratins, int groupSize, int mutaton, int seed, int chanceToUseCloseCty, Citiies cityist)	Старт обчислення. populationSize – число випадкових маршрутів, що створюються до початку обчислень; maxGenerations – максимальне число схрещувань (поколінь); groupSize – число маршрутів, що перевіряються в кожному поколінні. Mutation – ймовірність мутації дочірнього маршруту; Seed – Seed генератора випадкових чисел; chanceToUseCloseCity – можливість вибору міста-сусіда; cityList – список міст.

Продовження таблиці 3.12

Метод	Синтаксис та сигнатура	Опис
makeChildren	bool makeChilden(int groupSize, int mutaton)	Випадковим чином вибирається з популяції група маршрутів. Після сортування групи за зростанням довжини маршруту два перших розглядаються як батьківські. Вони беруть участь в схрещуванні. Дочірні маршрути замінюють у популяції 2 гірших маршрути tourGroup. groupSize – розмір робочої групи; Mutation – ймовірність мутації (в%) дочірнього маршруту.
displayTour	void displayTor(Tour bestTor, int generationNumbr, bool complte)	Породжує подію, що сприймається GUI, який відображає маршрут
Halt	public bool Halt	Метод доступу для визначення значення відповідного йому поля

Основні складові класу tsppEventArgs наведено у табл. 3.13 та табл. 3.14.

Таблиця 3.13 – Поля класу tsppEventArgs

Поле	Тип	Опис
CityList	public Citiies CityList	Список міст
BestTour	public Tour BestTour	Найкращий маршрут
Generation	public int Generation	Номер покоління, в якому з'явився поточний найкращий маршрут
Complete	public bool Complete	Прапор завершення алгоритму.

Таблиця 3.14 – Функції-члени класу tsppEventArgs

Метод	Синтаксис та сигнатура	Опис
TspEventArgs	public tsppEventArgs()	Конструктор за замовчуванням
CityList	public Citiies CityList	Методи доступу для визначення значень відповідних їм полів
BestTour	public Tour BestTour	
Generation	public int Generation	
Complete	public bool Complete	
TspEventArgs	public tsppEventrgs(Citiies cityLst, Tour bestTur, int generaton, bool complte)	Конструктор, який встановлює значення властивостей події. cityList – список міст для відображення bestTour – кращий маршрут, що з'єднає всі міста. Generation – номер покоління Complete – Прапор завершення алгоритму

Клас Program містить всередині себе Main-функцію запуску форми, що є точкою входу додатка.

3.3 Проектування інтерфейсу

Елементи інтерфейсу користувача основної форми Main_Form наведено у табл. 3.15.

Таблиця 3.15 – Елементи інтерфейсу користувача форми Main_Form

Назва елемента	Тип	Опис
tourDiagram	PictureBox	Компонент відображення карти міст маршруту
populationSizeTextBox	TextBox	Текстове поле для вводу початкового розміру популяції
lastIterationValue	Label	Текстове поле відображення номеру ітерації отримання останнього найкращого маршруту
lastFitnessValue	Label	Компонент відображення відстані останнього найкращого маршруту (найліпшого значення фітнес-функції)
StartButton	Button	Кнопка запуску основного алгоритму модуля
fileNameTextBox	TextBox	Текстове поле для відображення й зазначення шляху до файлу списку міст
maxGenerationTextBox	TextBox	Текстове поле для вводу максимальної кількості поколінь
groupSizeTextBox	TextBox	Текстове поле для вводу кількості міст, що увійдуть до складу «робочої групи»
randomSeedTextBox	TextBox	Текстове поле вводу числа для «затравки» генератора випадкових чисел
openCityListButton	Button	Кнопка формування списку міст за XML файлом та відображення їх на карті
clearCityListButton	Button	Кнопка очистки списку міст та, відповідно до цього, карти туру
StatusLabel	Label	Компонент відображення поточного статусу виконання програми та створюваних виключень
selectFileButton	Button	Кнопка вибору XML файлу з списком міст
mutationTextBox	TextBox	Текстове поле для вводу ймовірності мутації
NumberCitiiesValue	Label	Компонент відображення кількості міст у турі
NumberCloseCitiiesTextBox	TextBox	Число міст сусідів кожного міста
CloseCityOddsTextBox	TextBox	Текстове поле для вводу вірогідності вибору міста-сусіда поточного міста під час формування поточної популяції

Основним елементом головної форми відповідають компоненти Label, що надають кожному елементу текст опису.

Функції Main_Form, що забезпечують роботу програмної складової користувальницького інтерфейсу, наведено у табл. 3.16.

Таблиця 3.16 – Функції-оброблювачі подій форми Main_Form

Назва функції	Синтаксис	Опис
StartButton_Click	private void StartButton_Click(object sender, EventArgs e)	Запускає або зупиняє рішення задачі комівояжера sender – об'єкт що генерує подію; e – аргументи події
updateCityCount	private void updateCityCount()	Відображення кількості міст у відповідному компоненті форми
selectFileButton_Click	private void selectFileButon_Click(object sender, EventArgs e)	Користувач вибирає новий XML-файл списку міст. Не може бути використано, якщо запущений tspp-алгоритм. sender – об'єкт що генерує подію; e – аргументи події
openCityListButton_Click	private void openCityListButon_Click(object sender, EventArgs e)	Користувач вибрав "відкрити список міст". Формування списку міст по XML-файлу та їх відображення на карті. Не може бути використано якщо запущений tspp-алгоритм
clearCityListButton_Click	private void clearCityListButton_Click(object sender, EventArgs e)	Користувач вибрав "Очистити список міст". Видаляє всі міста зі списку міст та, відповідно до цього, їх графічне представлення з карти. Не може бути використано якщо запущений tspp-алгоритм
tourDiagram_MouseDown	private void tourDiagram_MouseDown(object sender, MouseEventArgs e)	Користувач клікнув на карті міст Не може бути використано якщо запущений tspp-алгоритм. Розміщення нового міста на карті і додавання його в список міст

На рисунку 3.5 представлений загальний вигляд форми tsppForm.

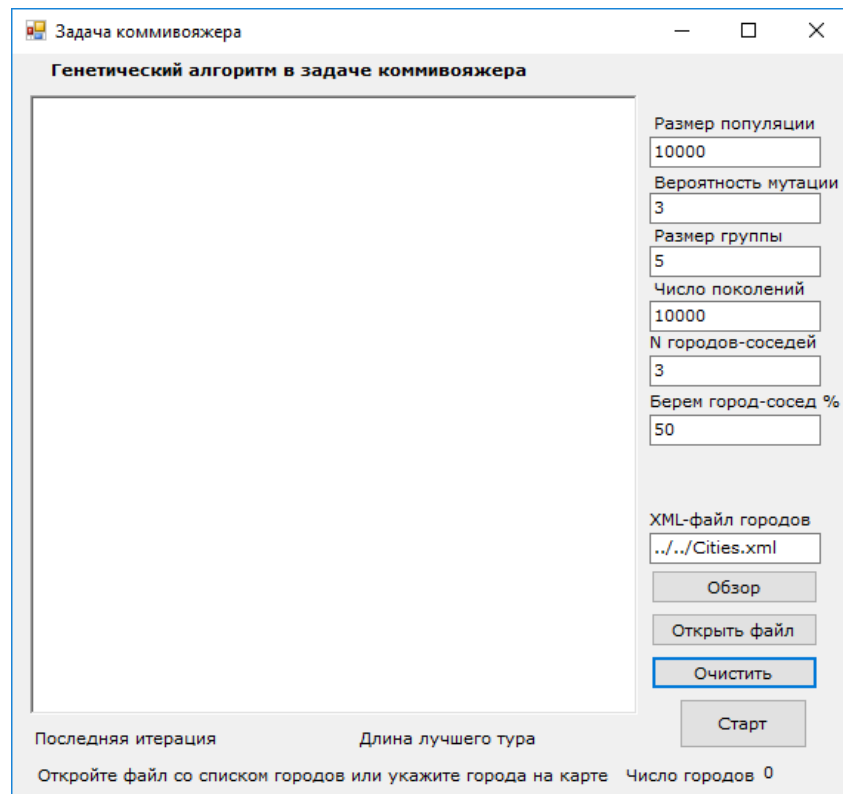


Рисунок 3.5 – Вигляд форми tsppForm

3.4 Висновки за розділом 3

У даному розділі було розглянуто основні рішення, пов'язані з реалізацією компонентів програми. Розроблено схему функціонування програми.

Був приведений опис організації початкової та вихідної інформації, що використовується, наведена інформація стосовно основних складових програми.

Окремо були розглянуті основні рішення щодо проектування і реалізації інтерфейсу програми.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програмний продукт вирішує задачу комбінаторної оптимізації на прикладі задачі комівояжера і призначений для використання будь-якими суб'єктами (фізичними особами, підприємствами або фірмами), що мають справу з транспортною логістикою для вирішення задач зниження логістичних витрат.

Програма забезпечує виконання наступних функцій:

- зазначення координат міст, обраних для побудови маршруту, за допомогою XML-файлу або вручну;
- введення розміру популяції;
- вибір ймовірності (у процентах) мутації;
- зазначення розміру робочої групи;
- введення максимального числа поколінь;
- введення ймовірності (у процентах) вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції;
- інтерпретація та відображення результатів роботи програми.

Для експлуатації ПП необхідні такі програмно-технічні засоби:

- персональний компютер під керуванням операційною системою сімейства Windows;
- жорсткий диск з вільним простором не менше 100 Мб;
- обсяг ОЗП – не менш 600 Мб;
- монітор з роздільною здатністю екрану 1280·1024 або вище;
- маніпулятор миша;
- клавіатура.

Для роботи програми повинна бути встановлена програмна платформа .NET Framework версії 4.5.1 або вище.

Функціональні характеристики розробленої програмної системи – згідно з технічним завданням, наведеним у додатку А. Опис програми та текст програми наведені в додатках Б і В відповідно.

4.2 Характеристики програми

Час роботи програми вирішення задачі комбінаторної оптимізації залежить від конфігурації комп'ютера, на якому вона виконується.

Програма складається з таких функціональних модулів:

- модуля оптимізації;
- модуля інтерфейсу користувача та візуалізації.

Модуль оптимізації маршруту дозволяє виконувати пошук найкоротшого шляху між містами, які повинен відвідати комівояжер, список (та координати) яких задаються користувачем, іншими словами модуль дозволяє знайти маршрут з мінімальною вартістю, шляхом застосування генетичного алгоритму з урахуванням зазначених користувачем кількості міст та їх положення відносно один одного, розміру популяції, ймовірності мутації, розміру робочої групи, числа міст-сусідів та проценту ймовірності вибору наступного міста маршруту зі списку його міст-сусідів при формуванні початкової популяції.

Модуль інтерфейсу користувача та візуалізації містить засоби для забезпечення введення-виведення даних – результатів роботи розробленої програми.

Програма відстежує коректність даних та у випадку введення некоректних даних, або даних неправильного формату на екран видається повідомлення про виникнення виняткової ситуації та її характер.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для завантаження програми потрібно скопіювати файли програми з флеш-носія на ПК у відповідну директорію.

Для початку роботи з програмою необхідно запустити файл `Salesman.exe` на виконання.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програми є числа, що описують параметри алгоритму модуля оптимізації маршруту, значення яких вводяться користувачем з клавіатури у відповідні поля під час роботи з програмою, та, при наявності, XML-файл з даними певного формату, що описують позиції міст туру.

Вхідними даними модуля оптимізації маршруту є:

- `cityList` – список, що містить координати обраних для побудови маршруту міст (елемент списку має тип `Point`). Індекс списку є номером міста;
- `populationSize` – розмір популяції (тип `int`) маршрутів, що вдосконалюється генетичним алгоритмом;
- `groupSize` – розмір групи (тип `int`), яка зберігається у масиві `tourGroup`. До масиву випадковим чином обираються маршрути популяції, після чого він сортується за зростанням довжини маршруту. Перші два маршрути відсортованого масиву породжують дочірній маршрут. Для одних і тих же двох батьківських турів схрещування відбувається два рази (під час другого батьківські параметри міняються місцями), після чого два дочірніх маршрути замінюють найгірші тури у групі. Розмір робочої групи задається у діапазоні $[2, (populationSize-1)]$;
- `mutation` – ймовірність мутації дочірнього маршруту (тип `int`).
Задається в процентах у діапазоні $[0, 100]$;

– `maxGenerations` – максимальне число поколінь (тип `int`). Обчислення припиняються, коли число схрещувань (генерацій дочірніх маршрутів) перевищить `maxGenerations`;

– `numberOfCloseCities` – число міст-сусідів (тип `int`). Для кожного міста формується список `CloseCities` розміру `numberOfCloseCities` міст, що є найближчими до поточного міста. Задається в діапазоні від 3 до кількості міст;

– `chanceToUseCloseCity` – ймовірність вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції. Початкові маршрути формуються відповідно до принципу «жадібного» алгоритму. Генерується випадкове число з діапазону $[0, 100]$. Якщо число не перевищує `chanceUseCloseCity`, то наступне місто маршруту випадково вибирається зі списку міст-сусідів поточного міста. В іншому випадку місто випадково вибирається з підмножини міст, які не включені маршрут. Задається в процентах в діапазоні $[0, 100]$;

– `Seed` – невід'ємне число «затравки» генератора випадкових чисел, що використовується для формування початкової популяції маршрутів та під час виконання операції схрещування та мутації для дублювання попередніх результатів з метою спрощення процесу пошуку помилок у роботі алгоритму.

Вихідними даними модуля є:

– `Population` – популяція зі зміненими маршрутами (елемент списку має тип `Tour`);

– `BestTour` – кращий маршрут популяції (властивість класу `Population`, має тип `Tour`);

– `lastIterationValue` – номер ітерації (покоління) отримання останнього найкращого маршруту (тип `int`);

– `lastFitnessValue` – довжина останнього кращого маршруту (тип `double`);

– відображення останнього найкращого маршруту.

Проміжні дані:

- списки міст-сусідів `closeCities` (елемент списку має тип `int`).
Формуються окремо для кожного міста;
- списки відстаней `Distances` (елемент списку має тип `double`).
Формуються для кожного міста і містять всередині себе відстані до всіх інших міст;
- `fitness` – довжина маршруту (властивість маршруту `Tour`, має тип `double`). Обчислюється для кожного маршруту окремо;
- `generation` – номер поточного покоління (тип `int`);
- робоча група – масив `tourGroup` (тип `int`). Формується для кожного покоління зі всієї множини маршрутів. Містить номери маршрутів. Два перших маршрути відсортованого масиву (два його кращих маршруту) використовуються для створення нових дочірніх маршрутів;
- дочірній маршрут `child` (тип `Tour`).

4.3.3 Повідомлення

Розроблений програмний продукт гарантує стійке функціонування. У разі виникнення виняткових ситуацій у програмі передбачено відображення відповідних повідомлень, текст і опис яких наведено в табл. 4.1.

Таблиця 4.1 – Повідомлення програмісту

Текст повідомлення	Опис виняткової ситуації
Задайте размер популяции	Кількість особин у популяції повинна бути більша за нуль
Задайте максимальное число поколений	Максимальна кількість поколінь (генерацій дочірніх маршрутів) повинна бути більша за нуль
Встановіть відсоток ймовірності мутації	Ймовірність мутації дочірнього маршруту повинна приймати значення від 0 до 100 процентів
Встановіть розмір робочої групи	Розмір робочої групи повинен приймати значення від 2 до розміру популяції-1
Вкажіть кількість міст-сусідів для створення початкової популяції	Число міст сусідів повинно належати діапазону від 3 до кількості міст туру

Продовження таблиці 4.1

Текст повідомлення	Опис виняткової ситуації
Задайте можливість використання міста-сусіда при створенні початкової популяції	Визначте ймовірність використання міста-сусіда при створенні початкової популяції в діапазоні від 0 до 95 процентів
Вкажіть seed RNG	Вкажіть параметр «затравки» генератора випадкових чисел відмінний від 0
Завантажте файл списку міст, або клацніть по карті, щоб розмістити щонайменше 5 міст	Маршрут туру повинен містити щонайменше 5 міст
Неможливо змінити список міст під час роботи алгоритму	Спроба створення нового списку міст або змінення вже існуючого під час роботи алгоритму
Файл не знайдено: " + fileName	Зазначений XML-файлу не знайдено у вказаній директорії
Невірний формат файлу XML	Помилка формату даних XML-файлу

При введенні даних неправильного формату на екран видається повідомлення про виникнення виняткової ситуації та її характер.

4.4 Приклад роботи з програмою

Для початку роботи з програмою необхідно запустити файл Salesman.exe на виконання.

Після запуску головного вікна програми заповнюємо вхідними даними алгоритму основні поля форми (рисунок 4.1).

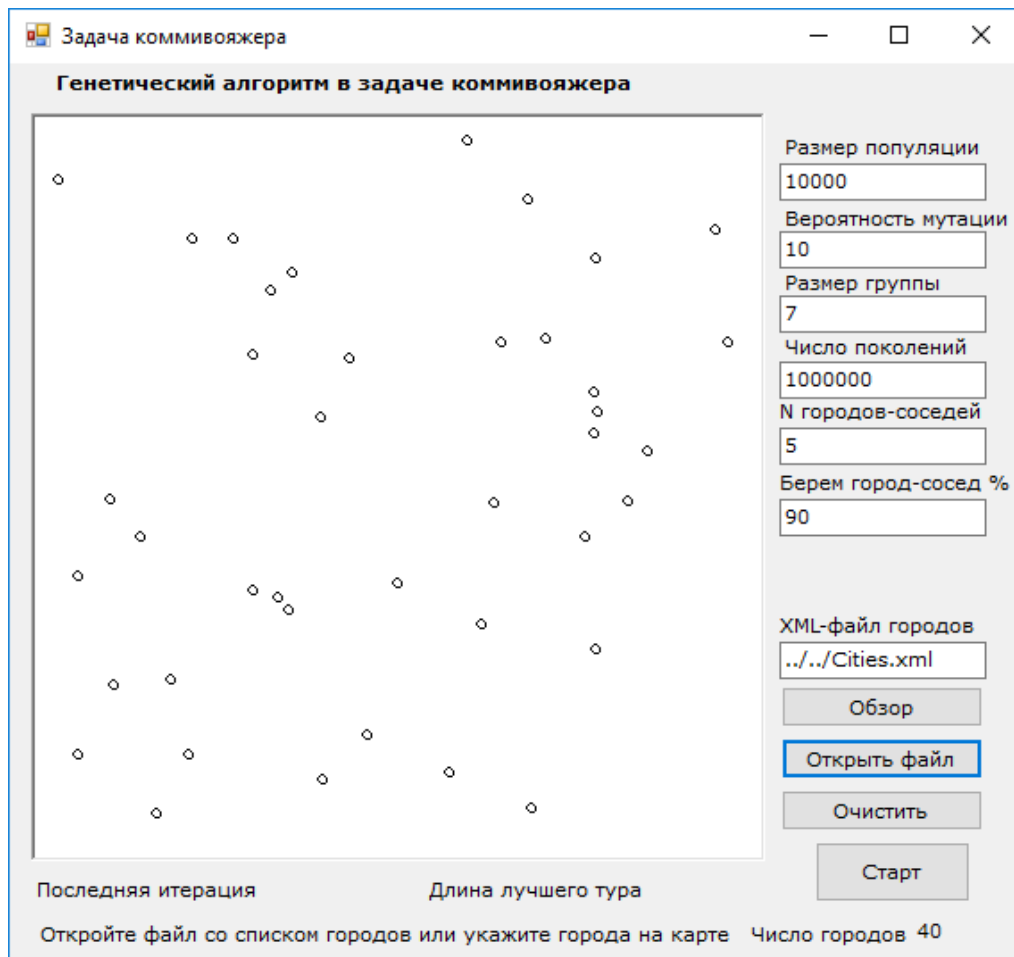


Рисунок 4.1 – Початкові дані до програми

Вказується кількість особин (турів) початкової популяції, ймовірність мутації дочірнього маршруту в процентах, розмір робочої групи маршрутів, число поколінь (генерацій дочірніх маршрутів), число міст-сусідів кожного міста туру, та процент взяття міст-сусідів під час формування початкової популяції.

Натискання кнопки «Обзор» відкриє діалогове вікно вибору XML-файлу міст, проте відповідне поле дозволяє зазначити шлях до нього вручну. Загальний вигляд файлу міст та формату даних всередині нього наведено на рисунку 4.2. Міста туру також можна додавати напряму, натискаючи мишею на карті.

```

1 <?xml version="1.0" encoding="utf-8" ?>
2 <CityList>
3   <City X="160" Y="370"/>
4   <City X="121" Y="132"/>
5   <City X="58" Y="234"/>
6   <City X="41" Y="213"/>
7   <City X="143" Y="86"/>
8   <City X="312" Y="176"/>
9   <City X="121" Y="264"/>
10  <City X="231" Y="366"/>
11  <City X="380" Y="62"/>
12  <City X="202" Y="260"/>
13  <City X="241" Y="12"/>
14  <City X="23" Y="356"/>
15  <City X="43" Y="317"/>
16  <City X="175" Y="134"/>
17  <City X="260" Y="125"/>
18  <City X="277" Y="386"/>
19  <City X="312" Y="153"/>
20  <City X="23" Y="256"/>
21  <City X="313" Y="297"/>
22  <City X="131" Y="96"/>
23  <City X="87" Y="67"/>
24  <City X="249" Y="283"/>
25  <City X="307" Y="234"/>
26  <City X="159" Y="167"/>
27  <City X="285" Y="123"/>
28  <City X="12" Y="34"/>
29  <City X="85" Y="356"/>
30  <City X="387" Y="125"/>
31  <City X="135" Y="268"/>
32  <City X="185" Y="345"/>
33  <City X="75" Y="314"/>
34  <City X="275" Y="45"/>
35  <City X="313" Y="78"/>
36  <City X="110" Y="67"/>
37  <City X="256" Y="215"/>
38  <City X="314" Y="164"/>
39  <City X="342" Y="186"/>
40  <City X="331" Y="214"/>
41  <City X="141" Y="275"/>

```

Рисунок 4.2 – XML-файл міст

Використання кнопок «Открыть файл» і «Очистить» надає користувачеві засоби управління поданням даних.

Кнопка «старт» розпочне дії основного алгоритму програми, повторне натискання зазначеної кнопки під час роботи програми зупинить поточні дії алгоритму і разом з цим – процес розв’язку задачі.

Результати представлено графічним поданням найкращого маршруту туру на карті, що супроводжується відображенням номеру ітерації (числа виконаних операцій генерації дочірніх маршрутів до моменту в якому було отримано поточний найкращий тур) та довжини останнього (на момент завершення роботи програми) знайденого найкращого маршруту (рис. 4.3).

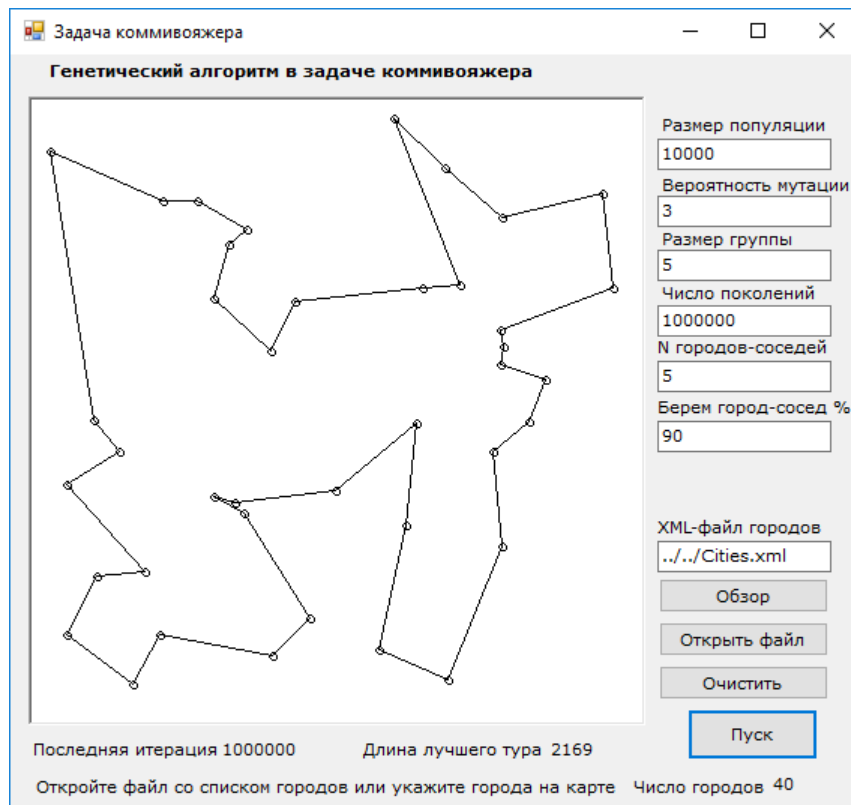


Рисунок 4.3 – Результаты работы программы

4.5 Эксперименты та результати

Внаслідок того, що генетичний алгоритм не може самостійно оцінити точність знайденого рішення, мета його роботи полягає в знаходженні кращого результату в порівнянні з наявним, а не оптимального, проблема досягнення оптимуму при цьому є вторинною.

Як зазначалося раніше, використання генетичного алгоритму призведе в якийсь момент часу до того, що кожне з наявних рішень буде виглядати ідентично один одному, після чого алгоритм не зможе продовжувати пошук найкращого рішення. Відповідно до цього, можна зробити висновки про те, яким чином, кожний з параметрів, що описує генетичний алгоритм, впливає на його ефективність.

Достатньо велика кількість екземплярів початкової популяції (10000 і більше) збільшить ефективність розв'язку вирішуваної задачі у відповідності з тим, що ГА знадобиться більший проміжок часу для приведення всіх рішень

до ідентичного вигляду. Такий підхід збільшить ймовірність того, що буде знайдено найкраще (оптимальне) рішення. Вирішити зазначену проблему дозволяє також використання оператора мутації з метою видозмінення дочірніх маршрутів й наступного утворення нових виняткових турів.

Великий розмір робочої групи збільшить ймовірність того, що у якості батьківських турів будуть вибрані маршрути з заздалегідь кращим значенням функції пристосованості ніж у інших, у той же час, це означає те, що більшість турів популяції ніколи не буде використана у якості батьків. Великий розмір групи призведе до збільшення швидкодії алгоритму, проте зменшить ймовірність знаходження оптимального рішення

Розроблений алгоритм використовує «жадібний» принцип формування початкової популяції, внаслідок чого створення зв'язків на зазначеному етапі відбувається переважно між сусідніми, найближче розташованими один до одного, містами. Однак, це не робиться в 100 відсотках випадків для уникнення отримання великої кількості схожих турів в початковій популяції.

Тим не менш, проаналізувавши усе вищезазначене, можна сказати, що основними початковими параметрами досліджуваного алгоритму за величиною впливу на кінцевий результат оптимізації є розмір початкової популяції та ймовірність мутації утворюваних маршрутів (експериментальна оцінка чутливості роботи алгоритму до параметрів пошуку наведена нижче).

Для оцінки ефективності розроблюваного алгоритму було проведено його порівняльне дослідження з алгоритмами-представниками різних класів. Алгоритм було порівняно з точним алгоритмом повного перебору та евристичним BV-методом.

Для кожної задачі розмірності N експерименти проводяться за заздалегідь сформованим туром, що є однакоим для усіх описаних методів. Таким чином забезпечується використання для всіх досліджуваних методів однакових вхідних даних для забезпечення математичної коректності порівняння результатів експериментів.

Загальноприйнятим способом оцінювання якості та точності алгоритму є залежність часу та відносної похибки від розмірності задачі.

Експериментальним шляхом визначено вплив вхідних даних (параметрів) генетичного алгоритму на результат роботи програми (точність розробленого алгоритму). Розглядається залежність величини похибки у знаходженні результату від розміру початкової популяції та ймовірності мутації дочірніх турів. Відповідно до цього робиться вибір початкових параметрів розробленого алгоритму для подальшого його порівняння з іншими методами розв'язку досліджуваної задачі.

На основі отриманих результатів визначається середнє відхилення алгоритму повного перебору та BV методу від розробленого генетичного алгоритму.

При застосуванні ГА до задач оптимізації, початкові значення параметрів алгоритму для отримання точних результатів є заздалегідь невідомими, відповідно до цього дослідимо чутливість алгоритму та його результатів до зміни вхідних параметрів. Результати наведено у табл. 4.2–4.3.

Буде розглядатися залежність величини похибки у знаходженні результату від розміру початкової популяції та ймовірності мутації дочірніх турів. Відповідно до цього буде зроблений вибір початкових параметрів розробленого алгоритму для подальших порівнянь.

Таблиця 4.2 – Чутливість ГА та результатів до зміни розміру початкової популяції

Розмірність задачі (число міст)	Розмір популяції	Ймовірність мутації (%)	Номер покоління отримання популяції ідентичних маршрутів	Результат	Похибка (%)
40	1000	0	5435	2913.8	39.25
40	3000	0	22094	2623.2	25.36
40	5000	0	34610	2377.1	13.6
40	10000	0	102890	2246.48	7.35
40	20000	0	161153	2353.73	12.48

Таблиця 4.3 – Чутливість ГА та результатів до зміни ймовірності мутації

Розмір ність задачі (число міст)	Розмір популя ції	Ймовірність мутації (%)	Результат	Похибка (%)
40	1000	1	2236.42	6.87
40	1000	3	2385.65	14
40	1000	5	2160.94	3.27
40	1000	10	2153.45	2.91
40	1000	20	2176.01	3.99

Представимо графічно абсолютну похибку результатів ГА в залежності від зміни параметру ймовірності мутації та розміру початкової популяції (рис. 4.4, рис. 4.5).

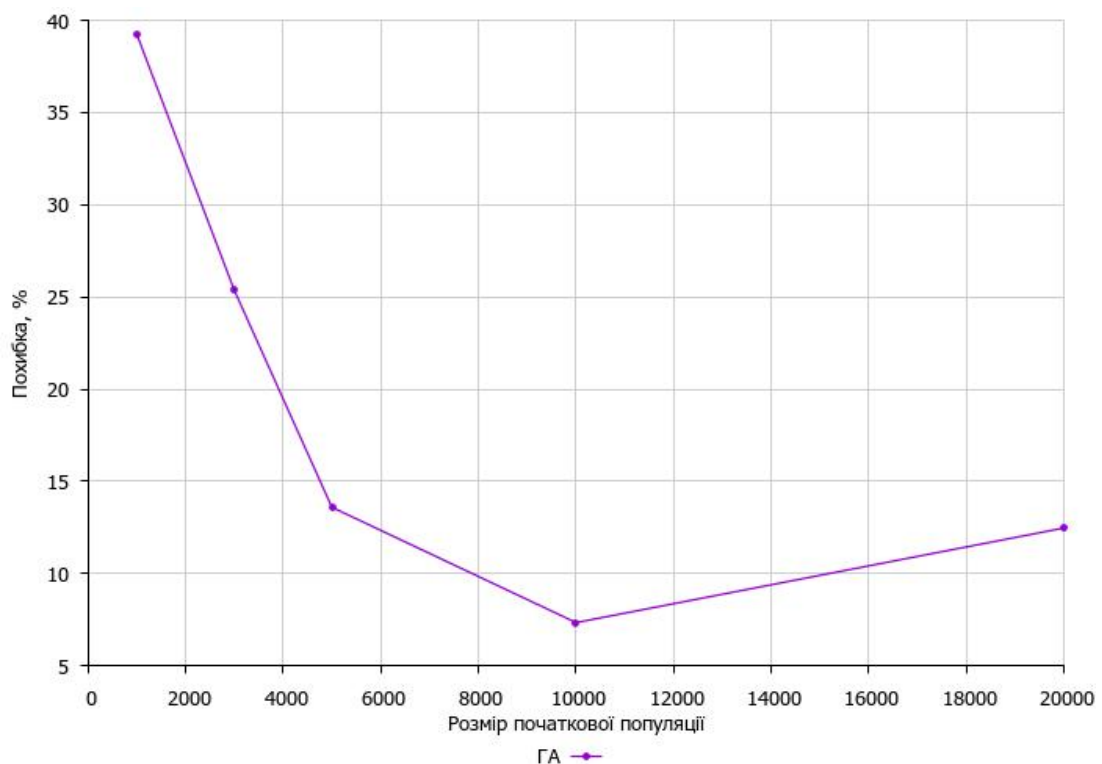


Рисунок 4.4 – Похибка ГА в залежності від зміни розміру початкової популяції

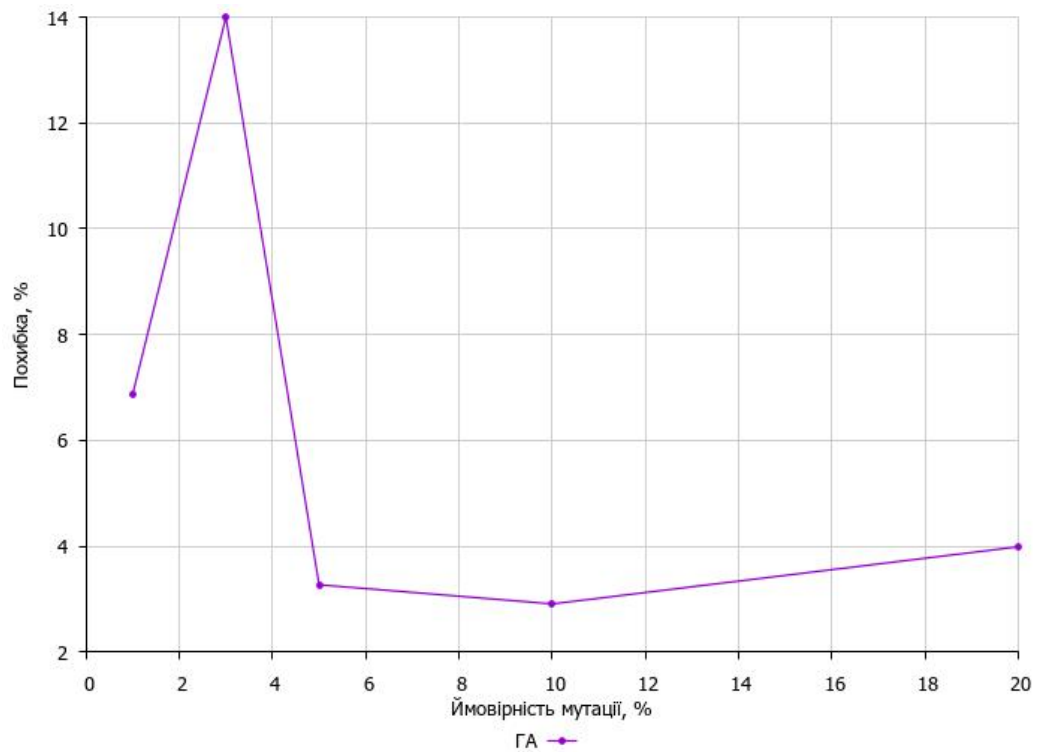


Рисунок 4.5 – Похибка ГА в залежності від зміни ймовірності мутації

Таким чином, маючи на увазі значення початкових параметрів, за яких ГА демонструє найбільшу точність, і як наслідок, найменше значення похибки, дослідимо залежність точності ГА від комплексної зміни обох досліджуваних параметрів (табл. 4.4).

Таблиця 4.4 – Чутливість ГА та результатів до комплексної зміни початкових параметрів

Розмірність задачі (число міст)	Розмір популяції	Ймовірність мутації	Результат	Похибка (%)
40	1000	1	2236.42	6.877
40	3000	3	2097.35	0.231
40	5000	5	2181.45	4.25
40	10000	10	2092.51	0
40	20000	20	2097.35	0.231

Виходячи з результатів дослідження, можна зробити висновок, що тестування ГА при комбінації рівнів початкових параметрів, за окремими

значеннями кожного з яких ГА демонстрував найбільш високу експериментальну точність, так само дає найкращий (найточніший) результат.

Внаслідок цього, для подальшого порівняння ГА з іншими методами штучного інтелекту, обрано найоптимальніші значення початкових параметрів пошуку ГА. Розмір початкової популяції – 10000, ймовірність мутації дочірнього туру – 10 процентів.

Табл. 4.5 та рисунок 4.6 демонструють результати дослідження часової залежності досліджуваних алгоритмів від розмірності вирішуваної задачі.

Таблиця 4.5 – Часова залежність від розмірності

Розмірність задачі (число міст)	Точний алгоритм	Евристичний алгоритм	Пошуковий алгоритм
	АПП	BV	ГА
5	5	25	9298
10	41	84	15578
15	10800000	401	28370
20	-	918	35942
40	-	2856	101862
60	-	5504	113969
80	-	9337	134209
100	-	16187	176899

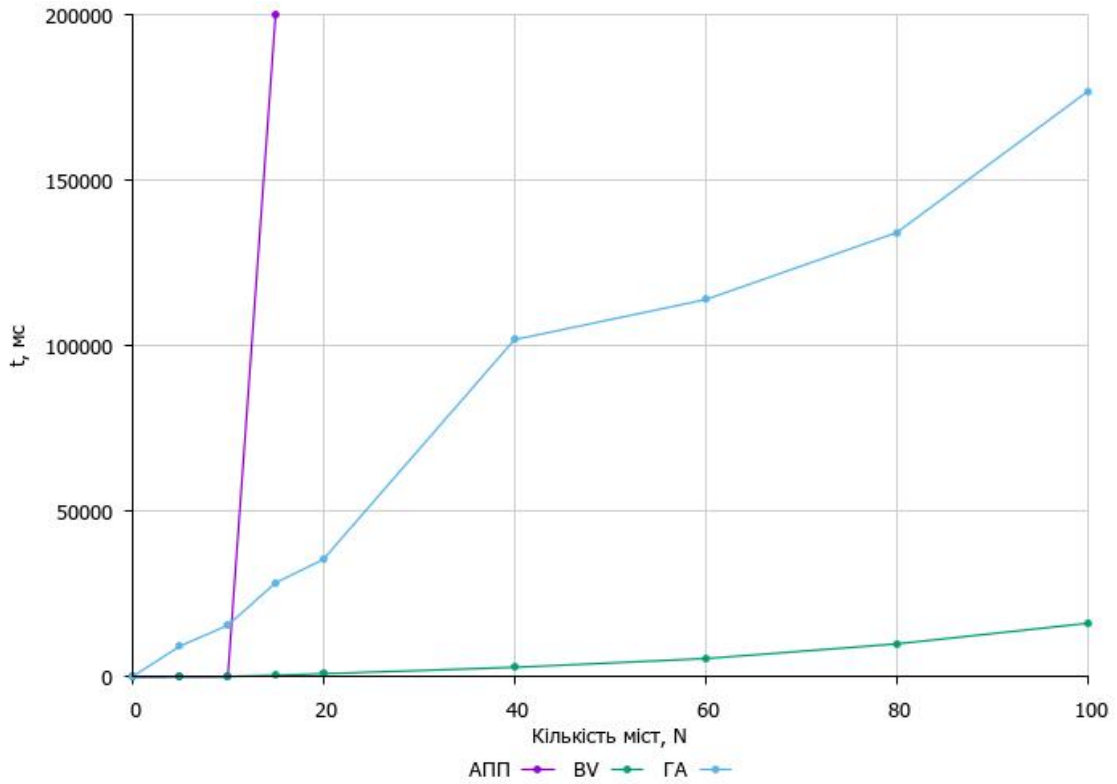


Рисунок 4.6 – Часова залежність від розмірності задачі

На основі отриманих вище даних зазначимо, що надзвичайно велика залежність часу розрахунків точних методів від розмірності задачі робить АПП не оптимальним алгоритмом за параметрами «результат-час». Оцінимо величину відносної похибки алгоритму для BV методу відносно розробленого ГА алгоритму. Результати наведено у табл. 4.6.

Таблиця 4.6 – Результати порівняння ГА з BV

Розмірність задачі (число міст)	Відхилення BV від ГА, %
5	7.76
10	12.13
15	13.18
20	13.89
40	14.07
60	15.18
80	16.33
100	16.55

Виходячи з цього побудуємо графік залежності відносної похибки від розмірності задачі для визначення переваг досліджуваних методів у точності та ефективності відносно один одного. Результати наведено на рисунку 4.7.

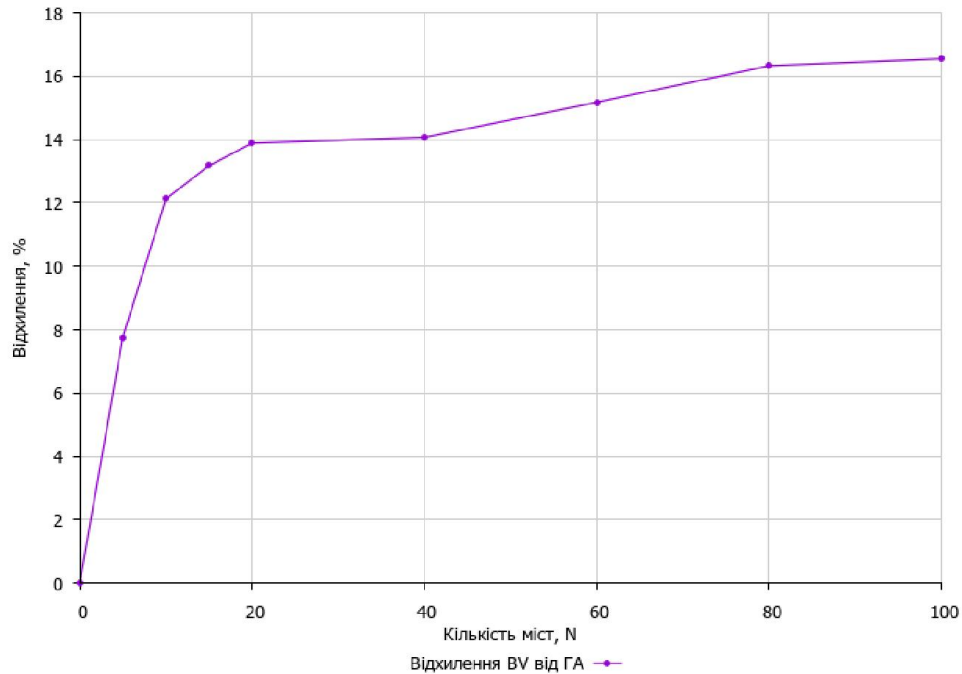


Рисунок 4.7 – Відносна похибка від розмірності задачі

За отриманими результатами, можна сказати, що розроблюваний метод має переваги у точності (більш ніж 15) в порівнянні з евристичним BV методом, та у часі (швидкодія ГА більше у 360 разів) в порівнянні з досліджуваним точним алгоритмом повного перебору. При чому зазначені переваги стають більш вагомими зі збільшенням розмірності задачі, що свідчить про доцільність використання даного методу та його подальшого розвитку.

4.6 Висновки за розділом 4

Наведено результати експлуатації, тестування та експериментального дослідження розробленої програми для розв'язання задачі комівояжера на основі генетичної оптимізації.

Результати роботи модифікованого за допомогою ГА методу розв'язку задачі комбінаторної оптимізації порівнювалися з результатами різноманітних підходів і показали, що розроблений алгоритм забезпечує ефективний і точний метод розв'язку задачі комівояжера і є найоптимальнішим серед представлених за показниками точності та витраченого часу.

Проведені експерименти показали, що розроблений метод перевершує евристичний BV метод за точністю більш ніж на 15% і цей показник підвищується при збільшенні розмірності розв'язуваної задачі. У той же час алгоритм у 360 разів перевершує точний алгоритм повного перебору за швидкістю, проте поступається за цим показником BV методу.

Можна зробити припущення, що ключем до покращення показників роботи алгоритму є саме те, що в процесі роботи ГА для породження нащадків ГА використовує як імовірнісні, так і детерміновані правила, що дає значно більший ефект, ніж кожен з цих методів окремо. Проте необхідність проведення додаткових експериментів для визначення початкових параметрів пошуку алгоритму і формулювання критеріїв відбору для забезпечення ефективної роботи операторів ГА робить пошуковий алгоритм ресурсномістким засобом розв'язку задачі.

ВИСНОВКИ

В ході виконання даної кваліфікаційної роботи магістра було проведено огляд та аналіз генетичних методів розв'язання завдань комбінаторної оптимізації.

Для досягнення поставленої мети роботи було виконано наступні завдання:

- досліджено проблемну область з комбінаторної оптимізації;
- проаналізовано сучасні методи розв'язання завдань комбінаторної оптимізації;
- розроблено удосконалений генетичний метод розв'язання задачі комівояжера;
- виконано проєктування програмного забезпечення для розв'язання задачі комівояжера;
- розроблено програмне забезпечення для розв'язання задачі комівояжера;
- виконано тестування та експериментальне дослідження розробленої програми для розв'язання задачі комівояжера.

Відповідно до поставленої мети зроблений висновок, щодо актуальності питання створення універсального інструменту для вирішення широкого кола задач комбінаторної оптимізації, впровадження якого дозволить користувачам отримати під час рішення задачі комівояжера наступні переваги:

- зазначити будь-яку кількість міст туру, що не вплине на час розв'язку задачі;
- побудувати найкоротший маршрут серед міст туру (отримати найкраще значення цільової функції пристосованості) за допомогою використання модифікованого алгоритму розв'язку, що засновується на еволюційному алгоритмі.

Розроблено удосконалений генетичний метод розв'язання задачі комбінаторної оптимізації (задачі комівояжера), що, на відміну від існуючих, використовує модифікований спосіб кодування вхідних даних, що дозволяє окремо зберігати кожне місто туру та зв'язки для нього в обох напрямках, і відповідно до цього, розроблено поліпшений оператор схрещування, що дозволяє уникати проблеми створення помилкових маршрутів і дублювання вже існуючих, що, в свою чергу, дозволяє зменшити час розв'язання задачі комівояжера.

У середовищі розробки Microsoft Visual Studio із використанням мови програмування C# створено програмний продукт, що забезпечує для користувача виконання наступних функцій:

- зазначення координат міст, обраних для побудови маршруту, за допомогою XML-файлу або вручну;
- введення розміру популяції;
- вибір ймовірності (у процентах) мутації;
- зазначення розміру робочої групи;
- введення максимального числа поколінь;
- введення ймовірності (у процентах) вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції;
- інтерпретація та відображення результатів роботи програми.

Розроблене програмне забезпечення має зручний інтерфейс, що робить процес взаємодії користувача з програмою легким і зрозумілим.

Проведені експерименти показали, що розроблений метод перевершує евристичний BV метод за точністю більш ніж на 15% і цей показник підвищується при збільшенні розмірності розв'язуваної задачі. У той же час алгоритм суттєво (в рази) перевершує точний алгоритм повного перебору за швидкістю, проте поступається за цим показником BV методу.

Програмний продукт може бути використаний будь-якими суб'єктами (фізичними особами, підприємствами або фірмами), що мають справу з транспортною логістикою для вирішення задач зниження логістичних витрат.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дыбская, В.В. Логистика [Текст] / В.В. Дыбская, Е.И. Зайцев, В.И. Сергеев. – М. : Эксмо, 2013. – 944 с.
2. Гуляницький, Л.Ф. Прикладні методи комбінаторної оптимізації : навч. посібник [Текст] / Л.Ф. Гуляницький, О.Ю. Мулеса. – К. : Видавничо-поліграфічний центр «Київський університет», 2016. – 142 с.
3. Мудров, В.И. Задача о коммивояжере [Текст] / В.И. Мудров. – М. : Знание, 1969. – 62 с.
4. Сергиенко, И.В. Классификация прикладных методов комбинаторной оптимизации [Текст] / И.В. Сергиенко, Л.Ф. Гуляницький, С.И. Сиренко // Кибернетика и системный анализ. – 2009. – № 5. – С. 71–83.
5. Борознов, В.О. Исследование решения задачи коммивояжера [Текст] / В.О. Борознов // Управление, вычислительная техника и информатика. – 2009. – № 2. – С. 147–151.
6. Игнатьев, А.Л. Сравнение различных методов решения задачи коммивояжера на многопроцессорных системах [Электрон. ресурс] / А.Л. Игнатьев. – Режим доступа: <https://pandia.ru/text/78/339/1401.php>
7. Костюк, Ю.Л. Эффективная реализация алгоритма решения задачи коммивояжера методом ветвей и границ [Текст] / Ю.Л. Костюк // Прикладная дискретная математика. Вычислительные методы в дискретной математике. – 2010. – №2. –С. 78–90.
8. Борознов, В.О. Исследование эвристического метода решения задачи коммивояжера [Текст] / В.О. Борознов // Компьютерное обеспечение и вычислительная техника. – 2008. – № 28. – С. 322–328.
9. Кормен, Т.Х. Алгоритмы. Построение и анализ [Текст] / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Р. Ривест. – М. : Вильямс, 2012. – 1296 с.
10. Левитин, А.В. Алгоритмы: введение в разработку и анализ [Текст] / А.В. Левитин. – М. : Вильямс, 2015. – 576 с.

11. Базилевич, Р.П. Дослідження ефективності існуючих алгоритмів для розв'язання задачі комівояжера [Текст] / Р.П. Базилевич, Р.К. Кутельмах // Комп'ютерні науки та інформаційні технології. – 2009. – № 650. – С. 235–245.
12. Троелсен, Э. С# и платформа .NET. Библиотека программиста [Текст] / Э. Троелсен. – Санкт-Петербург : Питер, 2007. – 800 с.
13. Стецюра, Г.Г. Эволюционные методы в задачах управления, выбора, оптимизации [Текст] / Г.Г. Стецюра // Приборы и системы управления. – 1998. – № 3. – С. 54–62.
14. Гудман, Э.Д. Эволюционные вычисления и генетические алгоритмы [Текст] / Э.Д. Гудман // Обзорение прикладной и промышленной математики. – 1996. – № 5. – С. 14–17.
15. Курейчик, В.М. Эволюционные алгоритмы: генетическое программирование [Текст] / В.М. Курейчик, С.И. Родзин // Теория системы управления. – 2002. – № 1. – С 127–137.
16. Рутковская, Д. Нейронные сети, генетические алгоритмы и нечеткие системы [Текст] / Д. Рутковская, М. Пилиньский, Л. Рутковский. – М. : Телеком, 2004. – 452 с.
17. Куприянов, М.С. Генетические алгоритмы и их реализация в системах реального времени [Текст] / М.С. Куприянов, Н.И. Матвиенко // Информационные технологии. – 2001. – № 1. – С. 17–21.
18. Васильев, В.И. Интеллектуальные системы управления с использованием генетических алгоритмов [Текст] / В.И. Васильев, Б.Г. Ильясов // Информационные технологии. – 2000. – № 12. – С. 25–26.
19. Олійник, А.О. Еволюційні обчислення та програмування : навчальний посібник [Текст] / А.О. Олійник, С.О. Субботін, О.О. Олійник. – Запоріжжя : ЗНТУ, 2010. – 324 с.
20. Субботін, С.О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень [Текст] / С.О. Субботін. – Запоріжжя: ЗНТУ, 2008. – 341 с.

21. Gen, M. Genetic algorithms and engineering design [Text] / M. Gen, R. Cheng. – New Jersey : John Wiley & Sons, 1997. – 352 p.
22. Haupt, R. Practical genetic algorithms [Text] / Randy L. Haupt, Sue Ellen Haupt. – 2nd ed., 2004. – 261 p.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може застосовуватися до широкого кола задач комбінаторної оптимізації.

A.1 Підстава для розробки

Підставою для розробки слугує завдання на кваліфікаційну роботу на тему «Дослідження та програмна реалізація генетичних методів розв’язання завдань комбінаторної оптимізації», затверджене наказом Національного університету «Запорізька політехніка» № 435 від 05 листопада 2021 р.

A.2 Призначення розробки

Програмний продукт призначений для використання будь-якими суб'єктами (фізичними особами, підприємствами або фірмами), що мають справу з транспортною логістикою для вирішення задач зниження логістичних витрат.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати виконання наступних функцій:

- зазначення координат міст, обраних для побудови маршруту, за допомогою XML-файлу або вручну;
- введення розміру популяції;
- вибір ймовірності (у процентах) мутації;
- зазначення розміру робочої групи;
- введення максимального числа поколінь;

- введення ймовірності (у процентах) вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції;
- інтерпретація та відображення результатів роботи програми.

A.3.2 Вимоги до інтерфейсу програми

Комп'ютерна програма повинна мати графічний інтерфейс для зручного використання оператором. Інтерфейс повинен забезпечувати зручну навігацію всередині програми. Взаємодія програми з оператором повинна забезпечуватися за допомогою форм введення даних, їх редагування, за необхідністю, засобів перегляду результатів.

A.3.3 Вимоги до надійності

Програма повинна відстежувати коректність і формат вхідних даних, правильно обробляти виняткові ситуації, що виникають під час її роботи.

A.3.4 Умови експлуатації

Експлуатація програми повинна здійснюватися відповідно до експлуатаційної документації на розроблену програму, яка відповідає стандартам і містить інформацію, необхідну для освоєння та експлуатації.

A.3.5 Вимоги до складу та параметрів технічний засобів

Для експлуатації програмного продукту необхідні такі програмно-технічні засоби:

- персональний компютер під керуванням операційною системою сімейства Windows;

- жорсткий диск з вільним простором не менше 100 Мб;
- обсяг ОЗП – не менш 600 Мб;
- монітор з роздільною здатністю екрану 1280·1024 або вище;
- маніпулятор миша;
- клавіатура.

A.3.6 Вимоги до маркування і пакування

Програма може бути записана на флеш-носії.

A.3.7 Вимоги до транспортування й зберігання

Вимоги до транспортування й зберігання аналогічні вимогам, пропонованим до носія, на якому записана програма.

A.4 Вхідні дані до роботи

Вхідними даними, що використовуються для виконання роботи, є:

- список, що містить координати обраних для побудови маршруту міст;
- розмір популяції маршрутів, що вдосконалюються генетичним алгоритмом;
- розмір робочої групи. Кількість випадковим чином обраних маршрутів популяцій для створення дочірніх елементів;
- ймовірність мутації дочірнього маршруту;
- максимальне число поколінь;
- число міст-сусідів. Для кожного міста формується список міст, що є найближчими до поточного міста;
- вірогідність вибору наступного маршруту міста з списку його міст-сусідів при формуванні маршрутів початкової популяції;

– невід'ємне число «затравки» генератора випадкових чисел для дублювання попередніх результатів з метою спрощення процесу пошуку помилок у роботі алгоритму.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Програма розроблена за допомогою використання середовища розробки Microsoft Visual Studio та мови програмування C#.

Б.2 Функціональне призначення

Програмний продукт призначений для використання будь-якими суб'єктами (фізичними особами, підприємствами або фірмами), що мають справу з транспортною логістикою для вирішення задач зниження логістичних витрат.

Програма забезпечує виконання наступних функцій:

- зазначення координат міст, обраних для побудови маршруту, за допомогою XML-файлу або вручну;
- введення розміру популяції;
- вибір ймовірності (у процентах) мутації;
- зазначення розміру робочої групи;
- введення максимального числа поколінь;
- введення ймовірності (у процентах) вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції;
- інтерпретація та відображення результатів роботи програми.

Б.3 Опис логічної структури

Програма представляє собою Windows-застосування.

Структурна схема програми наведена на рис. Б.1.



Рисунок Б.1 – Структурна схема програми

Програма має ієрархічну структуру і складається з двох функціональних модулів:

- модуля оптимізації;
- модуля інтерфейсу користувача та візуалізації.

Кожен модуль вирішує окрему задачу.

Модуль оптимізації маршруту дозволяє виконувати пошук найкоротшого шляху між містами, які повинен відвідати комівояжер, список (та координати) яких задаються користувачем, іншими словами модуль дозволяє знайти маршрут з мінімальною вартістю, шляхом застосування генетичного алгоритму з урахуванням зазначених користувачем кількості міст та їх положення відносно один одного, розміру популяції, ймовірності мутації, розміру робочої групи, числа міст-сусідів та проценту ймовірності вибору наступного міста маршруту зі списку його міст-сусідів при формуванні початкової популяції.

Модуль інтерфейсу користувача та візуалізації містить засоби для забезпечення можливостей введення-виведення даних.

Б.4 Використані технічні засоби

Для експлуатації програмного продукту необхідні такі програмно-технічні засоби:

- ПК на апаратній платформі x64 під керуванням ОС сімейства WINDOWS (Windows XP/7/8/10);
- жорсткий диск з вільним простором не менше 100 Мб;
- обсяг ОЗП – не менш 512 Мб;
- монітор з роздільною здатністю екрану 1280·1024 або вище;
- маніпулятор миша;
- клавіатура.

Б.5 Виклик і завантаження

Для завантаження програми користувачеві потрібно скопіювати файли програми з флеш-носія на ПК у відповідну директорію.

Для початку роботи з програмою необхідно запустити файл Portfolio Salesman.exe на виконання.

Б.6 Вхідні та вихідні дані

Вхідними даними, що використовуються для виконання роботи, є:

- список, що містить координати обраних для побудови маршруту міст. Індекс списку є номером міста;
- розмір популяції маршрутів, що вдосконалюються генетичним алгоритмом;
- розмір групи, яка зберігається у відповідному масиві. До масиву випадковим чином обираються маршрути популяції, після чого він сортується за зростанням довжини маршруту. Перші два маршрути відсортованого масиву породжують дочірній маршрут. Для одних і тих же двох батьківських

турів схрещування відбувається два рази (під час другого батьківські параметри міняються місцями – батьківські тури беруться в зворотньому порядку), після чого два дочірніх маршрути замінюють у популяції найгірші тури групи;

- ймовірність мутації дочірнього маршруту. Задається в процентах в діапазоні;
- максимальне число поколінь. Обчислення припиняються, коли число схрещувань (генерацій дочірніх маршрутів) перевищить його;
- число міст-сусідів. Для кожного міста формується список міст, що є найближчими до поточного міста;
- ймовірність вибору наступного міста маршруту зі списку його міст-сусідів при формуванні маршрутів початкової популяції. Початкові маршрути формуються відповідно до принципу «жадібного» алгоритму;
- невід'ємне число «затравки» генератора випадкових чисел, що використовується для формування початкової популяції маршрутів та під час виконання операції схрещування та мутації для дублювання попередніх результатів з метою спрощення процесу пошуку помилок у роботі алгоритму.

Вихідними даними модуля є:

- популяція зі зміненими маршрутами;
- найкращий маршрут популяції;
- номер ітерації (покоління) отримання останнього найкращого маршруту;
- довжина останнього найкращого маршруту;
- відображення останнього найкращого туру.

ДОДАТОК В
Текст програми

```

/// Citiies.cs

using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.IO;
using System.Globalization;

namespace tspp
{
    /// <summary>
    /// Класс Citiies
    /// </summary>
    public class Citiies : List<City>
    {
        /// <summary>
        /// </summary>
        /// <param name="numberOfCloseCitiies">При создании
начальной популяции туров, больше шансов того,
        /// что соседний город будет выбран для связи. Это
количество близлежащих городов, которые будут считаться
соседями.</param>

        /// <summary>
        /// Открытие XML файла который содержит список городов.
        /// </summary>
        /// <param name="fileName">Имя XML файла.</param>
        /// <returns>The city list.</returns>
        /// <exception cref="FileNotFoundException">Файл не
найден.</exception>
        /// <exception cref="InvalidCastException">Неверный
формат данных файла.</exception>
    }
}

```

```

///City.cs

using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace tspp
{
    /// <summary>
    /// Обеспечивание формирования и хранения позиции отдельного
города (свойство Location).
    /// </summary>
    public class City
    {
        /// <summary>
        /// Конструктор, обеспечивающий формирование позиции
города.
        /// </summary>
        /// <param name="x">X координата города.</param>
        /// <param name="y">Y координата города.</param>
        public Cityy(int x, int y)
        {
            Location = new Point(x, y);
        }

        /// <summary>
        /// Позиция города (private копия).
        /// </summary>
        private Point location;
        /// <summary>
        /// Позиция города (public).
        /// </summary>
        public Point Location
        {
            get
            {
                return location;
            }
        }
    }
}

```

```

        set
        {
            location = value;
        }
    }

    /// <summary>
    /// </summary>
    private List<double> distancees = new List<double>();
    /// <summary>
    /// </summary>
    public List<double> Distances
    {
        get
        {
            return distancees;
        }
        set
        {
            distancees = value;
        }
    }

    /// <summary>
    /// </summary>
    private List<int> closeCitiies = new List<int>();
    /// <summary>
    /// Список породов-соседей (public).
    /// </summary>
    public List<int> CloseCitiies
    {
        get
        {
            return closeCitiies;
        }
    }

    /// <summary>
    /// </summary>

```

```

        /// <param name="numberOfCloseCitiies">При создании
начальной популяции туров, больше шансов того,
        /// что соседний город будет выбран для связи. Это
количество близлежащих городов, которые будут считаться
соседями.</param>

```

```

/// Link.cs
using System;
using System.Collections.Generic;
using System.Text;

namespace tspp
{
    /// <summary>
    /// Индивидуальная связь между 2 городами в туре.
    /// Этот город соединяется с 2 другими городами.
    /// </summary>
    public class Link
    {
        /// <summary>
        /// Соединение с первым городом.
        /// </summary>
        private int connection1;
        /// <summary>
        /// Соединение с первым городом..
        /// </summary>
        public int Connection1
        {
            get
            {
                return connection1;
            }
            set
            {
                connection1 = value; ;
            }
        }

        /// <summary>

```

```

        /// Соединение со вторым городом.
        /// </summary>
        private int connection2;
        /// <summary>
        /// Соединение со вторым городом.
        /// </summary>
        public int Connection2
        {
            get
            {
                return connection2;
            }
            set
            {
                connection2 = value;
            }
        }
    }
}

```

```

        /// Population.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace tspp
{
    /// <summary>
    /// Класс Population обеспечивает формирование начальной
    популяции.
    /// </summary>
    class Population : List<Tour>
    {
        /// <summary>
        /// Лучший маршрут (private-копия), найденный
        генетическим алгоритмом.
        /// </summary>
        private Tour bestTour = null;
    }
}

```

```

        /// <summary>
        /// Лучший маршрут (public), найденный генетическим
алгоритмом.
        /// </summary>
        public Tour BestTour
        {
            set
            {
                bestTour = value;
            }
            get
            {
                return bestTour;
            }
        }
    }

```

```

///Program.cs
using System;
using System.Windows.Forms;

namespace tspp
{
    /// <summary>
    /// Содержит Main-функцию запуска формы.
    /// </summary>
    static class Program
    {
        /// <summary>
        /// Точка входа приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();

            //true - элементы управления используют для отрисовки
текста класс Graphics
            //false -//- класс TextRenderer основан на GDI

```

```

        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new tsppForm());
    }
}

/// tspp.cs
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Text;
using System.Data;
using System.Drawing;

namespace tspp
{
    /// <summary>
    /// Этот класс выполняет алгоритм задачи коммивояжера.
    /// </summary>
    class tspp
    {
        /// <summary>
        /// </summary>
        /// <param name="sender">Объект, создавший
событие.</param>
        public delegate void NewBestTourEventHandler(Object
sender, tsppEventArgs e);

        /// <summary>
        /// Событие срабатывает, когда найден новый лучший тур.
        /// </summary>
        public event NewBestTourEventHandler foundNewBestTour;

        /// <summary>
        /// Объект генератора случайных чисел.
        /// GUI разрешается установить начальное число для
генератора случайных чисел, чтобы помочь при отладке.
        /// </summary>
        Random rand;
    }
}

```

```

    /// <summary>
    /// Список городов. Используется только для расчета
расстояния между городами.
    /// </summary>
    Cities cityList;

    /// <summary>
    /// Полный список всех маршрутов.
    /// </summary>
    Population population;

    /// <summary>
    /// </summary>
    private bool halt = false;
    /// <summary>
    /// GUI устанавливает этот флаг==true для остановки
алгоритма tspp и позволяет функции Begin() вернуть значение
(return).
    /// </summary>
    public bool Halt
    {
        get
        {
            return halt;
        }
        set
        {
            halt = value;
        }
    }

    /// <summary>
    /// Конструктор
    /// </summary>
    public tspp()
    {
    }

```

```

        if (topTour != tourCount)
        {
            tempTour = tourGroup[tourCount];
            tourGroup[tourCount] = tourGroup[topTour];
            tourGroup[topTour] = tempTour;
        }
    }

    bool foundNewBestTour = false;

    // Берется лучших 2 тура, делается скрещивание
    (кроссовер). Результат - это дочерний маршрут.
    // Худший тур заменяется полученным
    // Находим длину дочернего маршрута
    population[childPosition].DetermineFitness(cityList);

    return foundNewBestTour;
}

/// <summary>
/// <param name="generationNumber">Число порожденных
поколений.</param>
/// <param name="complete">Если true, то вычисления
завершаются.</param>
}
}

/// tsppEventArgs.cs
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Drawing;

namespace tspp
{
    /// <summary>
    public class tsppEventArgs : EventArgs
    {
        /// <summary>

```

```

    /// Конструктор по умолчанию.
    /// </summary>
    public tsppEventArgs()
    {
    }

    /// <summary>
    /// Конструктор, устанавливающий значения свойств
события.
    /// </summary>
    /// <param name="cityList">Список городов для
отображения.</param>
    /// <param name="bestTour">Лучший маршрут соединяющий все
города.</param>
    /// <param name="generation">Номер поколения.</param>
    /// <param name="complete">Флаг завершения
алгоритма.</param>

    /// <summary>Список городов (private копия).</summary>
    private Cities cityList;
    /// <summary>Список городов (public).</summary>
    public Cities CityList
    {
        get
        {
            return cityList;
        }
    }

    /// <summary>Лучший маршрут (private копия).</summary>
    private Tour bestTour;
    /// <summary>Лучший маршрут (public).</summary>
    public Tour BestTour
    {
        get
        {
            return bestTour;
        }
    }

```

```

        public int Generatiion
        {
            get
            {
                return generatiion;
            }
            set
            {
                generatiion = value;
            }
        }

        private bool complete = false;
        /// <summary>Флаг завершения алгоритма
(public).</summary>
        public bool Complete
        {
            get
            {
                return complete;
            }
            set
            {
                complete = value;
            }
        }
    }
}

```

```

///TspForm.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;

```

```

using System.IO;
using System.Globalization;

namespace tspp
{
    /// <summary>
    /// Главная форма Travelling Salesman Problem (TSP).
    /// </summary>
    public partial class tsppForm : Form
    {
        /// <summary>
        /// </summary>
        Citiies cityList = new Citiies();

        /// <summary>
        tspp tspp;

        /// <summary>
        /// Образ для отображения городов и маршрута.
        /// </summary>
        Image cityImage;

        Graphics cityGraphics;

        /// <summary>
        /// Delegate (класс хранящий ссылку на метод с
определенной сигнатурой) для потока запускающего tspp алгоритм.
        /// </summary>
        /// <param name="sender">Объект генерирующий
событие.</param>
        /// <param name="e">Аргументы события.</param>
        public delegate void DrawEventHanddler(Object sendeer,
tsppEventArgs ee);

        /// <summary>
        /// Стандартный конструктор.
        /// </summary>
        public tsppForm()
        {

```

```

        InitializeComponent();
    }

    /// <summary>
    /// </summary>
    /// <param name="sender">Объект генерирующий
событие.</param>
    /// <param name="e">Аргументы события.</param>

    DrawTour(sender, e);
}

    /// <summary>
    /// Алгоритмом tsprr получен новый «лучший» тур
    /// Отображает последний лучший маршрут
    /// Обновляет соответствующие поля (label)
    /// </summary>
    /// <param name="sender">Объект генерирующий
событие.</param>
    /// <param name="e">Аргументы события.</param>
    public void DrawTouur(object sender, tsppeventArgs e)
    {
        this.lastFitnesssValue.Text =
Math.Round(e.BestTour.Fitnesss,
2).ToString(CultureInfo.CurrentCulture);
        this.lastIterationValue.Text =
e.Generation.ToString(CultureInfo.CurrentCulture);

        if (cityImage == null)
        {
            cityImage = new Bitmap(tourDiagram.Width,
tourDiagram.Height);
            cityGraphics = Graphics.FromImage(cityImage);
        }

        int lastCity = 0;
        int nextCity = e.BestTour[0].Connection1;

```

```

        cityGraphics.FillRectangle(Brushes.White, 0, 0,
cityImage.Width, cityImage.Height);
        foreach( City city in e.CityLiist )
        {
            // Отрисовка круга для обозначения города.
            cityGraphics.DrawEllipse(Pens.Black,
ciity.Location.X - 2, ciity.Location.Y - 2, 5, 5);

            // Отрисовка линии соединяющей два города.
            cityGraphics.DrawLine(Pens.Black,
cityList[lastCity].Location, cityList[nextCity].Location);

            // ВЫЯСНИТЬ, является ли следующий город в списке
[0] или [1]
            if (lastCity != e.BestTour[nextCity].Connection1)
            {
                lastCity = nextCity;
                nextCity = e.BestTour[nextCity].Connection1;
            }
            else
            {
                lastCity = nextCity;
                nextCity = e.BestTour[nextCity].Connection2;
            }
        }

        this.tourDiagram.Image = cityImage;

        if (e.Complete)
        {
            StartButton.Text = "Пуск";
            StatusLabel.Text = "Откройте файл со списком
городов или укажите города на карте мышкой.";
            StatusLabel.ForeColor = Coloor.Black;
        }
    }

    /// <summary>
    /// </summary>

```

```

        /// <param name="cityList">Список городов для отображения
(рисования).</param>
        private void DrawCityLiist(Citiies cityList)
        {
            Image cityImage = new Bitmap(tourDiagram.Width,
tourDiagram.Height);
            Graphics graphics = Graphics.FromImage(cityImage);

            foreach (City city in cityList)
            {
                // Рисует окружность для обозначения города.
                graphics.DrawEllipse(Pens.Black, city.Location.X
- 2, city.Location.Y - 2, 5, 5);
            }

            this.tourDiagram.Image = cityImage;

            updateCityCount();
        }

        /// <summary>
        /// </summary>
        /// <param name="sender">Объект генерирующий
событие..</param>
        /// <param name="e">Аргументы события.</param>
        catch (FormatException)
        {
        }

        if (populationSize <= 0)
        {
            MessageBox.Show("Задайте размер популяции",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1
,MessageBoxOptions.DefaultDesktopOnly);
            return;
        }
        if (maxGenerations <= 0)
        {

```

```

        MessageBox.Show("Задайте максимальное число
поколений", "Ошибк", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
        return;
    }
    if ((mutation < 0) || (mutation > 100))
    {
        MessageBox.Show("Задайте процент вероятности
мутации", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
        return;
    }
    if ((groupSize < 2) || (groupSize > populationSize
))
    {
        MessageBox.Show("Задайте размер рабочей группы
задается в пределах от 2 до размеров популяции", "Ошибкаа",
MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
        return;
    }
    return;
}
if ((chanceUseCloseCity < 0) || (chanceUseCloseCity >
95))
{
    MessageBox.Show("Задайте вероятность города-
соседа при начальной популяции", "Ошибк", MessageBoxButtons.OK,
MessageBoxIcon.Error, MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
    return;
}
if (seed < 0)
{
    MessageBox.Show("Укажите seed RNG", "Ошибк",
MessageBoxButtons.OK, MessageBoxIcon.Error,

```

```

MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
        return;
    }
    if (cityList.Count < 5)
    {
        MessageBox.Show("Загрузите файл списка городов,
либо щелкните по карте, чтобы разместить как минимум 5 городов",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error,
MessageBoxDefaultButton.Button1,
MessageBoxOptions.DefaultDesktopOnly);
        return;
    }

}

/// <summary>
/// Создает объект класса tsp.
/// Эта функция выполняется в потоке пула потоков.
/// </summary>
/// <param name="stateInfo">не исп</param>
private void BeginTsp(Object stateInfo)
{
    int populationSize =
Convert.ToInt32(populationSizeTextBoxx.Text,
CultureInfo.CurrentCulture);
    int maxGenerations =
Convert.ToInt32(maxGenerationTextBoxx.Text,
CultureInfo.CurrentCulture); ;
    int mutation = Convert.ToInt32(mutationTextBoxx.Text,
CultureInfo.CurrentCulture);
    int groupSize =
Convert.ToInt32(groupSizeTextBoxx.Text,
CultureInfo.CurrentCulture);
    int seed = Convert.ToInt32(randomSeedTextBoxx.Text,
CultureInfo.CurrentCulture);
    int numberOfCloseCitiies =
Convert.ToInt32(NumberCloseCitiiesTextBoxx.Text,
CultureInfo.CurrentCulture);

```

```

        int chanceUseCloseCity =
Convert.ToInt32(CloseCityOddsTextBoxx.Text,
CultureInfo.CurrentCulture);

cityyList.CalculateCityDistances(numberOfCloseCitiies);

        tspp = new tspp();
        tspp.foundNewBestTour += new
tspp.NewBestTourEventHandler(tsp_foundNewBestTour);
        tspp.Begin(populationSize, maxGenerations, groupSize,
mutation, seed, chanceUseCloseCity, cityList);
        tspp.foundNewBestTour -= new
tspp.NewBestTourEventHandler(tsp_foundNewBestTour);
        tspp = null;
    }

    /// <summary>
    /// Пользователь выбирает новый XML-файл списка городов.
    /// Не может быть использовано если запущен tspp
алгоритм.
    /// </summary>
    /// <param name="sender">Объект, сгенерировавший это
событие.</param>
    /// <param name="e">Аргументы события.</param>

    /// <summary>
    /// Пользователь выбрал "открыть список городов".
    /// Формирование списка городов по XML-файлу.
    /// Не может быть использовано если запущен tspp
алгоритм.
    /// </summary>
    /// <param name="sender">Объект, сгенерировавший это
событие.</param>
    /// <param name="e">Аргументы события.</param>

    /// <summary>
    /// Пользователь выбрал "Очистить список городов".

```

```

        /// Не может быть использовано если запущен tsp
алгоритм.
        /// </summary>
        /// <param name="sender">Объект, сгенерировавший это
событие.</param>
        /// <param name="e">Аргументы события.</param>
        private void clearCityListButton_Click(object sender,
EventArgs e)
        {
            if (tspp != null)
            {
                StatusLabel.Text = "Невозможно изменить городов
во время работы ";
                StatusLabel.ForeColor = Color.Red;
                return;
            }

            cityList.Clear();
            this.DrawCityList(cityList);
        }

        /// <summary>
        /// Пользователь кликнул на карте городов
        /// Не может быть использовано если запущен tsp
алгоритм.
        /// Размещение нового города на карте и добавление его в
список городов.
        /// </summary>
        /// <param name="sender">Объект, сгенерировавший это
событие.</param>
        /// <param name="e">Аргументы события.</param>

        /// <summary>
        /// Отображение количества городов на форме.
        /// </summary>
        private void updateCityCount()
        {
            this.NumberCitiesValue.Text =
cityList.Count.ToString();

```

}
}
}

ДОДАТОК Д
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

Дипломна кваліфікаційна робота

**Дослідження та програмна реалізація
генетичних методів розв'язання завдань комбінаторної
оптимізації**

Виконав

ст. гр. КНТ-110м

Б. В. Короткий

Керівник

к.т.н., доцент

Є. О. Гофман

Рисунок Д.1 – Слайд 1

Об'єктом дослідження є процес комбінаторної оптимізації.

Предмет дослідження – генетичні методи розв'язання завдань комбінаторної оптимізації.

Мета роботи – дослідження та реалізація генетичних методів розв'язання завдань комбінаторної оптимізації.

Рисунок Д.2 – Слайд 2

Задачі роботи:

- дослідження проблемної області з комбінаторної оптимізації;
- аналіз методів розв’язання завдань комбінаторної оптимізації;
- розробка удосконаленого генетичного методу розв’язання задачі комівояжера;
- проектування програмного забезпечення для розв’язання задачі комівояжера на основі генетичної оптимізації;
- розробка програмного забезпечення для розв’язання задачі комівояжера на основі генетичної оптимізації;
- тестування та експериментальне дослідження розробленої програми.

3

Рисунок Д.3 – Слайд 3

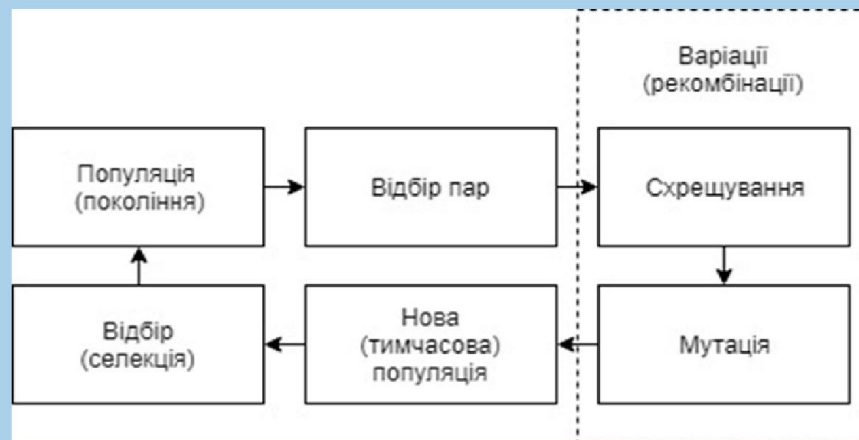
Аналіз методів розв’язання завдань комбінаторної оптимізації

Критерії	Алгоритм повного перебору	Метод гілок і меж	Метод гілок і меж + «жадібний» пошук	BV метод	Генетичний алгоритм
Простота реалізації	+	+	+	-	-
Висока часова залежність від розмірності задачі	+	+	+	-	-
Можливість розпаралелювання	+	+	+	Тільки на етапі оптимізації	+
Можливість досягнення оптимального рішення	+	+	+	-	-

4

Рисунок Д.4 – Слайд 4

Генетичні методи розв'язання завдань комбінаторної оптимізації



5

Рисунок Д.5 – Слайд 5

Вибір мови програмування

Критерії порівняння	Мова програмування		
	C#	Java	Python
Зручність використання для створення програмного забезпечення для розв'язання задачі комівояжера	+	+-	-
Методи та засоби розробки інтерфейсу користувача	+	+-	+-
Підтримка засобів процедурного програмування	+	-	+
Засоби програмування з використанням об'єктно-орієнтованої технології	+	+	+-
Можливість роботи з прототипами	+	-	-
Засоби роботи з базами даних	+	+	+

6

Рисунок Д.6 – Слайд 6

Порівняльний аналіз середовищ розробки для створення програмного забезпечення

Критерії порівняння	Середовище розробки		
	Microsoft Visual Studio	SharpDevelop	MonoDevelop
Зручність використання для створення програмного забезпечення для розв'язання задачі комівояжера	+	+	–
Можливість роботи з репозиторіями	+	+	–
Платформно-незалежність	+	–	+
Наявність візуальних засобів розробки	+	+	+
Швидкість модифікації створюваного ПЗ	+	–	+

7

Рисунок Д.7 – Слайд 7

Модифікований генетичний метод для розв'язання задачі комбінаторної оптимізації

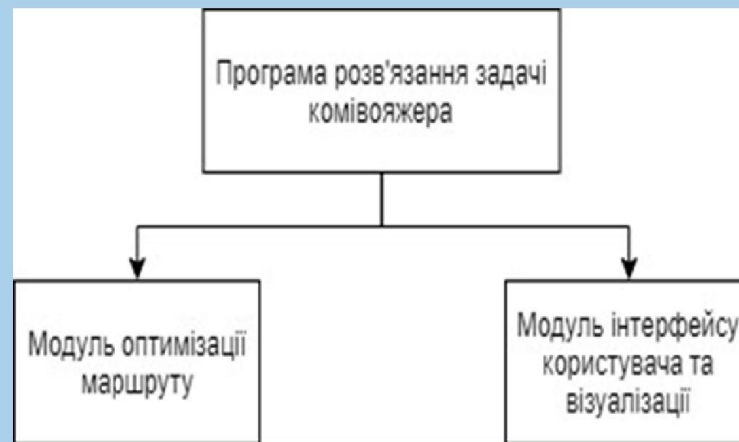
Структура зберігання списку міст туру та їх взаємозв'язків

Місто	Перший зв'язок	Другий зв'язок
A	F	B
B	A	E
C	E	G
D	G	F
E	B	C
F	D	A
G	C	D

8

Рисунок Д.8 – Слайд 8

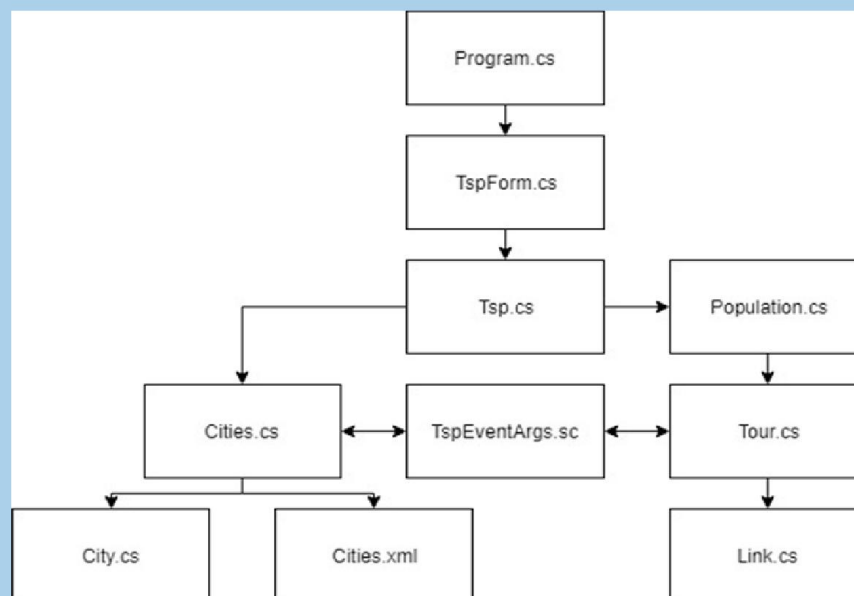
Структурна схема програми



9

Рисунок Д.9 – Слайд 9

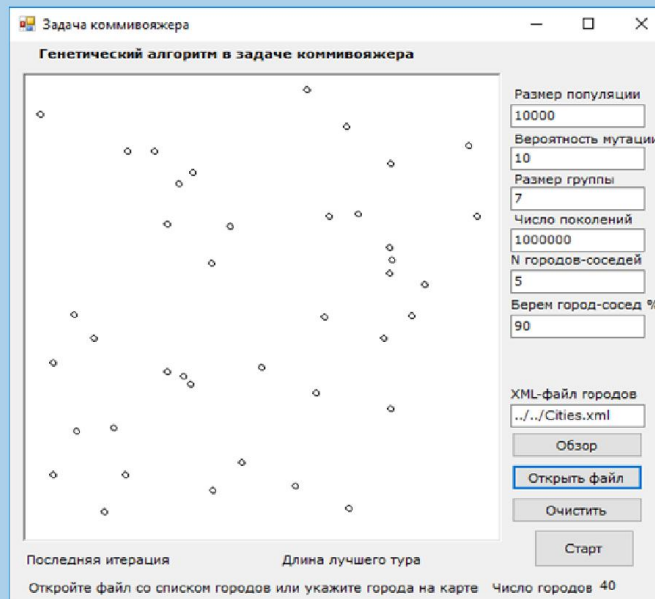
Діаграма файлових залежностей проєкту



10

Рисунок Д.10 – Слайд 10

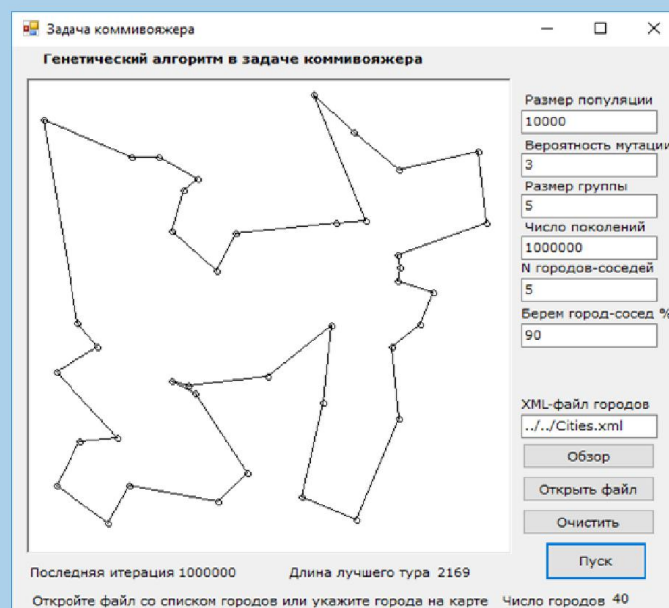
Приклад роботи з програмою. Початкові дані



11

Рисунок Д.11 – Слайд 11

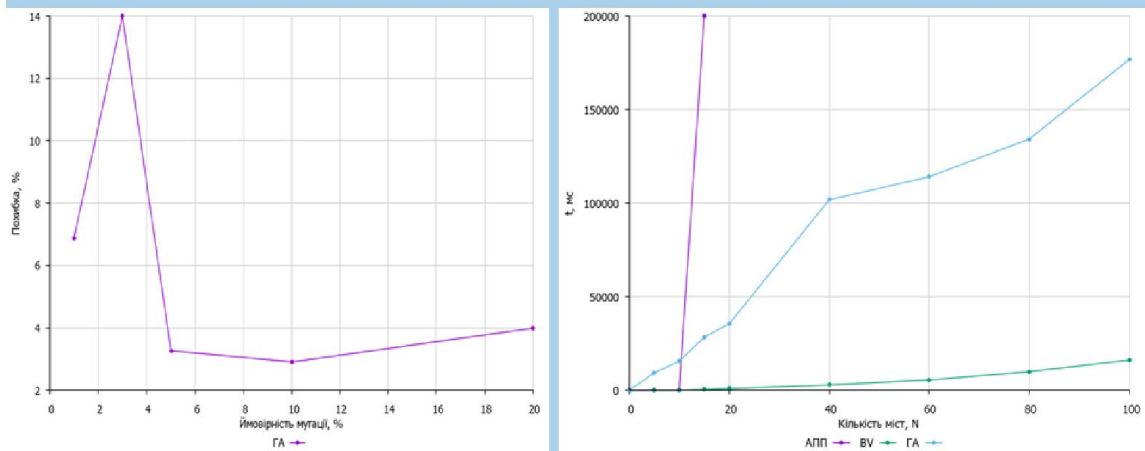
Приклад роботи з програмою. Результати роботи програми



12

Рисунок Д.12 – Слайд 12

Дослідження розробленого програмного забезпечення



Помилка ГА в залежності від зміни ймовірності мутації

Часова залежність роботи різних методів в залежності від розмірності задачі

13

Рисунок Д.13 – Слайд 13

Висновки

В ході виконання даної дипломної кваліфікаційної роботи магістра було проведено огляд та аналіз генетичних методів розв'язання завдань комбінаторної оптимізації.

В процесі розробки вирішено такі завдання:

- досліджено проблемну область з комбінаторної оптимізації;
- проаналізовано сучасні методи розв'язання завдань комбінаторної оптимізації;
- розроблено удосконалений генетичний метод розв'язання задачі комівояжера;
- виконано проектування програмного забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації;
- розроблено програмне забезпечення для розв'язання задачі комівояжера на основі генетичної оптимізації;
- виконано тестування та експериментальне дослідження розробленої програми для розв'язання задачі комівояжера на основі генетичної оптимізації.

14

Рисунок Д.14 – Слайд 14

Висновки

Наукова новизна роботи полягає у тому, що розроблено удосконалений генетичний метод розв'язання задачі комбінаторної оптимізації (задачі комівояжера), що, на відміну від існуючих, використовує модифікований спосіб кодування вхідних даних, що дозволяє окремо зберігати кожне місто туру та зв'язки для нього в обох напрямках, і відповідно до цього, розроблено поліпшений оператор схрещування, що дозволяє уникати проблеми створення помилкових маршрутів і дублювання вже існуючих, що, в свою чергу, дозволяє зменшити час розв'язання задачі комівояжера..

Практичне значення роботи полягає у тому, що розроблено програмне забезпечення для розв'язання задачі комівояжера.

Проведені експерименти засвідчили, що програма має достатню працездатність і може використовуватися за призначенням для розв'язання задачі комівояжера.

15

Рисунок Д.15 – Слайд 15