Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки,
Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр
(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МУЛЬТИАГЕНТНИХ МЕТОДІВ ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ.
RESEARCH AND SOFTWARE IMPLEMENTATION OF MULTIAGENT METHODS FOR THE OPTIMAL ROUTE FINDING

Виконав: студент 2 курсу, групи КНТ-129м
спеціальності
121 Інженерія програмного забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Єдноралюк І. В.
(прізвище та ініціали)
Керівник Зайко Т.А., к.т.н., доцент
(прізвище та ініціали)
Рецензент Скрупський С.Ю., к.т.н., доцент
(прізвище та ініціали)

2020

Форма № 25

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

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

ЗАТВЕРДЖУЮ
Завідувач кафедри проф., д.т.н.
С. О. Субботін
“__” 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

Єдноралюка Івана Вікторовича
(прізвище, ім’я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація мультиагентних методів пошуку оптимального маршруту. Research and Software Implementation of Multiagent Methods for the Optimal Route Finding

керівник проєкту (роботи) Зайко Т.А., к.т.н., доцент ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від “3”листопада 2020 року №301
2. Строк подання студентом проєкту (роботи) 5 грудня 2020 року
3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Вибір та обґрунтування структури системи, яка проєктується. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. 6. Економіко-організаційна частина. 7. Охорона праці та безпека у надзвичайних ситуаціях.

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
1–5	Зайко Т.А., доцент		
6	Пожуєва Т.О., професор		
7	Коробко О.В., ст. викладач		
Нормо-контроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання_____02.10.20_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз предметної області	1 тиждень	
2	Розробка та затвердження технічного завдання	2 тиждень	Технічне завдання
3	Проєктування структури системи	3-4 тиждень	Структурна схема
4	Розробка схеми функціонування системи та модулів програми	5-6 тиждень	Функціональна схема. Модулі
5	Створення програмного коду системи	7-8 тиждень	Текст програми
6	Тестування та відлагодження програми	9 тиждень	Працездатна система
7	Розробка розділів пояснювальної записки	10 тиждень	ПЗ
8	Розробка слайдів презентації	11-12 тиждень	Слайди презентації
9	Захист роботи	13 тиждень	

Студент(ка) _____ Єдноралюк І.В.
(підпис) (прізвище, ініціали)

Керівник проєкту (роботи) _____ Зайко Т.А.
(підпис) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра: 100 с., 27 рис., 4 додатки, 25 джерел.

МЕТОД МУРАШИНИХ КОЛОНІЙ, МУРАШИНИЙ АЛГОРИТМ, ЗАДАЧА КОМІВОЯЖЕРА, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, C#.

Об'єкт дослідження – процес розв’язання задач комбінаторної оптимізації.

Предмет дослідження – методи та програмні засоби розв’язання задачі комівояжера.

Мета роботи – дослідження та розробка реалізація мультиагентних методів пошуку оптимального маршруту.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, мова програмування C#, персональний комп’ютер з монітором, клавіатурою та маніпулятором типу «миша», що функціонує під управлінням операційної системи.

Результати. Створено програмні засоби для пошуку оптимального шляху на основі мультиагентного підходу. Розроблена програмна система може успішно експлуатуватися на персональних комп’ютерах під управлінням операційних систем та використовуватися для розв’язання завдань дискретної оптимізації.

Висновки. Розроблено програмне забезпечення для розв’язання завдань дискретної оптимізації основі мультиагентного підходу. Проаналізовано предметну область, розроблено технічне завдання для реалізації програми, обрано середовище розробки, розроблено структурну схему програми, виконано конструювання програмного забезпечення для пошуку оптимального шляху, здійснено тестування розробленого програмного забезпечення.

Галузь використання – дискретна оптимізація.

ABSTRACT

Explanatory note to the final qualifying work of the magister: 100 pages, 27 figures, 4 appendixes, 25 sources.

ANT COLONY METHOD, ANT ALGORITHM, TRAVELLING SALESMAN PROBLEM, SOFTWARE, C #.

Object of study is a process of combinatorial optimization problems solving.

Subject of study are methods and software for solving the travelling salesman problem.

The purpose of the work is to study and develop a of multi-agent methods for finding the optimal route.

Materials, methods, and tools: structured and object-oriented programming, programming language C#, computer with a monitor, keyboard and mouse manipulator operating under the control of the operating system.

Results. The software for finding the optimal path based on a multi-agent approach has been created. The developed software system can successfully operate on personal computers running operating system and can be used for discrete optimization problems solving.

Conclusions. Software for solving discrete optimization problems based on a multi-agent approach has been developed. The subject area is analyzed, the technical task for program realization is developed, the development environment is chosen, the structural scheme of the program is developed, the software for search of an optimum way is executed, the developed software is tested.

The field of use is discrete optimization.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ГА – генетичний алгоритм;

ЕА – еволюційний алгоритм;ОС – операційна система;

ПЗ – програмне забезпечення;

ПО – предметна область;

ПК – персональний комп'ютер;

ПП – програмний продукт;

ШІ – штучний інтелект.

ВСТУП

Вирішення задач оптимізації, а саме задачі комівояжера має багато прикладів практичного використання: оптимальне розташування газового трубопроводу, прокладання мереж, конфігурація транзисторів на платах, дизайн антени системи подачі, сортування об’єктів, розв’язання логістичних завдань та інше [1]-[4].

У загальному випадку задача формулюється так – навістити усі міста із заданого списку, при умові що цей шлях буде найменшим серед усіх можливих шляхів. Тобто існує проблема знаходження оптимального шляху, використовуючи як можна менше обчислювальних потужностей [1].

Існують різні методи, які дозволяють виявити близькі до оптимального розв’язки, зокрема, метод найближчого сусіда або метод рою часток. Такі методи дозволяють виявити субоптимальне рішення задачі комівояжера, затрачуючи при цьому певний час [4]-[23].

Завдання комівояжера дозволяє отримати рішення з використанням різних алгоритмів та методів. Одним з найбільш ефективних методів вважається мультиагентний алгоритм оптимізації наслідуванням мурашиної колонії (ant colony optimization) [6].

Відома одна з програмних реалізацій мурашиного алгоритму, написана на мові C ++ [10], що забезпечує наочне уявлення про механізм роботи алгоритму з можливістю його модифікації. Однак це рішення не дозволяє вносити зміни у наявні дані, крім того воно не забезпечує прийнятного функціоналу для візуального проєктування графів.

Метою роботи є дослідження та розробка реалізація мультиагентних методів пошуку оптимального маршруту.

Для досягнення визначеної мети необхідно було вирішити такі завдання:

– аналіз методів вирішення задачі комівояжера;

– розробка технічного завдання для реалізації програми;

– вибір середовища розробки програмного забезпечення;

– розробка модифікованого мультиагентного методу для розв’язання задачі комівояжера;

– конструювання програмного забезпечення пошуку оптимального маршруту на основі мультиагентного підходу;

– тестування розробленого програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз задачі комівояжера

Завдання комівояжера полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів [1]-[4].

Існують різні варіанти узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає відстані між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується нерівність трикутника), симетрична та асиметрична задачі комівояжера [1]-[4].

Прості методи розв'язання задачі комівояжера: повний лексичний перебір, жадібні алгоритми (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального кістяка дерева. На практиці застосовують різні модифікації ефективніших методів: метод гілок і меж і метод генетичних алгоритмів, а так само алгоритм мурашиної колонії [1]-[4], [12]-[20].

Всі ефективні (такі, що скорочують повний перебір) методи розв'язання задачі комівояжера — евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближений розв'язок. Користуються популярністю так звані any-time алгоритми, тобто алгоритми, що поступово покращують деякий поточний наближений розв'язок [1]-[4], [12]-[20].

Задача комівояжера — NP-повна. Часто на ній проводять випробування нових підходів до евристичного скорочення повного перебору [1]-[4], [12]-[20].

Додатковий інтерес задача являє для дослідників і фахівців в області дискретної оптимізації, оскільки завдання допускає застосування різних алгоритмів для вирішення [1]-[4], [12]-[20].

1.2 Аналіз методів розв'язання задачі комівояжера

Для знаходження рішення задачі комівояжера розроблено різні алгоритми. Серед них: метод повного перебору; жадібний алгоритм; метод гілок і меж; метод імітації відпалу; генетичний алгоритм; мурашиний алгоритм. Алгоритм повного перебору (також відомий як метод грубої сили) здійснює пошук всіх рішень шляхом перебору всіх варіантів в кількості N! штук, дозволяючи отримати точне рішення задачі. Це є головною перевагою алгоритму, ще одна перевага – можливість розпаралелювання. Головний недолік методу повного перебору – тимчасові витрати. Тому, якщо число N не є малим, застосовуються евристичні і пошукові алгоритми [2].

Всі алгоритми, що скорочують повний перебір, не дають точного значення. З їх допомогою можна знайти тільки наближене рішення задачі, при цьому не гарантується, що похибка обчислень буде низька. Так, наприклад, при використанні жодного алгоритму пошук оптимального шляху починається з випадковою вершини, після чого на кожному кроці здійснюється перехід в місто, який знаходиться найближче до поточного. Пошук в цьому випадку виходить порівняно швидким, проте похибка може виявитися досить великий [1, 3].

Принцип роботи методу гілок і меж полягає в додаванні перевірки критерію обмежує функції, за яким на деякому рівні можна призупинити побудова гілки дерева перестановок. Метод є поліпшенням алгоритму повного перебору і зберігає його позитивні властивості, проте при великих значеннях N його використання все ще неефективно [1, 4].

Метод імітації відпалу, що відноситься до категорії природних обчислень, заснований на процесі кристалізації металів. Мета застосування

методу – привести метал в стан з найменшою енергією. Метал в процесі нагрівається до певної температури, після чого починається контрольоване охолоджування. Метод визначає три функції: функцію енергії, яку необхідно оптимізувати, спадаючу функцію температури і функцію, що визначає новий стан [8].

При використанні методу імітації відпалу в рішенні задачі комівояжера функцією стану є всі можливі маршрути, функцією енергії є найбільш короткий маршрут. Температурної функцією виступає механізм вибору двох довільних міст і інверсії шляхів між ними [1]-[4], [12]-[20].

Алгоритм імітації відпалу має імовірнісну природу і може дати некоректний результат при знаходженні локального мінімуму. Перевагою методу є ефективність обчислення, що перевершує метод повного перебору. Інша перевага – можливість модифікації, що дозволяє домогтися більшого ефекту при обчисленні [1]-[4].

Механізм роботи генетичного алгоритму полягає в пошуку шляхом комбінування і варіацій параметрів, що нагадують біологічну еволюцію. Відмінною здатністю алгоритму є використання 10 операторів схрещування і мутації, які виробляють рекомбінацію рішень-кандидатів, аналогічної ролі схрещування в живій природі [5]. Один з найбільш ефективних алгоритмів, часто застосовуваний на практиці.

Мурашиний алгоритм (алгоритм оптимізації наслідуванням мурашиної колонії) являє собою імітацію поведінки колонії мурах в природі. В основі мурашиного алгоритму лежить імовірнісний підхід до пошуку оптимального шляху, проте мають велике значення додаткові критерії [1]-[4], [12]-[20].

Перевагами алгоритму є невисока похибка знайденого рішення, низькі тимчасові витрати при роботі з графами великої розмірності, модифицируемость алгоритму і можливість розпаралелювання. Алгоритм вважається одним з найефективніших поряд з генетичним і також широко застосовується на практиці [1]-[4].

1.3 Аналіз мультиагентного методу мурашиних колоній

Найвідомішим мультиагентним методом є метод мурашиних колоній. Ідея методу полягає в тому, що агент (мураха) прямує від мурашника (М) у випадковому напрямку. Якщо вона знаходить їжу (Ї), то повертається до мурашника і залишає по собі слід з феромону (рис. 1.1) [1]-[4], [12]-[20].

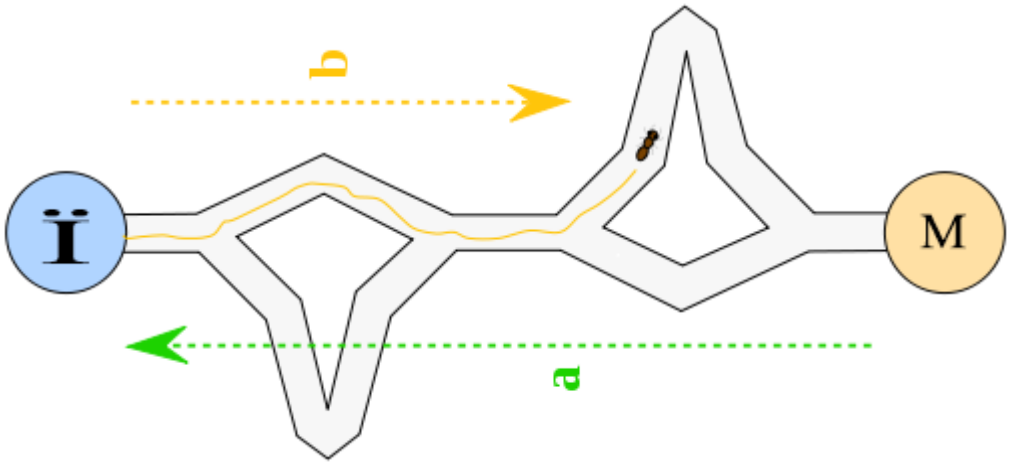


Рисунок 1.1 – Маршрут агента (мурахи)

Ці феромони приваблюють інших мурах, що знаходяться поруч, і скоріш за все вони рушать цим маршрутом. Повертаючись до мурашника, вони також залишають свій феромон і укріплюють доріжку (рис. 1.2) [1]-[4].

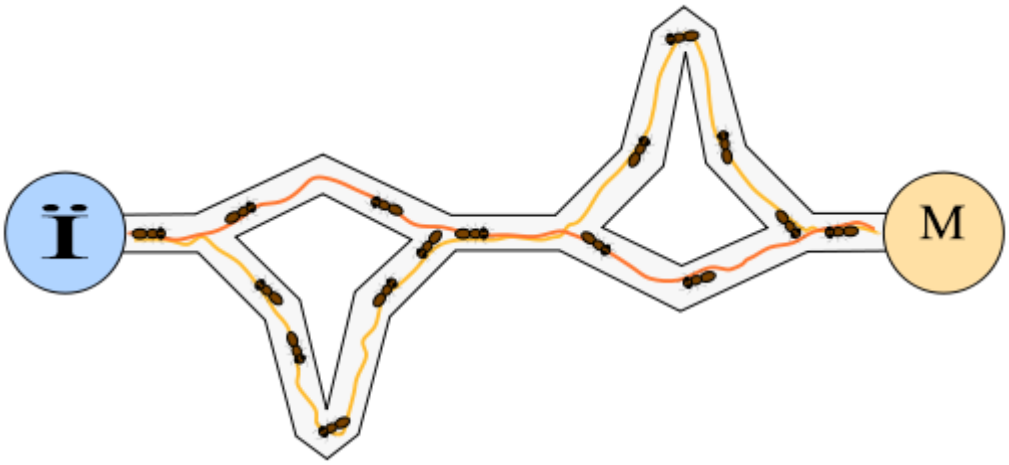


Рисунок 1.2 – Маршрут мурах

Якщо існує два маршрути, то коротким за цей час встигне пройти більше мурах ніж по довгому і короткий маршрут стане більш вигідним. Довші доріжки повільно зникають внаслідок випаровування феромонів (рис. 1.3).

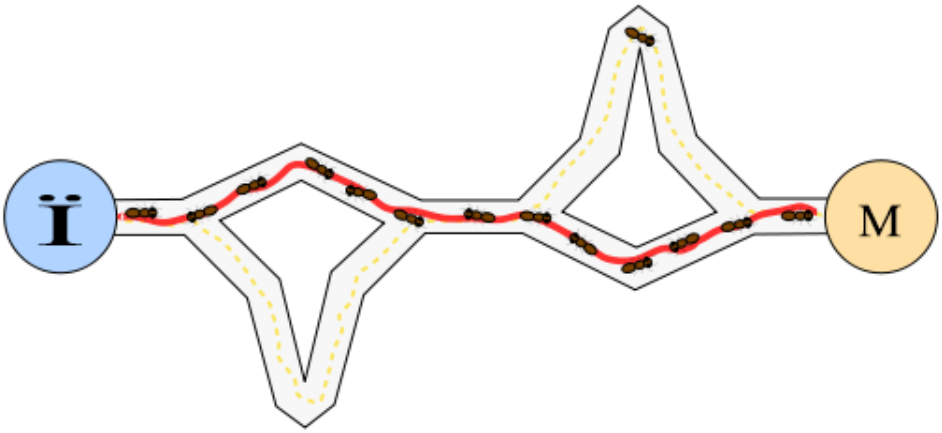


Рисунок 1.3 – Оптимальний маршрут агентів

Базова ідея алгоритму мурахи полягає в оптимізації шляхом непрямого зв'язку між автономними агентами [1]-[4], [12]-[20].

Мурашиний алгоритм моделює багатоагентну систему. Її агентів надалі будемо називати мурахами. Як і справжні мурахи, вони досить просто влаштовані: для виконання своїх обов'язків вони вимагають невелику кількість пам'яті, а на кожному кроці роботи виконують нескладні обчислення. Кожна мураха зберігає в пам'яті список пройдених їм вузлів. Цей список називають списком заборон (tabu list) або просто пам'яттю мурашки. Вибираючи вузол для наступного кроку, мураха «пам'ятає» про вже пройдені вузли і не розглядає їх як можливих для переходу. На кожному кроці список заборон поповнюється новим вузлом, а перед новою ітерацією алгоритму – тобто перед тим, як мурашка знову проходить шлях – він очищується [1]-[4], [12]-[20].

Для алгоритму мурашиної колонії необхідно вказати [1]-[4], [12]-[20]:

– закон виділення феромону;

– закон випаровування феромону;

– кількість агентів;

– місця розміщення.

Ці характеристики вибираються з врахуванням особливостей завдання на основі експериментальних досліджень (евристики). Алгоритм оптимізації мурашиної колонії може бути успішно застосований для вирішення складних комплексних завдань оптимізації. Мета вирішення складних комплексних завдань оптимізації – пошук і визначення найбільш відповідного рішення для оптимізації (знаходження мінімуму або максимуму) цільової функції (ціни, точності, часу, відстані тощо) з дискретної множини можливих рішень. Дослідження показали, що метод мурашиних колоній може давати результати, навіть кращі ніж при використанні генетичних алгоритмів і нейронних мереж [1]-[4], [12]-[20].

1.4 Аналіз модифікацій мультиагентного методу мурашиних колоній

Результати перших експериментів із застосуванням мурашиного алгоритму для вирішення завдання комівояжера були багатообіцяючими, проте далеко не кращими порівняно з вже існуючими методами. Однак, простота класичного мурашиного алгоритму («мурашиної системи») залишала можливості для доробок – і саме алгоритмічні вдосконалення стали предметом подальших досліджень фахівців у галузі комбінаторної оптимізації. В основному, ці вдосконалення пов'язано з великим використанням історії пошуку та більш ретельним дослідженням областей навколо вже знайдених вдалих рішень [1]-[4], [12]-[20].

Elitist Ant System: введення в алгоритм так званих «елітних мурах». Досвід показує, що проходячи ребра, що входять в короткі шляхи, мурахи з більшою ймовірністю будуть знаходити коротші шляхи. Ефективною стратегією є штучне збільшення рівня феромонів на найвдаліших маршрутах. Для цього на кожній ітерації алгоритму кожна з елітних мурах проходить шлях, який є найкоротшим із знайдених на даний момент [1]-[4], [12]-[20].

Ant-Q: мурашиний алгоритм, який отримав свою назву за аналогією з методом машинного навчання Q-learning. В основі алгоритму лежить ідея про

те, що мурашину систему можна інтерпретувати як систему навчання з підкріпленням. Ant-Q підсилює цю аналогію, запозичуючи багато ідей з Q-навчання [1]-[4], [12]-[20].

Ant Colony System: для підвищення ефективності в порівнянні з класичним алгоритмом введено три основних зміни. Рівень феромонів на ребрах оновлюється не лише в кінці чергової ітерації, але і при кожному переході мурах з вузла у вузол. Наприкінці ітерації рівень феромонів підвищується тільки на найкоротшому із знайдених шляхів. Алгоритм використовує змінене правило переходу: або, з певною часткою ймовірності, мураха безумовно вибирає краще ребро у відповідності до довжини і рівня феромонів, або робить вибір так само, як і в класичному алгоритмі [1]-[4], [12]-[20].

Max-min Ant System: мурашиний алгоритм, в якому підвищення концентрації феромонів відбувається тільки на кращих шляхах з пройдених мурахами. Така велика увага до локальних оптимумів компенсується введенням обмежень на максимальну і мінімальну концентрацію феромонів на ребрах, які вкрай ефективно захищають алгоритм від передчасної збіжності до субоптимальних рішень [1]-[4], [12]-[20].

ASrank: модифікація класичного мурашиного алгоритму, в якому в кінці кожної ітерації мурахи ранжуються у відповідно до довжин пройдених ними шляхів. Кількість феромонів, що залишається мурахою на ребрах, таким чином, призначається пропорційно до її позиції. Для ретельного дослідження околу вже знайдених вдалих рішень, алгоритм використовує елітних мурах [1]-[4], [12]-[20].

1.5 Висновки за розділом 1

Проаналізовано задачу комівояжера. Досліджено методи розв’язання задачі комівояжера. Визначено, що для розв’язання задачі комівояжера ефективно можуть застосовуватися мультиагентні методи.

Досліджено мультиагентний метод мурашиних колоній та його модифікації. Відзначено, що метод мурашиних колоній показав себе як дуже гнучкий інструмент, який може мати різні складові частини. Він знайшов широке використання, проте як було визначено у цьому розділі його суттєвим недоліком є значний час роботи, тому актуальною є розробка мультиагентного методу, вільного від зазначених недоліків.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЄКТУЄТЬСЯ

2.1 Дослідження та аналіз вимог до програмного продукту

Метою розробки є створення програмного забезпечення для пошуку оптимального маршруту.

Діловий контекст розробки полягає в тому, що по завершенню реалізації проєкту власник матиме готовий програмний продукт, з якого зможе отримати матеріальну вигоду, автоматизувавши процес пошуку оптимального маршруту.

Функціональними вимогами до програмного продукту (ПП) є такі:

– програма повинна створювати графи певноъ розмірності, дозволяти кінцевому користувачу будувати графи, а також виконувати завантаження графів з файлів форматів .tsp і .xls, що містять координати вершин;

– для створюваних графів повинна надаватися можливість відображення вершин і опциональное відображення ребер із зазначенням ваг;

– програма повинна відображати користувачеві процес виконання мурашиного алгоритму після кожної його ітерації у вигляді зображень: товщина ребер повинна відображати кількість феромону; кращий маршрут, знайдений на момент поточної ітерації, повинен бути виділено окремим кольором; необхідно вказувати номер ітерації, після якої було зафіксовано даний стан графа;

– по завершенні роботи програма повинна відображати довжину маршруту, який було знайденого в результаті роботи алгоритму, а також витрачений час.

Користувач повинен мати можливість переглядати всі зображення, що відображають кроки виконання алгоритму, з можливістю зберігати всі ці зображення.

Розроблений ПП має зберігатися на DVD диску розміром 8ГБ. Данні повинні зчитуватися на платформі Windows.

Для запуску розроблюваного ПП треба:

– процесор частотою 2 Ггц;
– місце на жорсткому диску – 1 Гб;
– відеокарта з пам’яттю 256 Мб;
– оперативна пам’ять – 4 Гб;
– маніпулятор миша.

Розповсюдження ПП повинно проводитись згідно з законам України.

2.2 Вибір інструментарію розробки програмного забезпечення

При розробці програми використовувалася мова C#. C# є дуже близьким родичем мови програмування Java. Мова Java була створена компанією Sun Microsystems, коли глобальний розвиток інтернету поставив задачу роззосереджених обчислень. Взявши за основу популярну мову C++, Java виключила з неї потенційно небезпечні речі (типу вказівників без контролю виходу за межі). Для роззосереджених обчислень була створена концепція віртуальної машини та машинно-незалежного байт-коду, свого роду посередника між вихідним текстом програм і апаратними інструкціями комп'ютера чи іншого інтелектуального пристрою [5]-[6].

Нововведенням C# стала можливість легшої взаємодії, порівняно з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проектів. Якщо програми на різних мовах виконуються на платформі .NET, .NET бере на себе клопіт щодо сумісності програм (тобто типів даних, за кінцевим рахунком) [5]-[6].

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є

розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо [5]-[6].

2.3 Проєктування програмного забезпечення

ПЗ розроблене з використанням принципів ООП та складається з кількох взаємопов'язаних модулів – класів з їх складовими – методами й даними, тому можна зобразити структуру програми за допомогою UML-діаграми класів (рис. 2.1).

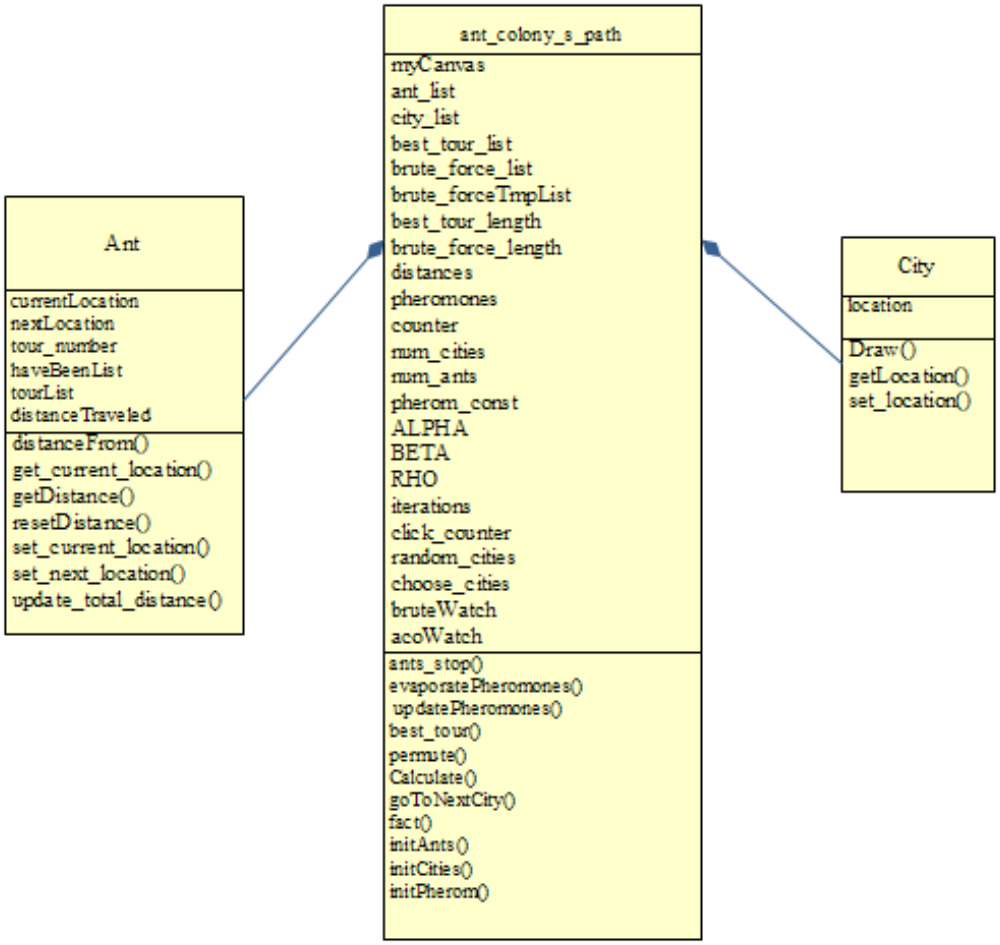


Рисунок 2.1 – UML-діаграма класів

На діаграмі класів видно, що головним класом є ant_colony_s_path. З ним, асоціативним зв’язком(композиція) пов’язані класи Ant та City. Для більш детального огляду функціональної складової системи доречно навести UML-діаграми станів, діяльності, зображені відповідно на рисунках 2.2 та 2.3.

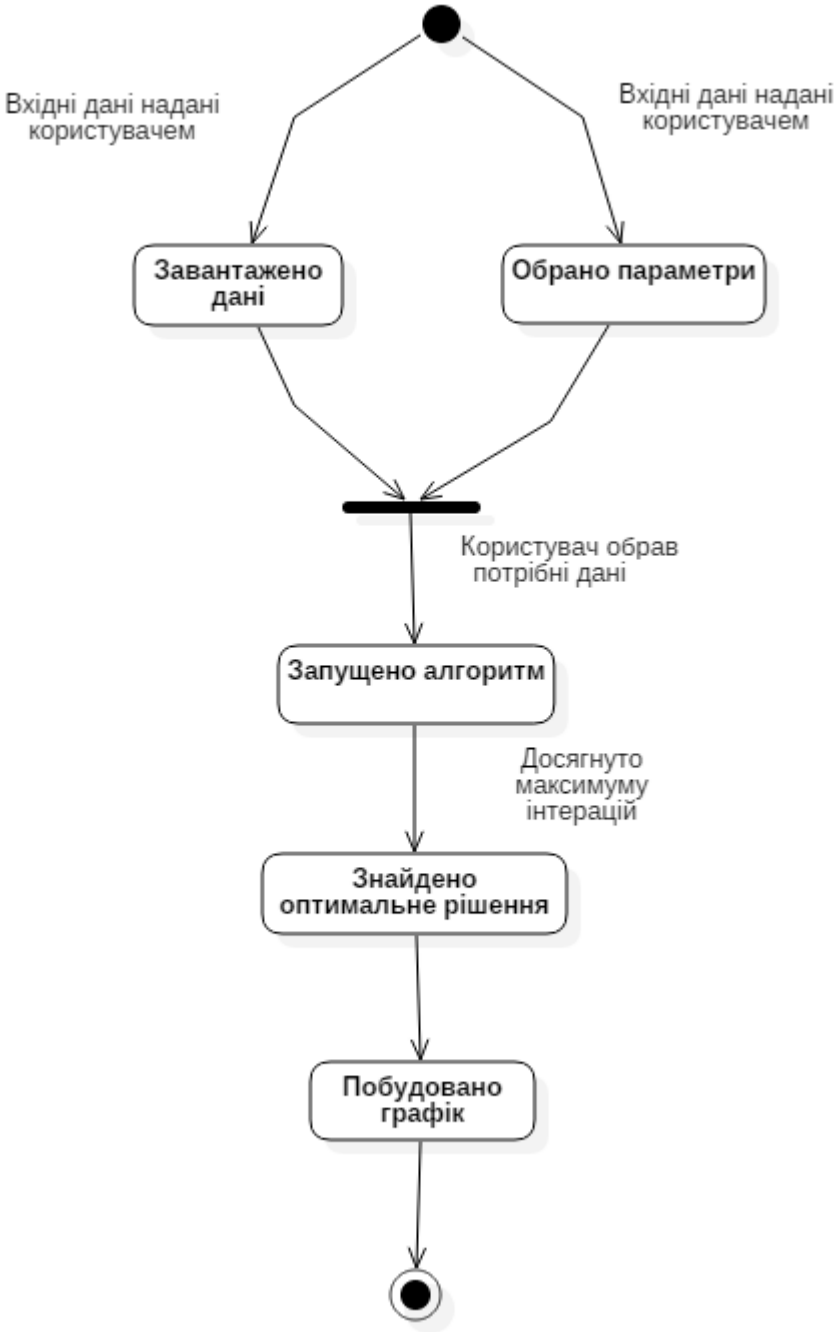


Рисунок 2.2 – UML-діаграма станів

На діаграмі станів зазначено послідовність подій які приводять до зміни абстракцій значень та зв’язків системи, тобто її станів. Діаграма показує що зміна до деяких станів, наприклад, “Запущено алгоритм”, потребує щоб система була у двох попередніх станах – “Завантажено дані”, “Обрано параметри”.

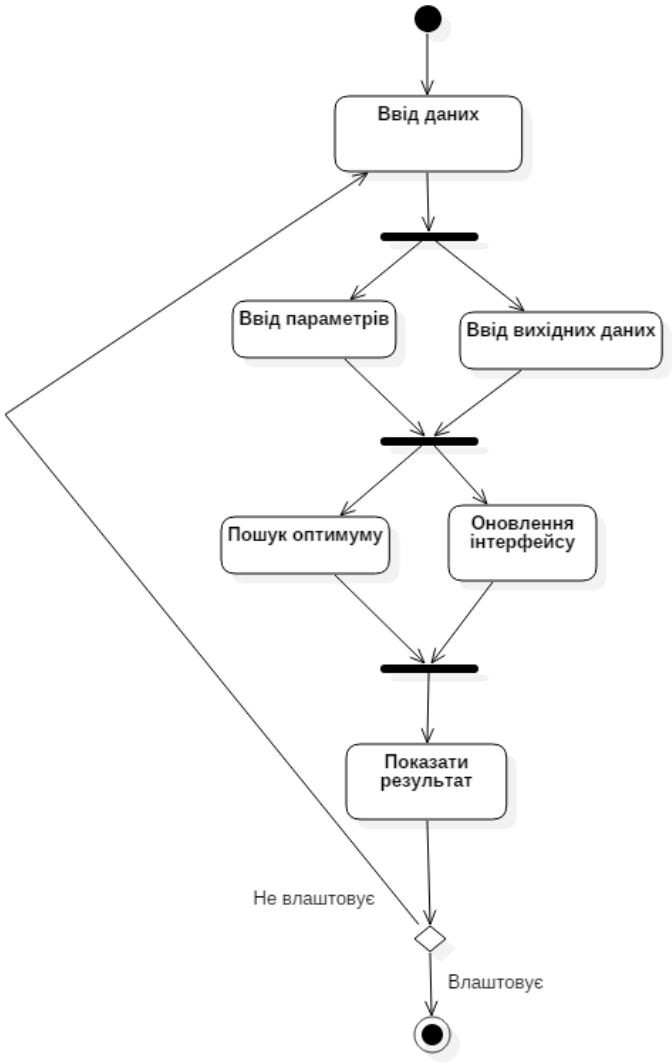


Рисунок 2.3 – UML-діаграма діяльності

На діаграмі діяльності зображено послідовність етапів, які відображають процес роботи методу мурашиних колоній, вона акцентує увагу на операціях та показує потік управління. Операція можуть бути показані паралельно як то “Пошук оптимуму”, “Оновлення інтерфейсу”.

2.4 Висновки за розділом 2

Досліджено та проаналізовано вимоги до програмного забезпечення пошуку оптимального маршруту. Проаналізовано інструментарії розробки програмного забезпечення. Порівнявши відомі середовища розробки програмних засобів, оцінивши їх переваги та недоліки, було прийнято рішення про використання для розробки програмного забезпечення пошуку оптимального маршруту мови C#, оскільки ця мова є досить зручною для використання, дозволяє створювати програмні продукти різного призначення, та має багато бібліотек, що спрощують реалізацію математичних обчислень при створенні програми пошуку оптимального шляху.

Розроблено модель програмного забезпечення пошуку оптимального шляху, що представляє собою набір діаграм у нотації UML, які дозволяють розробнику створювати програмне забезпечення відповідно до вимог замовника.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка модифікованого мультиагентного методу для розв’язання задачі комівояжера

Як відзначено вище, мультиагентні методи знайшли широке використання для розв’язання завдань дискретної оптимізації, проте суттєвим недоліком таких методів є значний час роботи, тому актуальною є розробка мультиагентного методу, здатного знаходити прийнятні рішення за менший час у порівнянні з базовим методом.

У розробленому модифікованому мультиагентному методі для розв’язання задачі комівояжера запропоновано додавати у множину агентів допоміжні «жадібні» мурахи.

Алгоритм модифікованого методу такий.

Крок 1. Мурахи-агенти характеризуються власною пам'яттю. Оскільки кожний пункт призначення повинен бути відвіданим рівно один раз, то у кожного агенту є список таких відвіданих міст (цей список називається списком заборон).

Крок 2. Агенти мають «зір». Під зором будемо розуміти евристичне бажання відвідати певне місто *j*, якщо агент знаходиться у місті *i*. Будемо вважати, що поняття видимості є зворотно пропорційний відстані між відповідними містами.

Крок 3. Агенти наділені здатністю розпізнавати запахи. Агенти мають можливість відчувати слід феромону. Феромон характеризує бажання агенту перейти з міста j до іншого міста і на основі наявного досвіду інших агентів. Кількість фєромону на ребрі (*i*, *j*) у деякий момент часу *t* будемо позначати таким чином: $\tau_{ij}(t)$.

Крок 4. Враховуючи попередню інформацію визначимо правило, за яким будемо визначати імовірність (2.1) переходу *k*-ого агенту з міста *i* до іншого міста *j*:

$$P_{ij,k}(t)=\begin{cases}\dfrac{\left[\tau_{ij}(t)\right]^{\alpha}\cdot\left[n_{ij}\right]^{\beta}}{\sum\limits_{l\in J_{i,k}}\left[\tau_{il}(t)\right]^{\alpha}\cdot\left[n_{il}\right]^{\beta}}, & j\in J_{i,k};\\ 0, & j\notin J_{i,k},\end{cases} \tag{2.1}$$

де α, β – параметри, що визначають значущість (вагу) сліду феромону. У випадку, якщо α = 0 мультиагентний метод вироджується до жадібного алгоритму (внаслідок чого в результаті буде обране найближче місто).

Крок 5. Пройшовши деяке ребро (*i*, *j*), агент залишає на ньому певну кількість феромону. Ця кількість пов'язана з оптимальністю виконаного вибору. Нехай $T_k(t)$ представляє собою шлях, який пройдений агентом *k* до моменту часу *t*, через змінну $L_k(t)$ позначимо довжину шляху, а параметр *Q* буде характеризувати порядок довжини оптимального шляху. Тоді кількість феромону, що залишає агент, може бути визначена за формулою (2.2):

$$\Delta\tau_{ij,k}(t)=\begin{cases}\dfrac{Q}{L_k(t)}, & (i,j)\in T_k(t);\\ 0, & (i,j)\notin T_k(t),\end{cases} \tag{2.2}$$

Правила зовнішнього для агента середовища визначають механізм випаровування феромону. Нехай коефіцієнт випаровування ρ приймає значення у інтервалі [0,1], тоді правило випаровування буде визначатися формулою (2.3):

$$\tau_{ij}(t+1) = (1-p)\tau_{ij}(t) + \Delta\tau_{ij}(t);\ \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \quad (2.3)$$

де m – кількість агентів в колонії.

Таким чином, розроблено модифікований мультиагентний метод для розв'язання задачі комівояжера, в якому у множину агентів додаються допоміжні «жадібні» мурахи, що підсилюють значущість ребер найкращого маршруту, знайденого на попередніх ітераціях роботи методу, що дозволяє скоротити час пошуку оптимального шляху.

3.2 Основні розроблені класи

Клас Ant представляє мурашки, який подорожує між вершинами. Мураха володіє «списком табу», зберігає пройдений маршрут, пройдену відстань, а також знає поточний і наступне розташування. У таблиці 3.1 приведені елементи класу Ant.

Таблиця 3.1 – Список елементів класа Ant

Поле	Опис поля
int currentLocation	Поточне місце розташування мурашки
double distanceTraveled	Пройдена мурахою відстань
List <bool> haveBeenList	«Список табу»
int nextLocation	Наступне місце розташування мурашки
List <int> tourList	Маршрут мурашки
Метод	Опис методу
Ant()	Конструктор за замовчуванням
Ant(int startPos, int num_cities)	Конструктор: створює мурашки в

	вершині startPos і списком табу розміру num_cities

Клас City представляє вершину графа з координатами X, Y. По цих вершинах переміщуються мурахи. У таблиці 3.2 приведені елементи класа City.

Таблиця 3.2 – Список елементів класа City

Поле	Опис поля
Point location	Координати вершини в програмі
Метод	Опис методу
City()	Конструктор за замовчуванням: створює вершину з координатами (0; 0)
City(Point p)	Конструктор: створює вершину з заданими координатами
void Draw(Graphics g)	Малює вершину
void DrawStart(Graphics g)	Малює початкову вершину маршруту
void DrawCurrent(Graphics g)	Малює вершину на поточному кроці
void DrawFinish(Graphics g)	Малює кінцеву вершину маршруту
Point getLocation()	Повертає координати вершини
void setLocation(int x, int y)	Задає координати вершини

3.3 Схема функціонування розробленої програми

На рисунку 3.1 зображено схему функціонування системи.

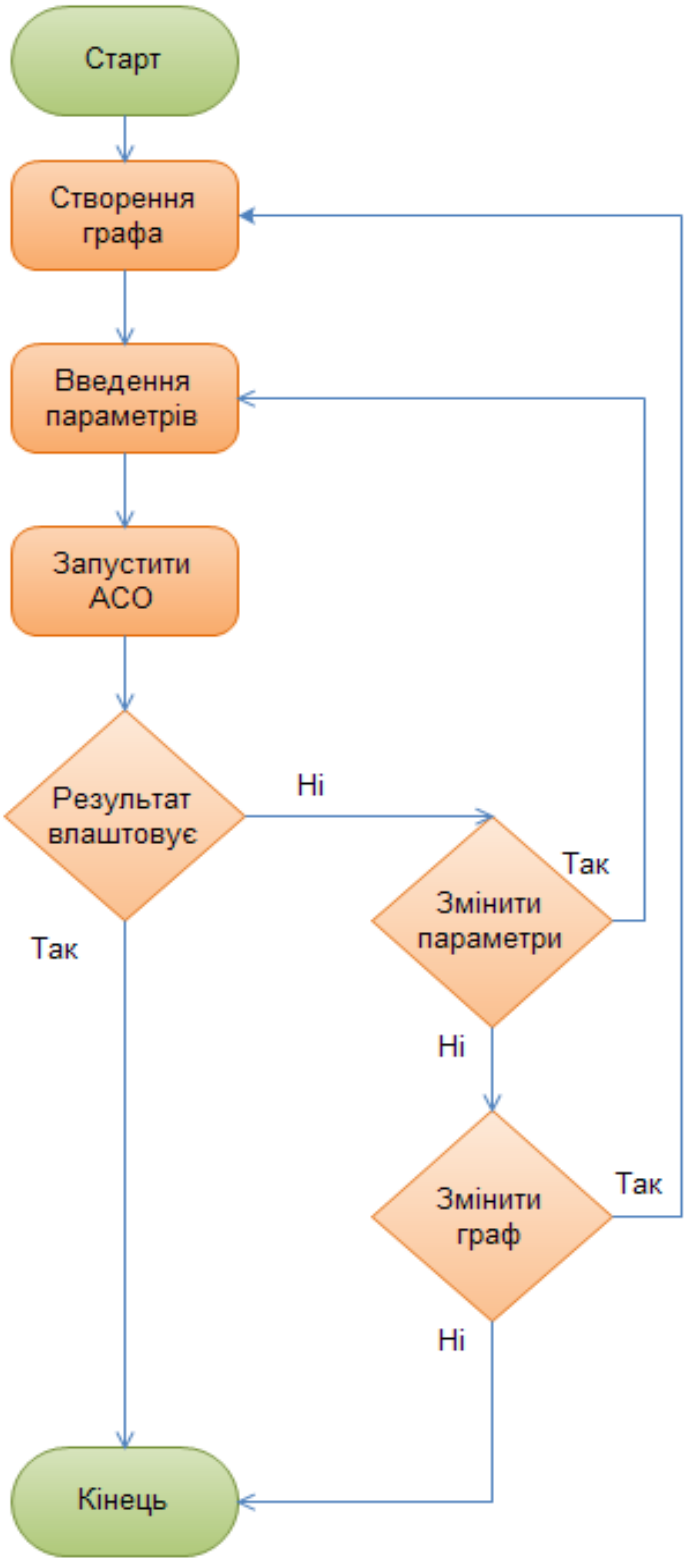


Рисунок 3.1 – Схема функціонування розробленої програми

На початковому етапі користувач повинен створити граф та обрати необхідні параметри.

До параметрів відносяться – кількість ітерацій, кількість мурах, розмір альфа, розмір бета, кількість вершин, коефіцієнт випаровування феромону, феромоний коефіцієнт Q.

Після завантаження даних і запуску алгоритму, почнеться пошук оптимального рішення. Пошук буде виконуватися у новому потоці, тому користувач може змінювати параметри чи граф для підготовки до нового запуску.

3.4 Тестування програмного забезпечення

Було проведено 9 експериментів, що описують ефективність методу мурашиних колоній порівняно з методо повного перебору. Кожен з методів запускався з однаковими параметрами кожен експеримент.

Вхідні дані експериментів: кількість ітерацій, кількість мурах, розмір альфа, розмір бета, кількість вершин, коефіцієнт випаровування феромону, феромоний коефіцієнт Q.

Кожен з експериментів характеризується результуючим графіком та оптимальним значенням відстані.

В якості критеріїв для експериментального порівняння досліджуваних методів використано такі:

- довжина найкращого шляху, що знайдена певним методом;
- швидкість роботи методу.

На рис. 3.2- 3.10 зображено результати експериментів, та у таблиці 3.3 зроблено порівняння методів.

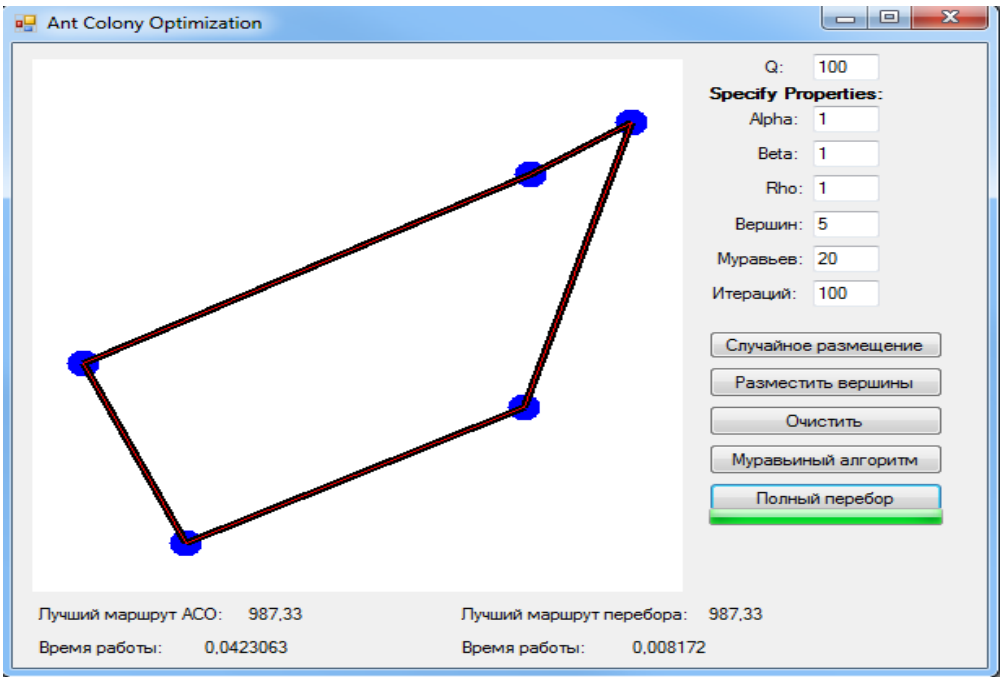


Рисунок 3.2 – Оптимальний шлях 1-го експерименту

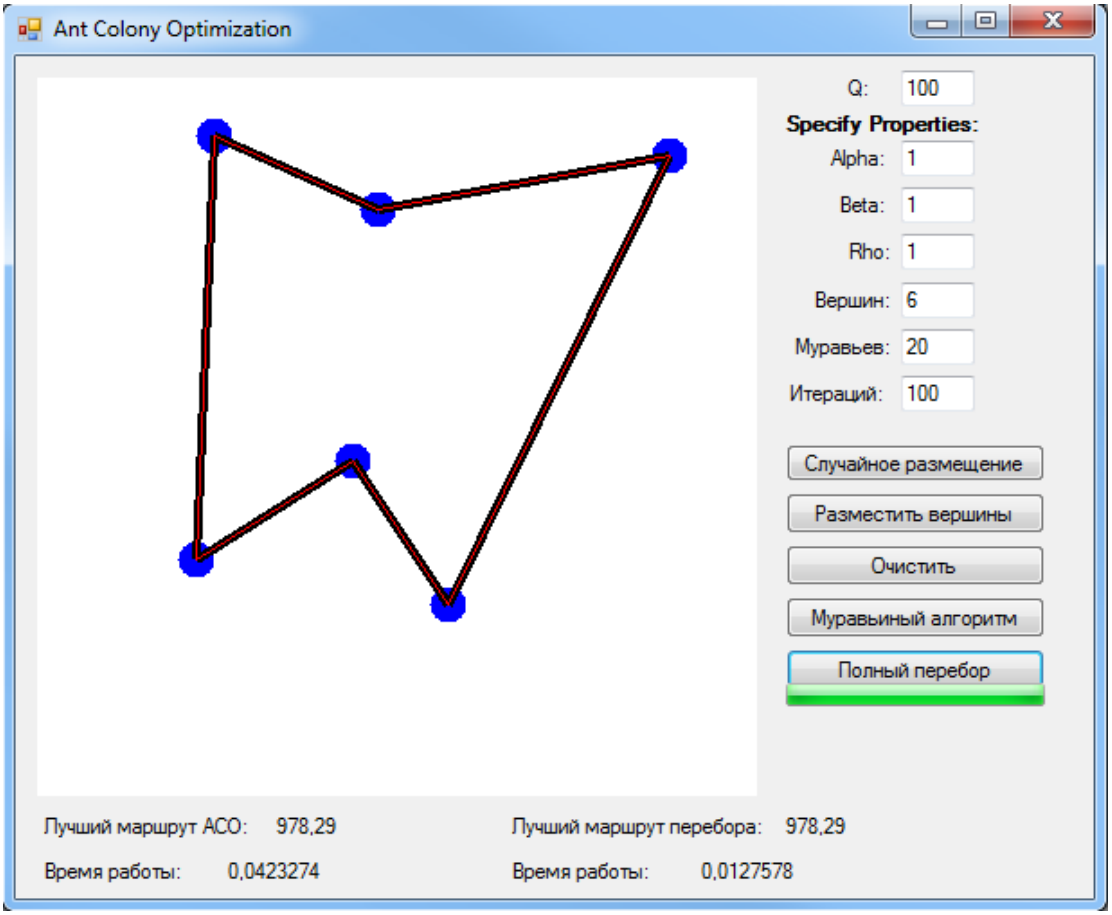


Рисунок 3.3 – Оптимальний шлях 2-го експерименту

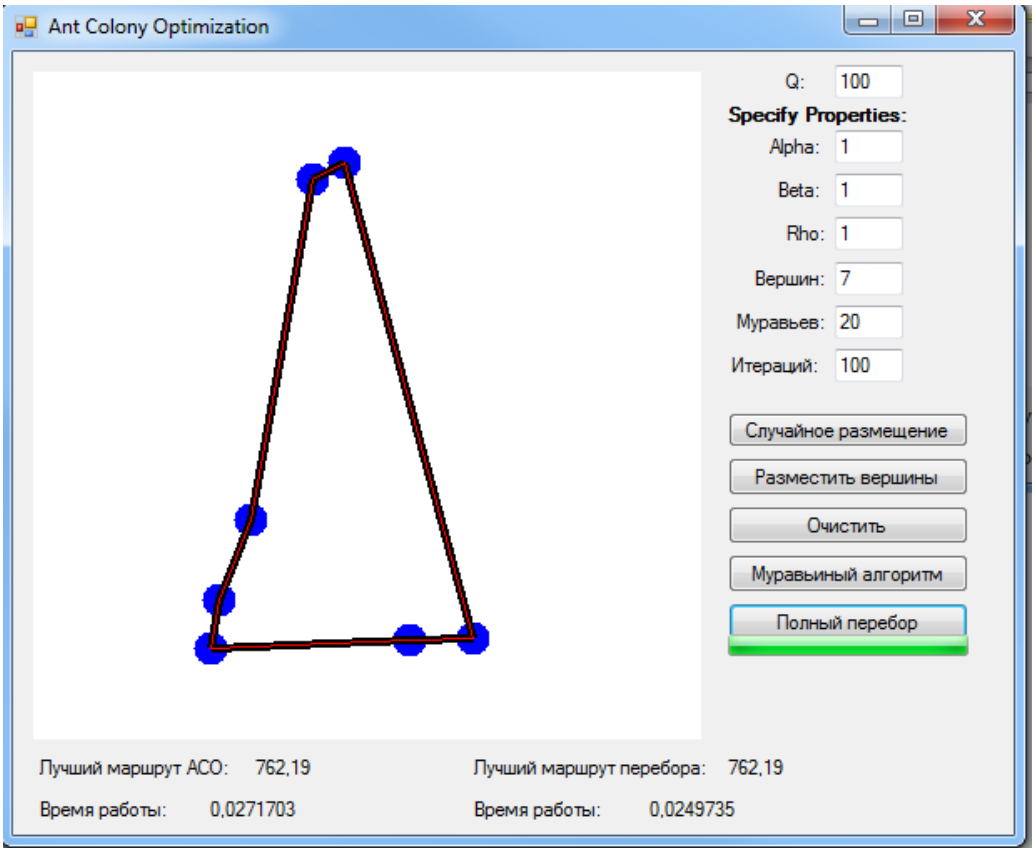


Рисунок 3.4 – Оптимальний шлях 3-го експерименту

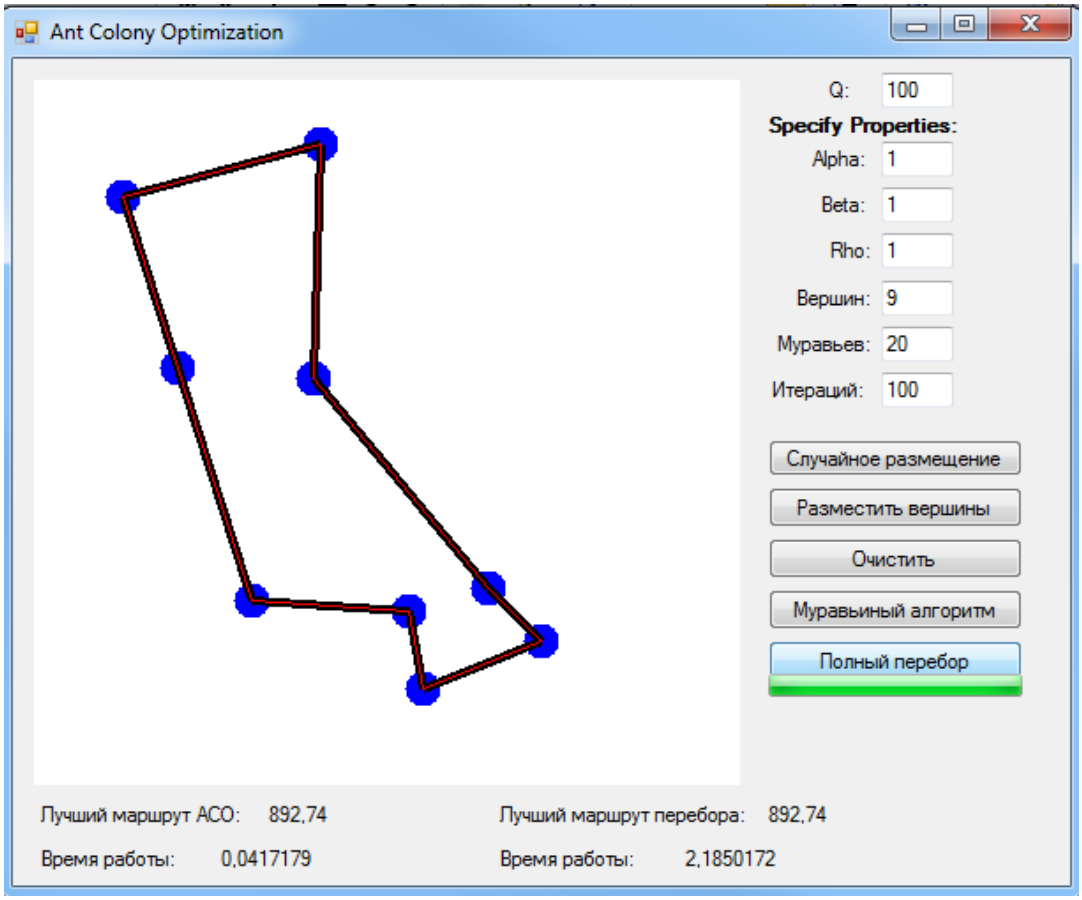


Рисунок 3.5 – Оптимальний шлях 4-го експерименту

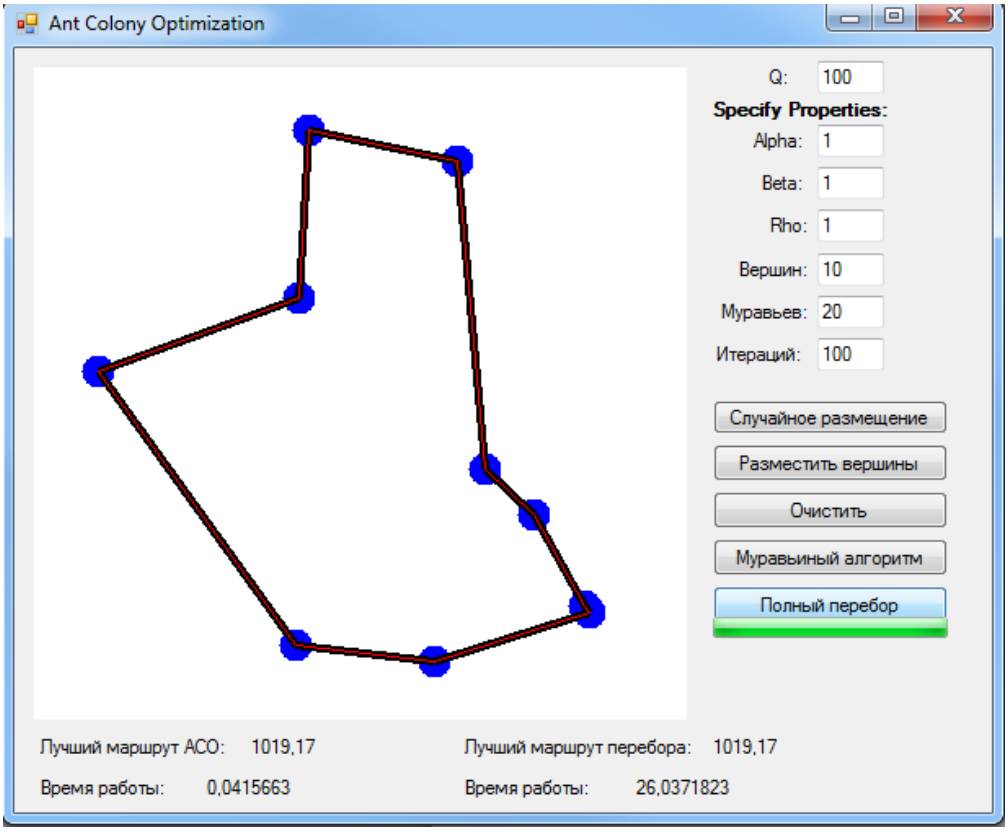


Рисунок 3.6 – Оптимальний шлях 5-го експерименту

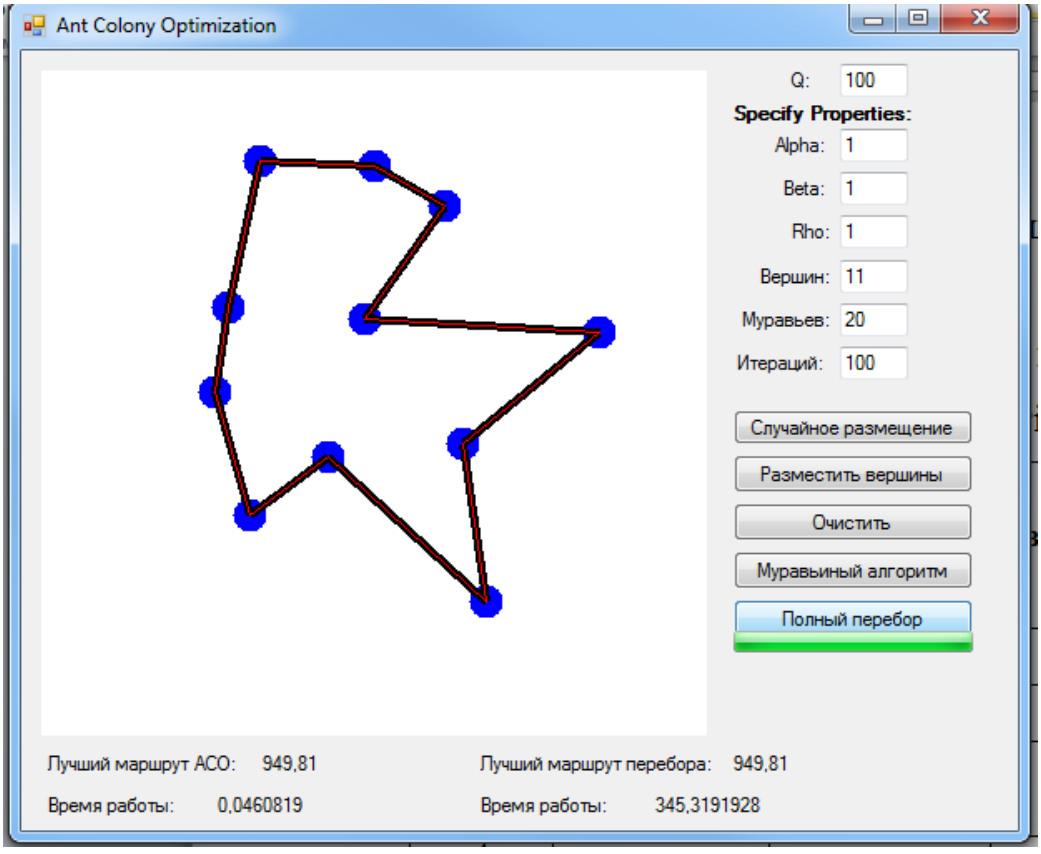


Рисунок 3.7 – Оптимальний шлях 6-го експерименту

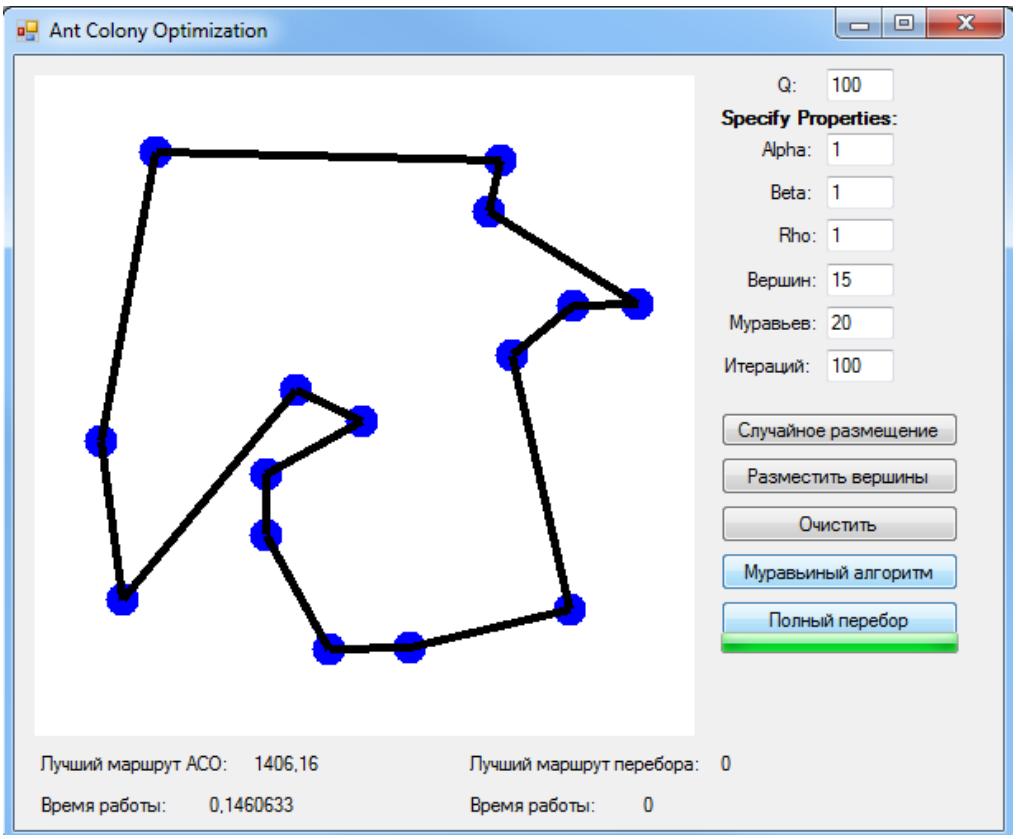


Рисунок 3.8 – Оптимальний шлях 7-го експерименту

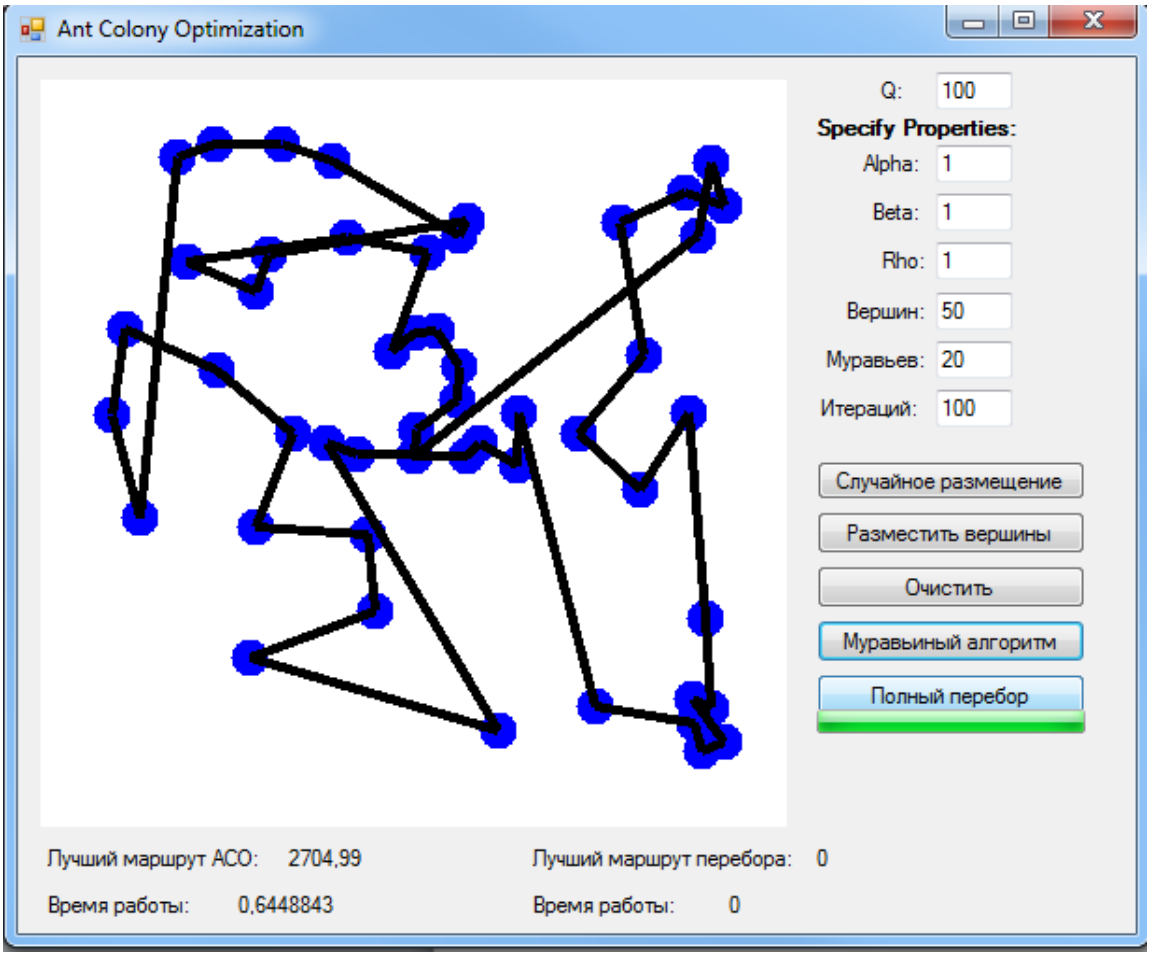


Рисунок 3.9 – Оптимальний шлях 8-го експерименту

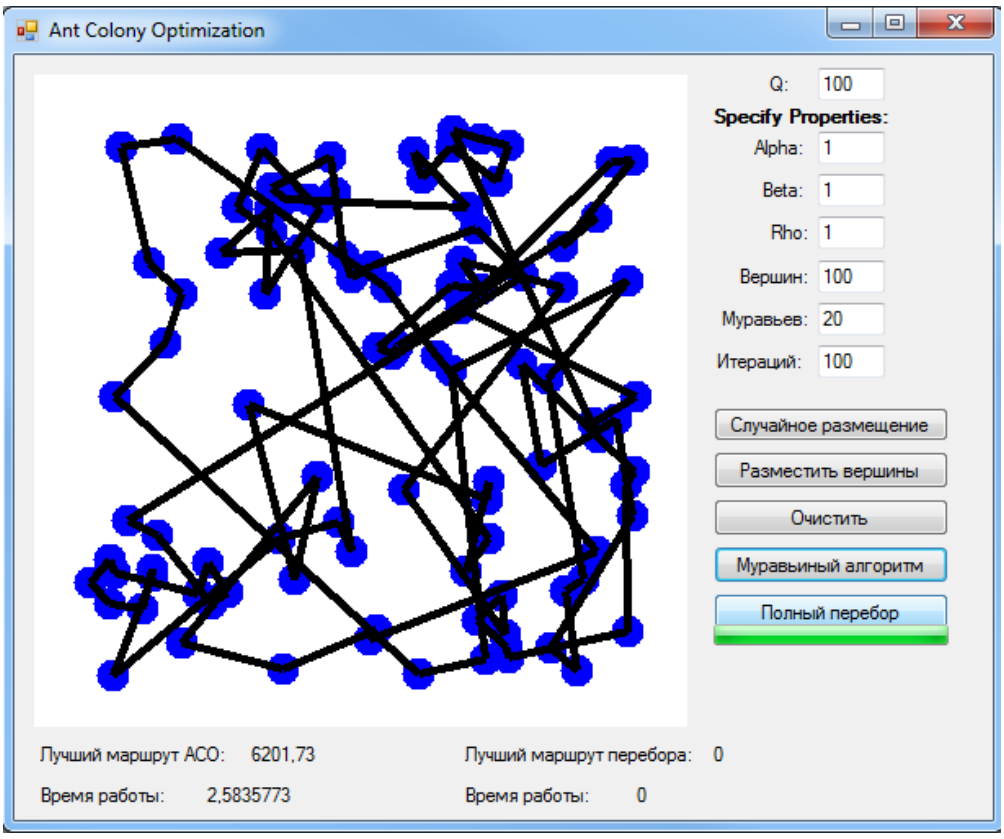


Рисунок 3.10 – Оптимальний шлях 9-го експерименту

У таблиці 3.3 приведено результати проведених експериментів.

Таблиця 3.3 – Порівняння результатів методів

№ експерименту	Кількість вершин	Найкращий маршрут ACO	Час виконання ACO	Найкращий маршрут повного перебору	Час виконання повного перебору
1	5	987,33	0,0423	987,33	0,00817
2	6	978,29	0,04232	978,29	0,01275
3	7	762,19	0,02717	762,19	0,024973
4	9	892,74	0,0417179	892,74	2,185017
5	10	1019,1	0,041566	1019,17	26,03718
6	11	949,81	0,04908	949,81	345,31919
7	15	1406,16	0,14606	--	--
8	50	2704,99	0,6448843	--	--
9	100	6201,73	2,5835	--	--

Після проведення 9-х експериментів доречно відмітити таке: на відносно невеликих графах алгоритм повного перебору забезпечує отримання точного розв’язку, витрачаючи при цьому прийнятну кількість часу. Проте при збільшенні кількості вершин витрати часу методу повного перебору збільшуються відповідно до теоретичної оцінки (графи розмірності 9, 10, 11 вершин). Використання повного перебору на графах великого розміру недоцільно.

3.5 Висновки за розділом 3

Розроблено модифікований мультиагентний метод для розв’язання задачі комівояжера, в якому у множину агентів додаються допоміжні «жадібні» мурахи, що підсилюють значущість ребер найкращого маршруту, знайденого на попередніх ітераціях роботи методу, що дозволяє скоротити час пошуку оптимального шляху.

Запропоновано основні рішення щодо реалізації програми. Описано основні розроблені класи, їх властивості та методи. Клас Ant представляє агента (мураху), який подорожує між вершинами. Агент володіє «списком табу», зберігає пройдену відстань, пройдений маршрут, а також знає про свою поточну позицію та наступне розташування (позицію на наступній ітерації). Клас City являє собою вершину графа з визначеними координатами о двовимірному просторі X та Y. По цих вершинах переміщуються мурахи. Запропоновано схему функціонування розробленої програми.

Виконано тестування розробленого програмного забезпечення. За результатами проведених експериментів відзначено, що на відносно невеликих графах алгоритм повного перебору дає точне рішення і витрачає малу кількість часу. Однак у міру збільшення кількості вершин часові витрати методу повного перебору змінюються відповідно до теоретичної оцінки (графи розмірності 9, 10, 11 вершин). Використання повного перебору на графах великого розміру недоцільно. В такому випадку більш ефективним є використання

запропонованого мультиагентного методу. Результати тестування підтвердили, що програма може використовуватися за призначенням.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення й умови застосування програми

Програмний продукт призначений для пошуку оптимального маршруту на основі мультиагентного підходу.

Для коректної роботи програму потрібно запускати в операційній системі Windows 7 або більш пізнішій версії операційної системи.

Для експлуатації програмного комплексу необхідні такі програмно-технічні засоби:

– ПК під керівництвом операційної системи та встановленим.Net Framework версії 4.5 та вище;

– тактова частота процесора 2 ГГц або більше;

– обсяг оперативної пам'яті не менше 4 Гб;

– обсяг вільного простору на диску – не менше 1 Гб;

– дозвіл екрана монітора не менше 1024x768;

– маніпулятор миша.

4.2 Характеристики програми

Програма була створена за допомогою мови програмування C#. Режим роботи програми ручний, тобто вона стартує за бажанням користувача. Працездатність розробленого ПЗ перевіряється описаним нижче способом :

– відкрити програму;

– знайти тестові вхідні дані для вирішення задачі комівояжера з відповіддю;

– запустити алгоритм з потрібними параметрами;

– порівняти результати.

4.3 Звернення до програми

Розроблене програмне забезпечення поширюється у вигляді інсталяційного пакета. Для можливості роботи з програмою необхідно розмістити файли модулів програмного забезпечення на комп'ютері. Перед використанням програми її необхідно встановити. Для цього необхідно запустити програму setup.exe, яка знаходиться на установчому диску. У відповідь на запит установчої програми потрібно вказати шлях встановлення програми. Після цього програма скопіює необхідні файли, створить і помістить в головне меню системи ярлик для запуску програми і зробить необхідні зміни в файлі реєстрації.

Після завершення процедури установки програму можна використовувати. Програму можна запустити будь-якими засобами операційної системи користувача.

Завантаження й запуск програми відбувається шляхом виконання користувачем .exe-файлу. Програма завантажується ОС до оперативної пам'яті, та починає своє виконання з функції main().

4.4 Вхідні і вихідні дані

Вхідними даними для програми є інформація про координати точок, які необхідно пройти комівояжеру, а також початкові параметри мультиагентного методу.

Вихідними даними для програми є результати розрахунків щодо пошуку оптимального шляху, зокрема: мінімальний шлях знайдений на кожній ітерації, візуалізація шляху, графіки залежності номеру ітерації, кількість ітерацій, що знадобилася для вирішення задачі, найоптимальніший шлях.

4.5 Повідомлення

Реалізована програмна система гарантує її стійке функціонування. У випадку виникнення помилок в програмі передбачено відображення стандартних діалогових вікон. Подібні повідомлення можуть виникати при деяких небажаних ситуаціях, до яких можна віднести введення користувачем некоректних даних.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Програмний продукт призначений для пошуку оптимального маршруту на основі мультиагентного підходу.

Для коректної роботи програму потрібно запускати в операційній системі Windows 7 або більш пізнішій версії операційної системи.

Інтерфейс користувача забезпечує просту і ефективну взаємодію користувача з програмним продуктом.

5.2 Умови виконання програми

Для експлуатації програмного комплексу необхідні такі програмно-технічні засоби:

– ПК під керівництвом операційної системи та встановленим.Net Framework версії 4.5 та вище;

– тактова частота процесора 2 ГГц або більше;

– обсяг оперативної пам'яті не менше 4 Гб;

– обсяг вільного простору на диску – не менше 1 Гб;

– дозвіл екрана монітора не менше 1024x768;

– маніпулятор миша.

Час роботи програми залежить від конфігурації комп'ютера, на якому вона виконується.

5.3 Виконання програми

Розроблене програмне забезпечення поширюється у вигляді інсталяційного пакета. Для можливості роботи з програмою необхідно розмістити файли модулів програмного забезпечення на комп’ютері. Перед використанням програми її необхідно встановити. Для цього необхідно

запустити програму setup.exe, яка знаходиться на установчому диску. У відповідь на запит установчої програми потрібно вказати шлях встановлення програми. Після цього програма скопіює необхідні файли, створить і помістить в головне меню системи ярлик для запуску програми і зробить необхідні зміни в файлі реєстрації.

Після завершення процедури установки програму можна використовувати. Програму можна запустити будь-якими засобами операційної системи користувача.

Завантаження й запуск програми відбувається шляхом виконання користувачем .exe-файлу. Програма завантажується ОС до оперативної пам’яті, та починає своє виконання з функції main().

5.4 Приклад роботи з програмою

Щоб змінити параметри або завантажити дані користувач повинен використовувати головне вікно програми (рисунок 5.1). Завдяки застосуванню компонентів Background Worker, обидва методи можуть виконуватися незалежно, внаслідок цього користувач має можливість запустити спочатку алгоритм повного перебору, потім паралельно запустити мурашиний алгоритм і подивитися кроки його виконання до завершення роботи повного перебору. Щоб змінити параметри потрібно використовувати текстові поля навпроти потрібного напису.

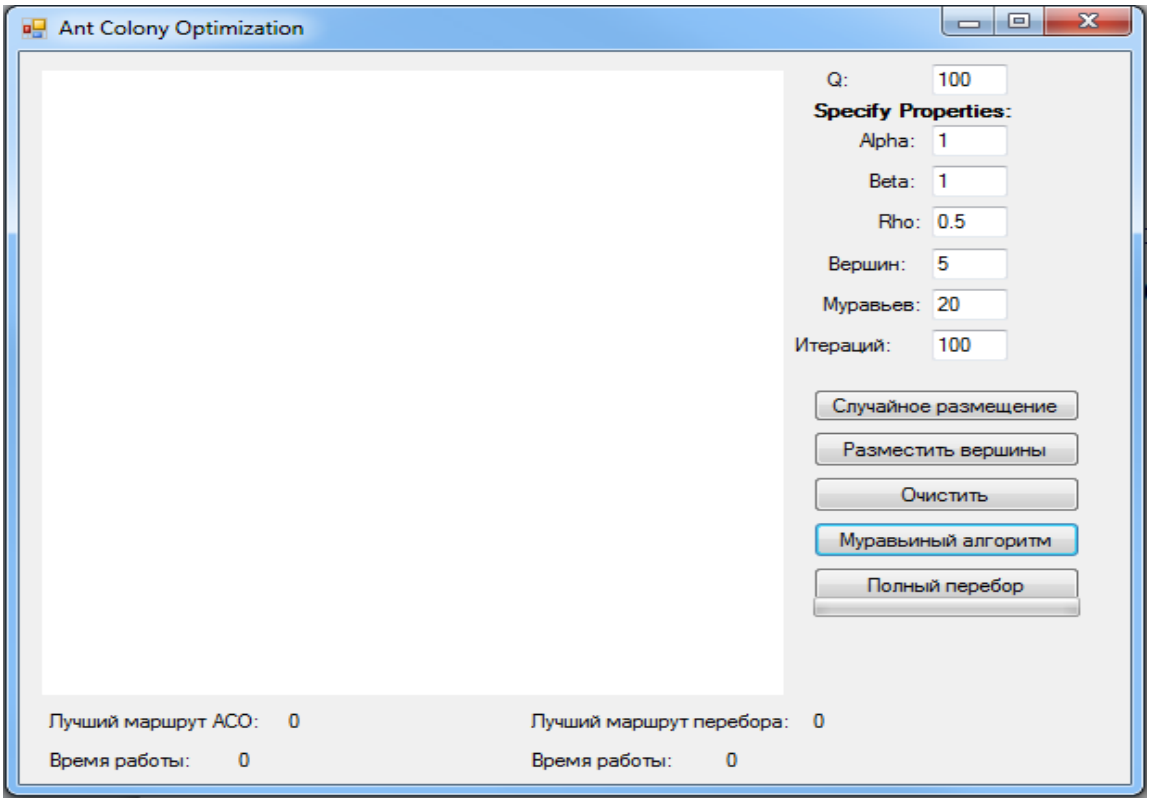


Рисунок 5.1 – Головне вікно програми

Після того як користувач обере кількість вершин, він зможе власноруч їх розставити за допомогою кнопки “Разместить вершины” або випадково за допомогою кнопки “Случайное размещение” (рисунок 5.2).

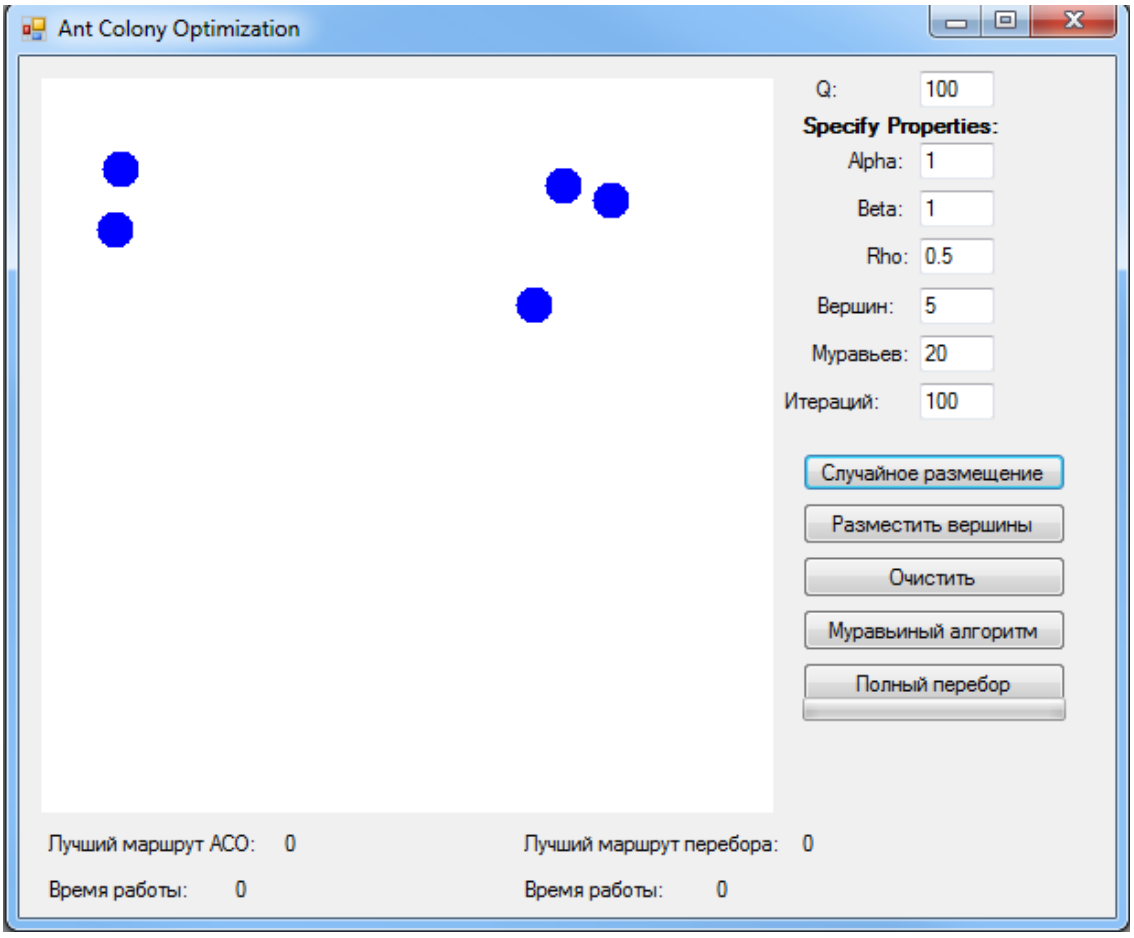


Рисунок 5.2 – Розміщення вершин

Користувачу буде надано візуалізацію роботи алгоритму (шляхи які перебираються). При закінченні роботи користувачу надається текстова інформація та візуалізація шляху (рисунок 5.3).

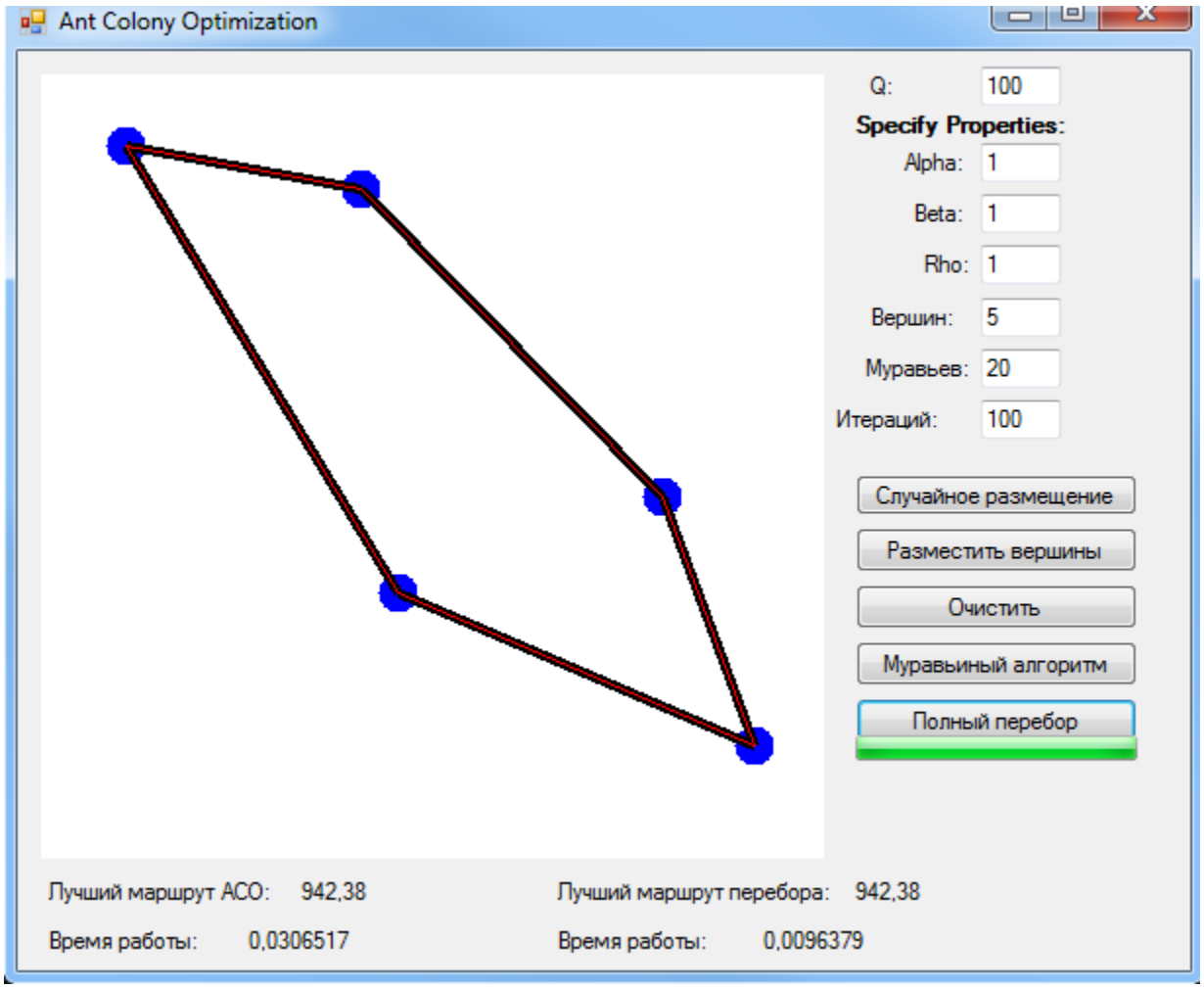


Рисунок 5.3 – Результуюча інформація

5.5 Повідомлення оператору

Як відзначено вище, у випадку виникнення помилок в програмі передбачено відображення стандартних діалогових вікон. Подібні повідомлення можуть виникати при деяких небажаних ситуаціях, до яких можна віднести введення користувачем некоректних даних.

Якщо користувач ввів кількість вершин 0. При натисканні на кнопки “Разместить вершины” або “Случайное размещение” він побачить інформаційне повідомлення (рис. 5.4).

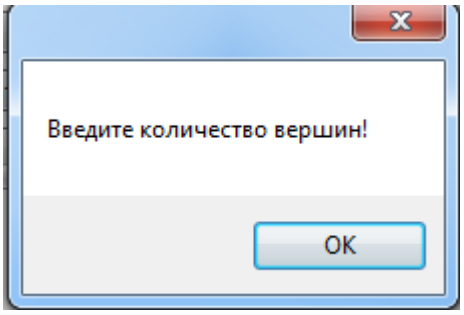


Рисунок 5.4 – Повідомлення про помилку

6 ЕКОНОМІКО-ОРГАНІЗАЦІЙНА ЧАСТИНА

У дипломній кваліфікаційній роботі магістра досліджено та реалізовано мультиагентний метод пошуку оптимального маршруту.

Очікується, що впровадження в експлуатацію розробленого програмного забезпечення на підприємстві дозволить скоротити одного співробітника, що відповідає за логістику, зокрема за складання оптимальних шляхів розвезення вантажів, з заробітною платою 11000 гривень на місяць.

У розробці беруть участь програміст із заробітною платою 13000 грн. в місяць і консультант з заробітною платою 9000 грн. у місяць. На розробку проєкту було витрачено 2 місяці. Розробка програми розпочалася першого вересня і повинна бути виконана до кінця жовтня 2020 року.

6.1 Планування розробки програмного продукту

Весь комплекс розробки програмного продукту можна розділити на етапи [24]. Для кожного етапу вказуються трудомісткість, кількість виконавців і тривалість робіт. Інформація з характеристики робіт по розробці програмного забезпечення зведена у таблицю 6.1.

Таблиця 6.1 – Характеристика робіт по розробці програми

Найменування робіт	Трудомісткість		Виконавці	Тривалість, днів
	люд-дн	%		
Аналіз ПЗ	4	6	Програміст Консультант	2 2
Визначення вимог до ПП	8	11,9	Програміст Консультант	4 4
Проектування структури програми	32	47,8	Програміст Консультант	16 16
Розробка схеми функціонування програми	3	4,5	Програміст	3
Створення програмного коду	9	13,4	Програміст	9
Тестування і налагодження програмного продукту	7	10,4	Програміст	7
Складання програмної документації	4	6	Програміст	4
Разом	67	100	Програміст Консультант	45 22

6.2 Визначення витрат на розробку програми

Для визначення витрат на розробку програми складають калькуляцію кошторисної вартості робіт, що включає статті:

– основна заробітна плата;

– додаткова заробітна плата;

– відрахування на єдиний соціальний внесок (ЄСВ);

– матеріали та комплектуючі вироби;

– витрати на спеціальне обладнання (на оплату машинного часу ЕОМ для написання і налагодження програмних засобів);

– накладні витрати.

Витрати за статтею «Основна заробітна плата» складаються з планового фонду зарплати всіх категорій працівників, зайнятих в розробці програми. Розрахунок зарплати на підставі даних про трудомісткість, представлений в таблиці 6.2.

Таблиця 6.2 – Основна заробітна плата

Посада виконавця	Чисельність, чол	місячний оклад	Кількість місяців роботи	Сума ЗП, грн
Програміст	1	13000	2	26000
Консультант	1	9000	1	9000
Разом	2			35000

Додаткову заробітну плату приймають рівною 10% від основної зарплати. Вона розраховується за формулою (6.1):

$$ЗП_{дод} = ЗР_{осн} \cdot 10\%. \quad (6.1)$$

Підставивши величину основної заробітної плати в дану формулу отримаємо: $ЗП_{дод} = 35000*0.1 = 3500$ грн.

Відрахування на єдиний соціальний внесок (ЄСВ)визначають у відсотковому відношенні від сум основної та додаткової зарплати з урахуванням премій і доплат. Відрахування вираховуються за формулою (6.2):

$$ОТ = (ЗП_{осн} + ЗП_{дод}) \cdot 2{,}22. \quad (6.2)$$

$$ОТ = (35000 + 3500) \cdot 0{,}22 = 8470 грн.$$

Використовується три найменування матеріалів: картридж – 700 грн, флеш-носій – 300 грн., папір – 200 грн. Витрати на матеріали розраховують за формулою (6.3):

$$M = \sum_{i=1}^{n} (Ц_i \cdot N_i \cdot (1 + K_{т.з.}) - Ц_{io} \cdot N_{io}, \quad (6.3)$$

де М – витрати на матеріали, покупні напівфабрикати і комплектуючі вироби, грн .;

$K_{т.з}$ – коефіцієнт, що враховує транспортно-заготівельні витрати;

$Ц_i$ – ціна і-го найменування матеріалу, напівфабрикату і комплектуючого, грн .;

N_i – потреба в і-му матеріалі, напівфабрикаті і комплектуючому;

$Ц_{io}$ – ціна зворотних відходів і-го найменування матеріалу, грн;

N_{io} – кількість зворотних відходів і-го найменування;

n – кількість найменувань матеріалів, напівфабрикатів і комплектуючих.

$Ц_{io}$ = 0; N_{io} = 0; $K_{т.з}$ = 0,05;

М = (1 + 0,05) · (700 + 300 + 200) = 1260 грн.

Таким чином, разом витрати на матеріали та комплектуючі вироби становлять 1260 гривень.

У статтю «Витрати на спеціальне обладнання» входять витрати на придбання, транспортування, монтаж і налагодження нестандартного обладнання. Практично, в даному випадку, в цій статті враховуються витрати на оплату машинного часу ЕОМ для написання і налагодження програмних засобів. Для цього необхідно скласти кошторис «витрат на утримання і експлуатацію устаткування», виходячи з якого визначиться вартість одного машино-години роботи ПК, після множення якої на машинний час пішло на написання і налагодження програми отримаємо витрати на оплату машинного часу.

Амортизаційні відрахування визначають за формулою (6.4):

$$A = БВ \cdot \frac{H_a}{100}, \quad (6.4)$$

де *БВ* – балансова вартість обчислювальної техніки, грн;

H_a – норма амортизаційних відрахувань на повне оновлення обчислювальної техніки,%. Приймаємо, що техніка повинна працювати 4 роки, тому H_a=25%.

Балансова вартість обчислювальної техніки становить 38000 грн. Тому амортизаційні відрахування становлять:

$$A = 38000 \cdot 0{,}25 = 9500 \text{ грн}$$

Статтю "Експлуатація обладнання" розраховують підсумовуванням витрат на силову електроенергію і допоміжні матеріали. Витрати на електроенергію визначають за формулою (6.5):

$$C_e = N_н \cdot Ф_{еф} \cdot K_{зв} \cdot K_{зм} \cdot Ц_e, \quad (6.5)$$

де $N_н$ – номінальна потужність ЕОМ, кВт;

$Ф_{еф}$ – річний ефективний фонд часу роботи ЕОМ, машино–рік;

$K_{зв}$ – середній коефіцієнт завантаження за часом;

$K_{зм}$ – коефіцієнт завантаження по потужності;

$Ц_e$ – ціна одного кВт ·рік електроенергії, грн ./ (кВт ·год).

Номінальна Потужність ЕОМ – 0,2 кВт. Річний ефективний фонд часу роботи ЕОМ ставити 1800 годин. Середні коефіцієнти завантаження за часом та за потужністю рівні відповідно 0,9 та 0,6. Ціна однієї кіловат–години електроенергії ставить 2,11 грн. Отримуємо:

$$Се = 0,2 \cdot 1800 \cdot 0,9 \cdot 0,6 \cdot 2,11 = 410,18 \text{ грн.}$$

Приймаємо, що Се = 410 грн.

Витрати за статтею "Заробітна плата обслуговуючих робітників та відрахування на соціальне страхування в інші фонди" визначають за формулою (6.6):

$$ЗП_{обсл} = ФЗП_{г} \cdot (1 + К_{відрах}) \cdot \frac{t_{обсл}}{Ф_{еф.обсл}}, \quad (6.6)$$

де $ФЗП_{г}$ – річний фонд заробітної плати (основної та додаткової) обслуговуючих робітників, грн.;

$К_{відрах}$ – коефіцієнт, що враховує відрахування на соціальне страхування і в інші фонди;

$t_{обсл}$ – час протягом року, необхідне технічне обслуговування ЕОМ, годин / рік. Приймаємо $t_{обсл}$ = 30 год.;

$Ф_{еф.обсл}$– річний ефективний фонд часу обслуговуючого персоналу, годин / рік. Приймаємо, що річний ефективний фонд часу обслуговуючого персоналу становить 1800 годин

Заробітна плата обслуговуючого персоналу = 9400 грн/місяць.

$$ФЗП_{г} = 9400 \cdot 12 = 112800 \text{ грн.}$$

$$ЗП_{обсл} = 112800 \cdot (1+0,22) \cdot 30/1800 = 2294 \text{ грн.}$$

Суму витрат за статтею "Поточний ремонт обладнання" обчислимо як 3% від балансової вартості обладнання, оскільки неможливо заздалегідь визначити, скільки коштів необхідно витрачати на придбання запасних частин наявного обладнання. Балансова вартість обчислювальної техніки становить

38000 грн. Тому сума витрат за статтею "Поточний ремонт обладнання" становить: 38000 0,03 = 1140 грн.

Інших витрат за проектом не передбачено, тому сума витрат за цією статтею дорівнює нуль гривень.

Кошторис витрат на утримання і експлуатацію устаткування наведений в таблиці 6.3.

Таблиця 6.3 – Кошторис витрат на утримання і експлуатацію устаткування

Найменування статей витрат	Сума, грн.
Амортизація обладнання	9500
Заробітна плата основна і додаткова обслуговуючих робітників з відрахуваннями на соціальні заходи	2294
Експлуатація обладнання (крім витрат на поточний ремонт)	410
Поточний ремонт обладнання	1140
Інші витрати	0
Разом	13344

Витрати на оплату машинного часу ЕОМ для написання і налагодження програмних засобів визначаються за формулою (6.7):

$$C_{мо} = P_{екс} \cdot t_{мо}, \tag{6.7}$$

де $C_{мо}$ – витрати на оплату машинного часу, грн;

$P_{екс}$ – експлуатаційні витрати на одну годину машинного часу цієї цифрової ЕОМ, грн. / машино–год.;

$t_{мо}$– машинний час цифрової ЕОМ для написання і налагодження даного програмного продукту, машино–год.

$$P_{екс} = 13344/1800 = 7{,}41 \text{ грн. / машино–год.}$$

Для написання і налагодження даного програмного продукту ЕОМ експлуатується протягом 45 днів в одну зміну по 8 годин, що складає в сумі 360 годин. Таким чином отримуємо суму за статтею «Витрати на спеціальне обладнання»:

$$С_{мо} = 7{,}41 \cdot 360 = 2668 \text{ грн.}$$

До накладних витрат відносяться витрати на загальне управління і загальногосподарські потреби (заробітна плата апарату управління, канцелярські витрати і т.д.), утримання та експлуатацію будівель. Накладні витрати включаються до вартості розробки програми непрямим шляхом – у відсотках до основної заробітної плати розробників. Для обчислення цієї статті видатків у дипломному проєкті приймемо, що накладні видатки становлять 40% від основної заробітної плати розробників, що складає 35000грн. Тому сума накладних видатків становить: 35000*0,40=14000 грн.

Калькуляція кошторисної вартості робіт з розробки програми наведена в таблиці 6.4.

Таблиця 6.4 – Калькуляція кошторисної вартості робіт з розробки програми

Найменування статей витрат	Сума, грн.
Основна заробітна плата	35000
Додаткова заробітна плата	3500
ЄСВ	8470
Матеріали і комплектуючі	1260
Витрати на спеціальне обладнання	2668
Накладні витрати	14000

Разом	64898

6.3 Розрахунок економічної ефективності програмного продукту

Річну економію від впровадження програми, що дозволяє знизити витрати машинного часу ЕОМ для рішення певного завдання в порівнянні із програмою, що застосовувалася раніше, визначають за формулою:

$$E = (T_{м1} - T_{м2}) \cdot B_{екс.ктз} + E_{н}^{a} \cdot \frac{Ф_{б.ктз} \cdot (T_{м1} - T_{м2})}{Ф_{еф.ктз}} - \frac{S_{рп}}{T_{с}} + E_{зп}, \qquad (6.8)$$

де $T_{м_1}, T_{м_2}$ – машинний час, необхідний для розв'язання поставлених завдань, відповідно в старому та у новому варіантах, машино–год./рік;

$B_{екс}$ – експлуатаційні витрати, що доводяться на 1 год машинного часу ЕОМ, грн/ машино–год.;

$E_{н}^{а}$ – нормативний коефіцієнт ефективності капітальних вкладень у засоби автоматизації;

$Ф_{б.ктз}$ – балансова вартість ЕОМ (КТЗ), грн.;

$Ф_{еф.ктз}$ – річний ефективний фонд часу роботи ЕОМ, машино–год.;

$S_{рп}$ – сумарні витрати на розробку програми, грн.;

$T_{с}$ – термін служби впроваджуваної програми до її морального зношування, років. Приймаємо цей термін 5 років;

$E_{зп}$ – економія на заробітній платі.

До впровадження даного ПЗ дані функції виконувались співробітником, що відповідає за обробку рукописних символів, 1800 годин на рік, а після 0 годин на рік (економія складає 1,0 ставки одного співробітника, що відповідає

за логістику, зокрема за складання оптимальних шляхів розвезення вантажів, з заробітною платою 11000 гривень на місяць).

$$E = (1800 - 0) \cdot 7{,}41 + 0{,}2 \cdot (38000 \cdot (1800 - 0)) / 1800 - 64898/5 +$$
$$+11000 \cdot 1{,}0(1 + 0{,}22) \cdot 12 = 168998 \text{ грн.}$$

Коефіцієнт ефективності капітальних вкладень E и строк їхньої окупності $T_{ок}$ розраховуються за формулами (6.9) і (6.10):

$$E_{ф} = \frac{E}{S_{рп}}, \tag{6.9}$$

$$T_{ок} = \frac{S_{рп}}{E}, \tag{6.10}$$

$$E_{ф} = 168998 / 64898 = 2{,}6,$$

$$T_{ок} = 64898 / 168998 = 0{,}38 \text{ року.}$$

Зведемо отримані техніко–економічні показники в таблицю 6.5.

Таблиця 6.5 – Техніко–економічні показники

Показники	Сума	Одиниця виміру
Витрати на розробку програми	64898	грн.
Річна економія	168998	грн.
Строк окупності програми	0,38	Рік

Таким чином, впровадження в експлуатацію розробленого програмного забезпечення на виробничому підприємстві дозволить скоротити одного співробітника, що відповідає за логістику, зокрема за складання оптимальних шляхів розвезення вантажів, з заробітною платою 11000 гривень на місяць. Економічні розрахунки показали, що строк окупності програми 0,38 року, що значно менше часу її нормативного строку окупності (5 років). Крім того, річна економія при використанні програми становить 168998 грн. Виходячи з

виконаних розрахунків, можна зробити висновок, що розробка даного програмного продукту є економічно доцільною.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Аналіз потенційних небезпек

В дипломному проекті проводиться дослідження та програмна реалізація мультиагентних методів пошуку оптимального маршруту.

При роботі на фахівця постійно або періодично діють наступні небезпечні і шкідливі чинники.

– підвищені статичні навантаження користувачів ПК, пов’язані з повторенням однотипних рухів кистями рук може призвести до защемлення нерву у зап’ястному каналі (тунельний синдром);

– невідповідність ергономічних характеристик устаткування нормованим величинам внаслідок нераціональної організації робочого місця;

– можливість ураження електрострумом внаслідок небезпечного рівня напруги в електричній мережі, тому у приміщені можуть виникнути аварійні ситуації: замикання, загоряння проводки, ураження електрострумом, що може призвести до електротравм або летальних наслідків;

– невідповідність нормам параметрів мікроклімату;

– незадовільні умови освітлення робочого місця, в наслідок не рівномірного розподілу освітлювальних приладів;

– небезпеки, пов’язані з підвищеним рівнем вібрації;

– можливість загорянь внаслідок порушень правил пожежної безпеки;

– непоінформованість персоналу у відповідній послідовності дій в умовах надзвичайної ситуації, що призводить до травмування та до інших непередбачених негативних наслідків.

7.2 Заходи по забезпеченню безпеки

Забезпечення безпеки праці, організації оптимальних умов праці, а також захисту навколишнього середовища досягається шляхом дотримання норм і проведення заходів, передбачених законодавчими актами України.

Для запобігання підвищеним статичним навантаженням, зокрема, «тунельному синдрому», користувачам ПК рекомендовано:

– для запобігання «тунельного синдрому» робоче місце має бути зручним і ергономічним;

– працюючи за клавіатурою, необхідно стежити, щоб кут згину руки в лікті був прямим, тобто 90°;

– дотримуватися оптимальної висоти клавіатури від підлоги – 65-75 см;

– користуватися ергономічними і зручними особисто для вас миші і клавіатури;

– використовувати силіконові подушки в якості опори для кисті [25].

Згідно ПУЕ [26] приміщення виробництва відноситься до категорії «Приміщення без підвищеної небезпеки». При виконанні правил ПУЕ ураження електричним струмом можливо тільки у випадку несправності апаратури й живильних кабелів.

Апаратура, що перебуває під небезпечною для життя напругою, припускає забезпечення електробезпечності. Всі металеві частини заземлені на третій провід спеціальних триконтактних розеток. Струмопровідні кабелі розташовані під дерев'яною підлогою, зверху покритою лінолеумом, або у вигляді схованої проводки в стінах, таким чином, контакт із кабелем виключається. Над кожною розеткою попереджуючий надпис.

Відповідно до ПУЕ в електроустановках до 1000 В мінімальне значення опору ізоляції силових і освітлювальних електропроводок на ділянці між суміжними запобіжниками або за останніми запобіжниками між будь-яким проведенням і землею становить не менш 0,5МОм.

Відповідно до ПУЕ захисне заземлення застосовується у всіх електроустановках змінного струму напругою 380В и вище й постійного струму 440В і вище. У приміщеннях з підвищеною небезпекою заземлення використовується в електроустановках змінного струму напругою 42 В та постійного струму 110 В [26].

Захисному заземленню підлягають корпуси трансформаторів, оболонки кабелів, металеві огородження електроустановок відповідно до ДСТУ 7237:2011 «Система стандартів безпеки праці. Електробезпека. Загальні вимоги та номенклатура видів захисту». Електрична ізоляція забезпечується конструкціями пристроїв і способами їхнього підключення. У процесі експлуатації періодично проводиться перевірка якості ізоляції, шляхом виміру її опору [27].

Вирівнювання потенціалів досягаються шляхом устрою контурних заземлювачів. У приміщеннях вирівнювання потенціалів здійснюється з'єднанням металевих стойок ЕОМ, корпусів апаратів світильників і іншого устаткування з усіма доступними для дотику металевими конструкціями будинку, спорудження, а також з контуром шиною захисного заземлення.

Електричний поділ мережі припускає розділ мережі на окремі електрично непов'язані між собою ділянки за допомогою розділових трансформаторів. Підвищення електробезпечності відбувається за рахунок зменшення ємкості мережі й підвищення опору ізоляції фаз мережі щодо землі.

Подвійна ізоляція забезпечує застосування крім основної, робочої ізоляції струмоведучих частин ще одного шару додаткової ізоляції, що ізолює людину від металевих неструмоведучих частин, які можуть випадково виявитися під напругою. При застосуванні електроустаткування з подвійною ізоляцією не потрібно ні заземлення, ні занулення їхніх корпусів [27].

Згідно з завданням, вихідні дані для розрахунку захисного заземлення приведені нижче.

1. U = 380 В,

Розмір приміщення L = 15 м, В = 9 м,

Параметри вертикально стрижня: $l_в$ = 3м, d = 0,014м,

Ширина смуги b_c = 0,04м,

$\rho_{вим}$ = 80 Ом·м,

Кліматична зона II,

Склад ґрунту – однорідний,

Вологість повітря – нормальна.

2. Визначаємо необхідний опір штучного заземлювача $R_{\text{ш}}$, Ом, якщо передбачається використання також природного заземлювача $R_{\text{ш}}$, за формулою:

$$R_{\text{ш}} = \frac{R_{\text{е}} R_{\text{з}}}{R_{\text{е}} - R_{\text{з}}}, \text{ Ом;}$$

де $R_{\text{е}}$ – опір розтіканню струму природних заземлювачів, Ом;

$R_{\text{з}}$ – розрахунковий нормований опір заземлювального пристрою (ЗП), Ом (додаток А); $R_{\text{з}}$=4 Ом

При відсутності природних заземлювачів необхідний опір штучного заземлювача дорівнює розрахованому нормованому опору ЗП: $R_и = R_з$.

$$R_и = 4 \text{ Ом}$$

3. Визначимо розрахунковий питомий опір ґрунту за формулою:

$$\rho = \rho_{вим} \cdot \Psi, \text{ Ом·м;}$$

де ρ – розрахунковий питомий опір ґрунту, Ом·м;

$\rho_{вим}$ – питомий опір землі, отриманий у результаті вимірів, Ом·м; $\rho_{вим}$ = 80 Ом·м

Ψ – коефіцієнт сезонності, що враховує промерзання чи висихання ґрунту; Ψ=1,5

$$\rho = \rho_{вим} \cdot \Psi = 80*1,5 = 120 \text{ Ом·м;}$$

4. Обчислимо опір розтіканню струму одиночного вертикального заземлювача $R_в$, Ом. Для стрижневого заземлювача круглого перерізу, заглибленого в ґрунт розрахункова формула має вигляд:

$$R_в = \frac{\rho}{2 \cdot \pi \cdot l_в}\left(\ln\frac{2 \cdot l_в}{d} + \frac{1}{2} \cdot \ln\frac{4 \cdot t_в + l_в}{5 \cdot t_в - l_в}\right), \text{ Ом;}$$

де ρ – розрахунковий питомий опір ґрунту, Ом·м; ρ = 120 Ом·м;

$l_в$ – довжина вертикального стрижня, м; $l_в$ = 3 м;

d – діаметр перерізу стрижня, м; d = 0,014 м

$t_в$ – відстань від поверхні ґрунту до середини довжини вертикального стрижня, яка обчислюється за формулою:

$$t_в = 0{,}8 + \frac{1}{2} l_в, \text{ м.}$$

$$t_в = 0{,}8 + \frac{1}{2} l_в = 0{,}8+1/2*3= 0{,}8+1{,}5 = 2{,}3 \text{ м}$$

Тоді $R_в$ = 39,73 Ом.

5. Розрахуємо наближену (мінімальну) кількість вертикальних стрижнів

$$n' = \frac{R_в}{R_и}, \text{ шт;}$$

де $R_в$ – опір розтікання струму одиночного вертикального заземлювача, Ом; $R_в$ = 39,73 Ом

$R_и$ – необхідний опір штучного заземлювача, Ом; $R_и$ = 4 Ом.

$$n' = \frac{R_в}{R_и} = \frac{39{,}73}{4} = 9{,}93 \text{ шт.}$$

Якщо виходити з розмірів контуру:

$$n = \frac{P}{a}, \text{ шт;}$$

де P – периметр прямокутника (приміщення), м. Який визначається за формулою:

$$P = 2 \cdot (L + B), \text{ м;}$$

L – довжина приміщення, м;

B – ширина приміщення, м;

$$P = 2*(15+9) = 48 \text{ м}$$

a – відстань між стрижнями, обумовлена зі співвідношення, м:

$$a = k \cdot l_в;$$

де $l_в$ – довжина вертикального стрижня, м;

k – коефіцієнт кратності, який дорівнює 1, 2, 3 (для заглиблених стаціонарних заземлювачів $k=1$).

$$a = k \cdot l_в = 1*3= 3 \text{ м}$$

Тоді $n = \frac{P}{a} = \frac{48}{3} = 16$, приймаємо n = 16 шт.

Отриману кількість стрижнів округляють до більшого довідкового значення, але не менше n'.

6. Визначаємо довжину горизонтальної смуги при конфігурації групового заземлювача – контур:

$$l_г = 1{,}05 \cdot a \cdot n, \text{ м}$$

де n – кількість вертикальних стрижнів; n = 16 шт.

a – відстань між вертикальними стрижнями, м; a=3 м

$$l_г = 1{,}05 \cdot a \cdot n = 1{,}05*16*3 = 50{,}4 \text{ м}.$$

7. Обчислити опір розтіканню струму горизонтальної з'єднуючої смуги $R_г$, Ом. У випадку горизонтального смугового заземлювача розрахунок виконується за формулою:

$$R_г = \frac{\rho}{2 \cdot \pi \cdot l_г} \cdot \ln \frac{2 \cdot l_г^{\,2}}{b_c \cdot t_г}, \text{ Ом;}$$

де ρ – розрахунковий питомий опір ґрунту, Ом·м; ρ = 120 Ом·м;

$l_г$ – довжина горизонтальної смуги, м; $l_г$ = 50,4 м

b_c – ширина смуги, м; b_c = 0,04 м

$t_г$ – відстань від поверхні ґрунту до середини ширини горизонтальної смуги, яка обчислюється за формулою:

$$t_г = 0{,}8 + \frac{1}{2} b_c, \text{ м.}$$

$$t_г = 0{,}8 + \frac{1}{2} * 0{,}04 = 0{,}82 \text{ м}$$

Тоді:

$$R_г = \frac{\rho}{2 \cdot \pi \cdot l_г} \cdot \ln \frac{2 \cdot l_г^{\,2}}{b_c \cdot t_г} = 4{,}53 \text{ Ом}$$

8 Розрахувати еквівалентний опір розтіканню струму групового заземлювача.

$$R_{гр} = \frac{R_в \cdot R_г}{R_в \cdot \eta_г + R_г \cdot \eta_в \cdot n}, \text{ Ом}$$

де $R_в$ – опір розтікання струму одиночного вертикального заземлювача, Ом; $R_в$ = 39,73 Ом

$R_г$ – опір розтікання струму горизонтальної смуги, Ом; $R_г$ = 4,53 Ом

$\eta_в, \eta_г$ – коефіцієнти використання вертикальних стрижнів і горизонтальної смуги, Ом (додаток Б, В); $\eta_в$ = 0,51; $\eta_г$ = 0,298

n – кількість вертикальних стрижнів; n = 16 шт.

$$R_{гр} = \frac{R_в \cdot R_г}{R_в \cdot \eta_г + R_г \cdot \eta_в \cdot n} = 3{,}69 \text{ Ом}$$

9. Отриманий опір розтіканню струму групового заземлювача не повинен перевищувати необхідний опір, визначений у пункті 2.

$$R_{гр} \le R_u$$

$$3{,}69 \text{ Ом} \le 4{,}00 \text{ Ом}$$

Умова виконана, рішення завдання вірне.

7.3 Заходи з виробничої санітарії та гігієни праці

У відповідності до НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» передбачені гігієнічні вимоги до організації устаткування робочих місць ВДТ ЕОМ і ПЭВМ [28]. Конструкція робочого стола відповідає сучасним вимогам ергономіки й забезпечує оптимальне розміщення на робочій поверхні використовуваного устаткування (дисплея, клавіатури, принтера) і документів.

Висота робочої поверхні робочого стола із ВДТ регулюється у межах 680...800 мм, а ширина й глибина – забезпечує можливість виконання операцій

у зоні досяжності моторного поля (рекомендовані розміри: 600...1400 мм, глибина – 800..1000 мм). Приймаємо робочу поверхню з розмірами 1400х800. Щоб уникнути відблисків на екрані вікон розмісщаємо робоче місце торцем до вікна не ближче чим на 1 м. Площа дозволяє нам розмістити додаткові меблі для зберігання документів, магнітних дисків, полками, стелажами, тумбами й т.п.

Під час організації робочого простору необхідно враховувати індивідуальні антропометричні параметри користувача з відповідними допусками на можливі зміни робочих поз та потребу у переміщеннях.

7.4 Заходи безпеки у надзвичайних ситуаціях

7.4.1 Заходи з пожежної безпеки

В приміщенні обчислювального центру джерелом загоряння можуть бути електронні схеми ЕОМ, прилади для технічного обслуговування, електроустаткування, кондиціонери повітря, де за рахунок різних порушень відбувається перегрів тих чи інших елементів, електричні іскри, дуга, що можуть визвати загоряння горючих матеріалів, дерев’яні та пластикові стільці та столи, папір, перевантаження мережі живлення, людські чинники тощо.

Заходи щодо протипожежного захисту розділяються на організаційні, експлуатаційні, технічні й режимні (спеціальні).

7.4.2 Заходи з цивільного захисту

Порядок дій сил цивільного захисту у вогнищі хімічного ураження. Вогнища хімічного ураження (ВХУ) виникають у результаті застосування супротивником ОР, а також при руйнуванні хімічно небезпечних об'єктів, які виготовляють чи використовують у технології СДОР (такі вогнища прийнято називати вторинними). При надходженні перших даних про виникнення ВХУ начальники і штаби ЦО оповіщають населення в зоні зараження і у районах,

яким загрожує небезпека від хмари зараженого повітря, яка поширюється, ставлять завдання хімічній і медичній розвідкам, віддають розпорядження про проведення заходів щодо захисту населення, приймають рішення й організовують проведення РІНР у вогнищах. До РІНР залучаються окремі батальйони спеціального захисту, підрозділи хімічного захисту військових частин ЦО, зведені загони (команди) протирадіаційного і протихімічного захисту, медичні формування, команди знезаражування й інші спеціально підготовлені й оснащені підрозділи і формування. собовий склад сил ЦО, що вводиться у ВХУ, забезпечується ЗІЗ органів дихання і шкіри, антидотами, індивідуальними протихімічними пакетами.

При проведенні рятувальних робіт у вторинному вогнищі ураження основні зусилля спрямовуються на локалізацію джерел отруйної речовини і запобігання її подальшого надходження на місцевість і в повітря.

Роботи проводяться з максимальною інтенсивністю до повного їх завершення в найкоротший термін, із залученням необхідної кількості сил і засобів та з дотриманням заходів безпеки [36].

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи магістра було розв’язано актуальне завдання дослідження та програмної реалізації мультиагентних методів пошуку оптимального маршруту. Проаналізовано задачу комівояжера. Досліджено методи розв’язання задачі комівояжера. Визначено, що для розв’язання задачі комівояжера ефективно можуть застосовуватися мультиагентні методи. Досліджено мультиагентний метод мурашиних колоній та його модифікації. Відзначено, що метод мурашиних колоній показав себе як дуже гнучкий інструмент, який може мати різні складові частини. Він знайшов широке використання, проте як було визначено у цьому розділі його суттєвим недоліком є значний час роботи, тому актуальною є розробка мультиагентного методу, вільного від зазначених недоліків.

Досліджено та проаналізовано вимоги до програмного забезпечення пошуку оптимального маршруту. Проаналізовано інструментарії розробки програмного забезпечення. Порівнявши відомі середовища розробки програмних засобів, оцінивши їх переваги та недоліки, було прийнято рішення про використання для розробки програмного забезпечення пошуку оптимального маршруту мови C#, оскільки ця мова є досить зручною для використання, дозволяє створювати програмні продукти різного призначення, та має багато бібліотек, що спрощують реалізацію математичних обчислень при створенні програми пошуку оптимального шляху.

Розроблено модель програмного забезпечення пошуку оптимального шляху, що представляє собою набір діаграм у нотації UML, які дозволяють розробнику створювати програмне забезпечення відповідно до вимог замовника.

Наукова новизна роботи полягає у тому, що розроблено модифікований мультиагентний метод для розв’язання задачі комівояжера, в якому у множину агентів додаються допоміжні «жадібні» мурахи, що підсилюють значущість

ребер найкращого маршруту, знайденого на попередніх ітераціях роботи методу, що дозволяє скоротити час пошуку оптимального шляху.

Практичне значення роботи полягає у тому, що розроблено програмне забезпечення, що реалізує мультиагентний метод пошуку оптимального маршруту.

Виконано тестування розробленого програмного забезпечення. Результати тестування підтвердили, що програма може використовуватися за призначенням.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Олійник А. О. Еволюційні обчислення та програмування : навчальний посібник / А. О. Олійник, С. О. Субботін, О. О. Олійник. – Запоріжжя : ЗНТУ, 2010. – 324 с.

2. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: Навчальний посібник. – Запоріжжя: ЗНТУ, 2008. – 341 с.

3. Дубровін В. І. Методи оптимізації та їх застосування в задачах навчання нейронних мереж : навчальний посібник / В. І. Дубровін, О. Субботін. – Запоріжжя: ЗНТУ, 2003. – 136 с.

4. Haupt, R. Practical genetic algorithms / Randy L. Haupt, Sue Ellen Haupt.—2nd ed., 2004. – 261 p.

5. Stroustrup, B. The C++ programming language / Bjarne Stroustrup.— Fourth edition., 2013. – 1366 p.

6. Шлее, М. Qt 5.5 Профессиональное программирование на C++/ М. Шлее// СПб.: «БХВ-Петербург», 2015. — 896 с.

7. Encyclopedia of artificial intelligence / Eds.: J. R. Dopico, J. D. de la Calle, A. P. Sierra. – New York : Information Science Reference, 2009. – Vol. 1-3. – 1677 p.

8. Gen M. Genetic algorithms and engineering design / M. Gen, R. Cheng. – New Jersey : John Wiley & Sons, 1997. – 352 p.

9. Емельянов В. В. Теория и практика эволюционного моделирования / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик. – М. : Физматлит, 2003. – 432 с.

10. Курейчик В. М. Генетические алгоритмы: монография / В. М. Курейчик. – Таганрог : ТРТУ, 1998. – 242 с.

11. Прогрессивные технологии моделирования, оптимизации и интеллектуальной автоматизации этапов жизненного цикла авиадвигателей : Монография / А. В. Богуслаев, Ал. А. Олейник, Ан. А. Олейник, Д. В.

Павленко, С. А. Субботин ; под ред. Д. В. Павленко, С. А. Субботина. – Запорожье : ОАО «Мотор Сич», 2008. – 468 с.

12. Рассел С. Искусственный интеллект: современный подход / С. Рассел, П. Норвиг. – М.: Вильямс, 2006. – 1408 с.

13. Ротштейн А. П. Интеллектуальные технологии идентификации: нечеткие множества, генетические алгоритмы, нейронные сети / А. П. Ротштейн. – Винница: Універсум-Вінниця, 1999. – 320 с.

14. Руденко О. Г. Штучні нейронні мережі / О. Г. Руденко, Є. В. Бодянський. – Х.: Компанія СМІТ, 2006. – 404 с.

15. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы / Д. Рутковская, М. Пилиньский, Л.Рутковский ; пер. с польск. И. Д. Рудинского. – М.: Горячая линия – Телеком, 2004. – 452 с.

16. Goldberg, D. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989. – 432 p.

17. Holland, J. Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press, 1975., – 232 p.

18. The practical handbook of genetic algorithms / ed. L. D. Chambers. – Florida : CRC Press, 2000. – Vol. I: Applications. – 520 p.

19. The practical handbook of genetic algorithms / ed. L. D. Chambers. – Florida : CRC Press, 2000. – Vol. II: New frontiers. – 421 p.

20. The practical handbook of genetic algorithms. / ed. L. D. Chambers. – Florida: CRC Press LLC, 2000. – Vol. III: Complex coding systems. – 659 p.

21. Cantu-Paz E. Efficient and accurate parallel genetic algorithms / E. Cantu-Paz. – Massachusetts: Kluwer Academic Publishers, 2001. – 162 p.

22. Altenberg, L. The evolution of evolvability in genetic programming. In K. E. Kinnear, Jr., ed., Advances in Genetic Programming. MIT Press, 1994. – 120 p.

23. Davis, L. Handbook of Genetic Algorithms. Van Nostrand Reinhold, 1991. – 385 p.

24. Экономика и организация производства программных продуктов [Електронний ресурс]. – 2019. – Режим доступу: http://megaobzor.net/e-book/book.html

25. Депутат О. П. Цивільна оборона. Навчальний посібник. / О. П. Депутат, І. В. Коваленко, І. С. Мужик. — Львів.: Афіша, 2001. — 336 с.

ДОДАТОК А

Технічне завдання

Вступ

Програма призначена для пошуку оптимального маршруту на основі мультиагентного підходу.

А.1 Підстави для розробки

Підставою для розробки є завдання на дипломну роботу на тему «Дослідження та програмна реалізація мультиагентних методів пошуку оптимального маршруту», затверджене наказом по Національному університету «Запорізька політехніка».

А.2 Призначення розробки

Програма призначена для пошуку оптимального маршруту на основі мультиагентного підходу.

А.3 Вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Програма має виконувати такі функції:

– програма повинна створювати випадкові графи заданої розмірності, дозволяти користувачеві самостійно будувати графи;

– для створюваних графів має бути передбачено відображення вершин і опциональное відображення ребер із зазначенням їх ваг;

– програма повинна відображати користувачу процес виконання мультиагентного методу після кожної ітерації у вигляді певних зображень;

– по завершенню роботи методу розроблювана програма повинна відображати довжину маршруту, який було знайденого в результаті виконання методу, а також час роботи методу.

Інтерфейс користувача повинен забезпечити зручну роботу користувача з програмою.

А.3.2 Вимоги до надійності

Програма повинна корректно працювати з даними користувача. Повинна забезпечуватись сумісність реальної інформації й інформації представленої користувачеві. Повинно забезпечуватися стійке функціонування програми

А.3.3 Вимоги до експлуатації

Програма може бути записана як на твердому магнітному диску, так і на флеш носії. Експлуатація програмного продукту здійснюється відповідно до експлуатаційної документації на розроблену програму, що містить інформацію, необхідну для освоєння та експлуатації програмного продукту.

А.3.4 Вимоги до складу й параметрів технічних засобів

Для роботи з програмним продуктом рекомендується використовувати наступне апаратне забезпечення:

– ПК під керівництвом операційної системи та встановленим.Net Framework версії 4.5 та вище;

– тактова частота процесора 2 ГГц або більше;

– обсяг оперативної пам'яті не менше 4 Гб;

– обсяг вільного простору на диску – не менше 1 Гб;

– дозвіл екрана монітора не менше 1024x768;

– маніпулятор миша.

А.3.5 Вимоги до інформаційної й програмної сумісності

Для роботи програми необхідна операційна система.

А.3.6 Вимоги до транспортування і зберігання

Програма повинна зберігатися на жорсткому або гнучкому магнітному диску і транспортуватися в спеціальних боксах.

А.3.7 Вимоги до програмної документації

Програма повинна поставлятися з технічним завданням, керівництвом оператора, керівництвом програміста.

А.4 Стадії та етапи розробки

Програмний продукт повинен бути виконаний у 16-ти тижневий строк з моменту отримання завдання.

ДОДАТОК Б

Опис програми

Б.1 Загальні відомості

Програма розроблена за допомогою мови програмування C#.

Б.2 Функціональне призначення

Програма призначена для пошуку оптимального маршруту на основі мультиагентного підходу.

Б.3 Опис логічної структури

Програму розроблено на основі модульної структури. У проєкті використовуються глобальні модулі, у яких здійснюється пошук оптимального шляху.

На рисунку Б.1 зображено алгоритм функціонування програми. На початковому етапі користувач повинен створити граф та обрати необхідні параметри. До параметрів відносяться – кількість ітерацій, кількість мурах, розмір альфа, розмір бета, кількість вершин, коефіцієнт випаровування феромону, феромоний коефіцієнт Q.

Після завантаження даних і запуску алгоритму, почнеться пошук оптимального рішення. Пошук буде виконуватися у новому потоці, тому користувач може змінювати параметри чи граф для підготовки до нового запуску.

Використані методи у програмному забезпеченні наведені у таблиці Б.1.

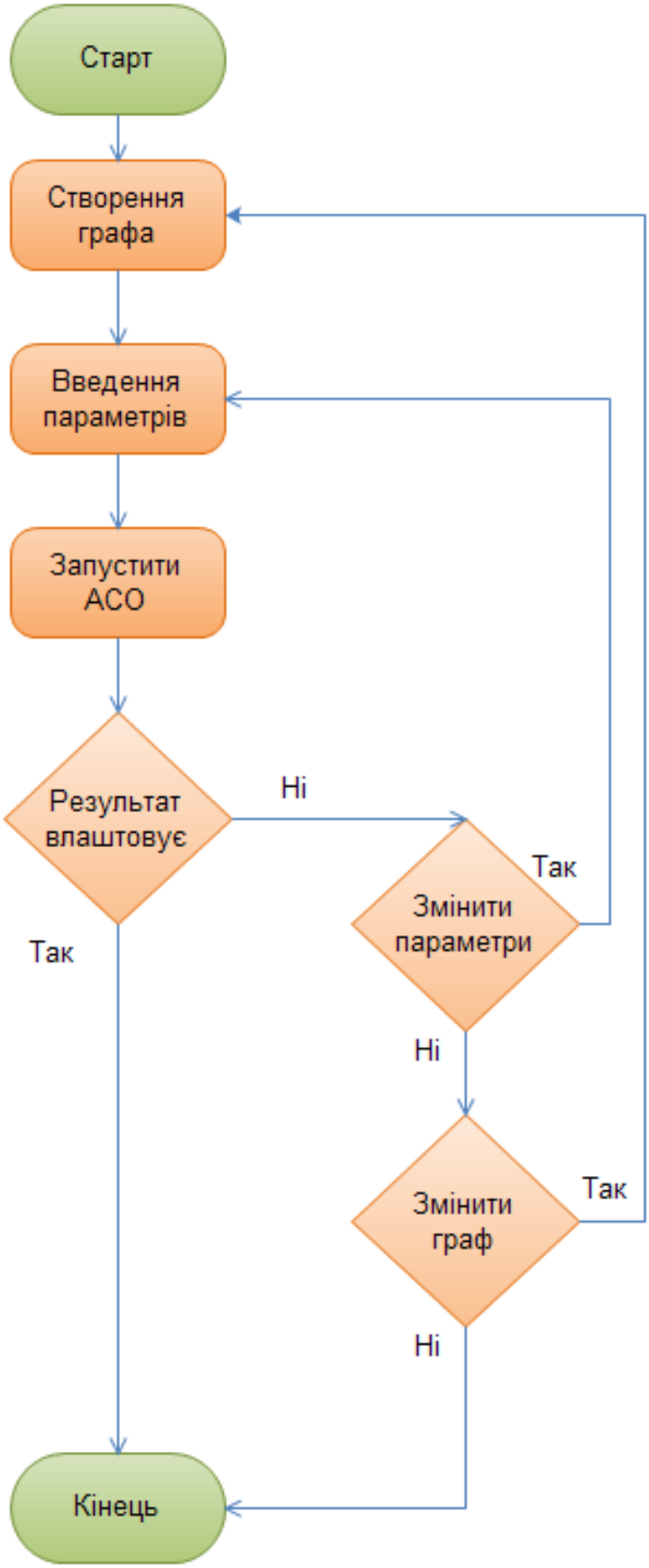


Рисунок Б.1 – Алгоритм програми

Таблиця Б.1 – Використані методи у ПЗ

Метод	Опис методу
Ant()	Конструктор за замовчуванням
Ant(int startPos, int num_cities)	Конструктор: створює мурашки в вершині startPos і списком табу розміру num_cities
City()	Конструктор за замовчуванням: створює вершину з координатами (0; 0)
City(Point p)	Конструктор: створює вершину з заданими координатами
void Draw(Graphics g)	Малює вершину
void DrawStart(Graphics g)	Малює початкову вершину маршруту
void DrawCurrent(Graphics g)	Малює вершину на поточному кроці
void DrawFinish(Graphics g)	Малює кінцеву вершину маршруту
Point getLocation()	Повертає координати вершини
void setLocation(int x, int y)	Задає координати вершини
double getDistance()	Повертає пройдене мурахою відстань
void resetDistance()	Скидає пройдене мурахою відстань, привласнюючи йому значення 0
void set_next_location(int location)	Задає точку location як наступне розташування мурашки
void update_total_distance(double d)	Додає до пройденої відстані величину d

Б.4 Використані технічні засоби

Для роботи з програмою рекомендується використовувати наступне апаратне забезпечення:

– ПК під керівництвом операційної системи та встановленим.Net Framework версії 4.5 та вище;

– тактова частота процесора 2 ГГц або більше;

– обсяг оперативної пам'яті не менше 4 Гб;

– обсяг вільного простору на диску – не менше 1 Гб;

– дозвіл екрана монітора не менше 1024x768;
– маніпулятор миша.

Б.5 Виклик і завантаження

Розроблене програмне забезпечення поширюється у вигляді інсталяційного пакета. Для можливості роботи з програмою необхідно розмістити файли модулів програмного забезпечення на комп’ютері. Перед використанням програми її необхідно встановити. Для цього необхідно запустити програму setup.exe, яка знаходиться на установчому диску. У відповідь на запит установчої програми потрібно вказати шлях встановлення програми. Після цього програма скопіює необхідні файли, створить і помістить в головне меню системи ярлик для запуску програми і зробить необхідні зміни в файлі реєстрації.

Після завершення процедури установки програму можна використовувати. Програму можна запустити будь-якими засобами операційної системи користувача.

Завантаження й запуск програми відбувається шляхом виконання користувачем .exe-файлу. Програма завантажується ОС до оперативної пам’яті, та починає своє виконання з функції main().

Б.6 Вхідні дані

Вхідними даними для програми є інформація про координати точок, які необхідно пройти комівояжеру, а також початкові параметри мультиагентного методу.

Б.7 Вихідні дані

Вихідними даними для програми є результати розрахунків щодо пошуку оптимального шляху, зокрема: мінімальний шлях знайдений на кожній ітерації, візуалізація шляху, графіки залежності номеру ітерації, кількість ітерацій, що знадобилася для вирішення задачі, найоптимальніший шлях.

ДОДАТОК В

Текст програми

```
        // ant_City.cs

        namespace ant_colony
        {
            public partial class ant_colony_s_path : Form
            {
                public Bitmap myCanvas;
                private List<Ant> ant_list;
                private List<City> city_list;
                private List<int> best_tour_list;
                private List<int> brute_force_list;
                private List<int> brute_forceTmpList;
                private double best_tour_length = -1;
                private double brute_force_length = -1;
                private double[,] distances;
                private double[,] pheromones;
                private int counter = 0;
                private int num_cities = 10;
                private int num_ants = 30;
                private int pherom_const = 100;
                private double ALPHA = 1.0;//weight of pheromone
                private double BETA = 1.0;//weight of distance
                private double RHO = 0.5;//decay rate
                private int iterations = 100;
                private int click_counter = 0;
                private bool random_cities = false;
                private bool choose_cities = false;
                private Stopwatch bruteWatch = new Stopwatch();
                private Stopwatch acoWatch = new Stopwatch();

                private Random rand_gen = new Random();
                public DoubleBufferPanel DrawingPanel = new DoubleBufferPanel();
                public ant_colony_s_path()
                {
                    InitializeComponent();
                    DrawingPanel.Size = new System.Drawing.Size(400, 400);
                    DrawingPanel.Location = new System.Drawing.Point(12, 12);
                    DrawingPanel.Paint += new
System.Windows.Forms.PaintEventHandler(this.DrawingPanel_Paint);
                    DrawingPanel.MouseClick += new
System.Windows.Forms.MouseEventHandler(this.DrawingPanel_MouseClick);
                    DrawingPanel.Parent = this;
                    this.Controls.Add(DrawingPanel);
                }

                private void ant_colony_s_path_Load(object sender, EventArgs e)
                {
                    myCanvas = new Bitmap(DrawingPanel.Width, DrawingPanel.Height,
                       System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                    Graphics g = Graphics.FromImage(myCanvas);
                    g.Clear(Color.White);
                    num_cities = int.Parse(number_Cities_Box.Text);
                    num_ants = int.Parse(number_Ants_Box.Text);
                    city_list = new List<City>();
                    ant_list = new List<Ant>();
                    best_tour_list = new List<int>();
                }
                private void DrawingPanel_Paint(object sender, PaintEventArgs e)
                {
                    Graphics g = e.Graphics;
                    g.DrawImage(myCanvas, 0, 0, myCanvas.Width, myCanvas.Height);
```

```
        }

        /*****************************************************************
         * Draw_button_Click: initiate ant colony method
         *****************************************************************/
        private void Draw_button_Click(object sender, EventArgs e)
        {
            if (click_counter == int.Parse(number_Cities_Box.Text) &&
!backgroundWorker1.IsBusy)
            {
                acoWatch.Start();
                choose_cities = false;
                random_cities = false;
                Graphics g = Graphics.FromImage(myCanvas);
                initAnts();

                if (!backgroundWorker1.IsBusy)
                {
                    backgroundWorker1.RunWorkerAsync();
                }
            }

        }


        /*****************************************************************
         * DrawingPanel_MouseClick: click generates city
         *****************************************************************/
        private void DrawingPanel_MouseClick(object sender, MouseEventArgs e)
        {
            Graphics g = Graphics.FromImage(myCanvas);
            num_cities = int.Parse(number_Cities_Box.Text);
            if (choose_cities == true && click_counter < num_cities)
            {
                City myCity = new City(e.Location);
                myCity.Draw(g);
                city_list.Add(myCity);
                click_counter++;
            }
            if (choose_cities == true && click_counter == num_cities)
            {
                initialize_AntColony(g);
            }
            DrawingPanel.Invalidate();

        }

        /*****************************************************************
         * random_cities_button_Click: user wants to get random cities
         *****************************************************************/
        private void random_cities_button_Click(object sender, EventArgs e)
        {
            if (number_Cities_Box.Text == "0") {
                MessageBox.Show("Введите количество вершин!");
            }
            else {
            Graphics g = Graphics.FromImage(myCanvas);
            g.Clear(Color.White);
            random_cities = true;
            initialize_AntColony(g);
            DrawingPanel.Invalidate();
            }
```

```
        }

        /*****************************************************************
         * define_cities_Button_Click: user wants to click/select cities
         *****************************************************************/
        private void define_cities_Button_Click(object sender, EventArgs e)
        {
            if (number_Cities_Box.Text == "0")
            {
                MessageBox.Show("Введите количество вершин!");
            }
            else
            {

                num_cities = int.Parse(number_Cities_Box.Text);
                choose_cities = true;
                random_cities = false;
            }
        }


        /*****************************************************************
         * initialize_AntColony: intialize values for ant colony
         *****************************************************************/
        private void initialize_AntColony(Graphics g)
        {
            ant_list = new List<Ant>();
            best_tour_list = new List<int>();
            best_tour_length = -1;
            num_cities = int.Parse(number_Cities_Box.Text);
            num_ants = int.Parse(number_Ants_Box.Text);
            distances = new double[num_cities, num_cities];
            pheromones = new double[num_cities, num_cities];
            initCities(g);
            initAnts();
            initPherom();
        }
        /*****************************************************************
         * initAnts: add ants to random starting position
         *****************************************************************/
        private void initAnts()
        {
            int rand_city = 0;
            ant_list.Clear();
            num_ants = int.Parse(number_Ants_Box.Text);
            for (int i = 0; i < num_ants; i++)
            {
                rand_city = rand_gen.Next(0, num_cities);
                ant_list.Add(new Ant(rand_city, num_cities));//random start
location
                ant_list[i].tourList[0] =
ant_list[i].get_current_location();//set tour's first position as current location
                ant_list[i].haveBeenList[ant_list[i].get_current_location()] =
1;//set to 1 to designate that we went to this city
                ant_list[i].tour_number = 1;
            }
        }
        /*****************************************************************
         * initCities: add cities with random staring positions
         *             //can be changed to allow user input
         *****************************************************************/
        private void initCities(Graphics g)
```

```
                {
                    int rand_city_x = 0;
                    int rand_city_y = 0;
                    distances = new double[num_cities, num_cities];
                    //initialize cities to random positions
                    if (random_cities == true)
                    {
                        city_list = new List<City>();
                        click_counter = num_cities;
                        for (int i = 0; i < num_cities; i++)
                        {
                            rand_city_x = rand_gen.Next(30, myCanvas.Width - 30);
                            rand_city_y = rand_gen.Next(30, myCanvas.Height - 30);
                            city_list.Add(new City(new Point(rand_city_x,
rand_city_y)));
                            city_list[i].Draw(g);
                        }
                    }
                    //compute city distances
                    //(n^2-n)/2 == number of connections btwn cities
                    for (int i = 0; i < num_cities; i++)
                        for (int k = 0; k < num_cities; k++)
                        {
                            double x = Math.Pow((double)city_list[i].getLocation().X -
                                (double)city_list[k].getLocation().X, 2.0);
                            double y = Math.Pow((double)city_list[i].getLocation().Y -
                                (double)city_list[k].getLocation().Y, 2.0);
                            distances[i, k] = Math.Sqrt(x + y);
                        }

                }

/*************************************************************************
                 * initPherom: initialize pheromone levels btwn cities to  a small
constant

*************************************************************************/
                private void initPherom()
                {
                    for (int from = 0; from < num_cities; from++)
                    {
                        for (int to = 0; to < num_cities; to++)
                        {//initialize all pheromone btwn cities as a small constant
                            pheromones[from, to] = 1.0 / (double)num_cities;
                            pheromones[to, from] = 1.0 / (double)num_cities;
                        }
                    }
                }



                private void backgrondWorker1_DoWork(object sendr, DoWorkEventArgs e)
                {
                    Calculate();
                }


/*********************************************************************
                 * backgrondWorker1_RunWorkerComplted: draw soluton based on ACO

*********************************************************************/
```

```
                private void backgroundWorker1_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
                {
                    Graphics g = Graphics.FromImage(myCanvas);
                    g.Clear(Color.White);
                    Best_Tour_Box.Text = Math.Round(best_tour_length, 2).ToString();

                    //draw cities
                    for (int i = 0; i < num_cities; i++)
                    {
                        city_list[i].Draw(g);
                    }
                    //draw solutions
                    for (int i = 0; i < best_tour_list.Count; i++)
                    {
                        g.DrawLine(new Pen(Color.Black, 5),
city_list[best_tour_list[i]].getLocation(),
                            city_list[best_tour_list[(i + 1) %
num_cities]].getLocation());//loop back to last point

                        if (i == best_tour_list.Count - 1)
                        {
                            best_tour_length = -1;
                            best_tour_list.Clear();
                        }
                    }
                    DrawingPanel.Invalidate();
                    acoWatch.Stop();
                    totalTimeACO.Text = acoWatch.Elapsed.TotalSeconds.ToString();
                    acoWatch.Reset();
                }



/*****************************************************************
                 * Calculate: main function that calculates ant movement,
                 *          evaporation and increment of pheromones

*****************************************************************/
                private void Calculate()
                {
                    ALPHA = double.Parse(alpha_Box.Text);
                    BETA = double.Parse(beta_Box.Text);
                    RHO = double.Parse(rho_Box.Text);
                    iterations = int.Parse(iterationBox.Text);
                    for (int k = 0; k < iterations; k++)
                    {
                        for (int i = 0; i < num_cities; i++)//move ants till they reach
the end
                            if (ants_stop()) //moves ants 1 step & check if all ants
finished moving?
                            {
                                evaporatePheromones();
                                updatePheromones();
                                best_tour();//go through every ant and check if it has
optimal solution
                                initAnts(); // reset ant position and tour
                            }
                    }
                }
```

```
/**************************************************************************
******
                 * goToNextCity: picks next city based on the attractiveness of the
path(the pheromones)
                 *              and its visibility (the distance)

**************************************************************************
******/
                private void goToNextCity(Ant current_ant)
                {
                    double sum_prob = 0;//denominator in probability function
                    double move_prob = 0;//numerator in probability function
                    int current_city = current_ant.get_current_location();
                    for (int i = 0; i < num_cities; i++)//loop through all cities
                    {
                        if (current_ant.haveBeenList[i] == 0)
                        {
                            sum_prob += Math.Pow(pheromones[current_city, i], ALPHA) *
                                Math.Pow(1.0 / distances[current_city, i], BETA);
                        }
                    }

                    int destination_city = 0;
                    double rand_move = 0;
                    int count = 0;
                    //loops until ant chooses a city
                    while (count < 400)//400 is the threshold for loops
                    {
                        if (current_ant.haveBeenList[destination_city] == 0)//ant hasnt
been to  this city
                        {//calculate probability of movement
                            move_prob = (Math.Pow(pheromones[current_city,
destination_city], ALPHA) *
                                Math.Pow(1.0 / distances[current_city,
destination_city], BETA)) / sum_prob;
                            rand_move = rand_gen.NextDouble();
                            if (rand_move < move_prob) break;//break loop if ant moves
to city
                        }
                        destination_city++;
                        if (destination_city >= num_cities) destination_city =
0;//reset city count
                        count++;
                    }
                    //update next location and tour itinerary
                    current_ant.set_next_location(destination_city);//going to that
city
                    current_ant.haveBeenList[destination_city] = 1;//moved to that city
                    current_ant.tourList[current_ant.tour_number] = destination_city;
                    current_ant.tour_number++;
                    //add to current distance

current_ant.update_total_distance(distances[current_ant.get_current_location(),
destination_city]);

                    //if the ant reached the end, add up the distance for return path
                    if (current_ant.tour_number == num_cities)
                    {
                        current_ant.update_total_distance(
                            distances[current_ant.tourList[num_cities - 1],
current_ant.tourList[0]]);
```

```
                }

                current_ant.set_current_location(destination_city);//update current
city to next city
            }


/*****************************************************************************
             * ants_stop: checks if all the ants have finished moving/reached final
destination

*****************************************************************************/
            private bool ants_stop()
            {
                int moved = 0;
                for (int i = 0; i < num_ants; i++)
                {
                    if (ant_list[i].tour_number < num_cities)
                    {//ants are still in moving
                        goToNextCity(ant_list[i]);
                        moved++;
                    }
                }
                if (moved == 0)
                {
                    return true;//ants have finished moving
                }
                else return false;
            }


/*****************************************************************************
             * evaporatePheromones: decreases current pheromones by a factor of
(1.0-RHO) so
             *                   larger values of pheromones would decrease at a
higher rate,
             *                   but this is good, since we want the ants to
explore more

*****************************************************************************/
            public void evaporatePheromones()
            {
                //handles both cases of [i,k] and [k,i] so dont need to code
pheromones 2x
                for (int i = 0; i < num_cities; i++)
                    for (int k = 0; k < num_cities; k++)
                    {
                        pheromones[i, k] *= (1.0 - RHO);
                        //pheromone levels should always be at base levels
                        if (pheromones[i, k] < 1.0 / (double)num_cities)
                        {
                            pheromones[i, k] = 1.0 / (double)num_cities;
                        }
                    }
            }



/*****************************************************************************
```

```
                 * updatePheromones: update pheromones along all edges after each ant
has completed
                 *                  its tour

*********************************************************************************/
                private void updatePheromones()
                {
                    for (int i = 0; i < num_ants; i++)
                    {
                        for (int k = 0; k < num_cities; k++)
                        {
                            int from = ant_list[i].tourList[k];//starting point of edge
                            //if city+1=num_cities, then city is last city and then
destination is the starting city
                            int to = ant_list[i].tourList[((k + 1) %
num_cities)];//endpoint of edge
                            pheromones[from, to] += (double)pherom_const /
ant_list[i].getDistance();
                            pheromones[to, from] = pheromones[from, to];

                        }
                    }
                }

/*********************************************************************************
                 * best_tour: updates global tour length with shortest tour length

*********************************************************************************/
                private void best_tour()
                {
                    double best_local_tour = ant_list[0].getDistance();
                    int save_index = 0;
                    for (int i = 1; i < ant_list.Count; i++)//checks the best tour
length among this iteration
                    {
                        if (ant_list[i].getDistance() < best_local_tour)
                        {
                            best_local_tour = ant_list[i].getDistance();
                            save_index = i;
                        }
                    }
                    //compare best local length with global length and update
accordingly
                    if (best_local_tour < best_tour_length || best_tour_length == -1)
                    {
                        best_tour_list = ant_list[save_index].tourList;
                        best_tour_length = best_local_tour;
                    }
                }

                private void clear_button_Click(object sender, EventArgs e)
                {
                    Graphics g = Graphics.FromImage(myCanvas);
                    g.Clear(Color.White);
                    Best_Tour_Box.Text = "0";
                    totalTimeACO.Text = "0";
                    totalTimeBrute.Text = "0";
                    brute_length_label.Text = "0";
                    ant_list.Clear();
                    city_list.Clear();
                    click_counter = 0;
                    best_tour_length = -1;
```

```
                    random_cities = false;
                    best_tour_list.Clear();
                    pheromones = new double[num_cities, num_cities];
                    distances = new double[num_cities, num_cities];
                    DrawingPanel.Invalidate();
                }


/*************************************************************************
                 * permute: generate all permutations of List v

*************************************************************************/
                private void permute(int total,List<int> v, int start, int n)
                {
                    int percentProgress = (int)(100.0 * ((float)(counter+1) /
(float)total));
                    if( percentProgress %5 ==0)
                        backgroundWorker2.ReportProgress(percentProgress);


                    if (start == n - 1)
                    {
                        double local_length = 0;
                        counter++;
                        for (int i = 0; i < v.Count; i++)
                        {
                            local_length +=
Math.Sqrt(Math.Pow(city_list[v[i]].getLocation().X - city_list[v[(i + 1) %
num_cities]].getLocation().X, 2.0) +
                                Math.Pow(city_list[v[i]].getLocation().Y -
city_list[v[(i + 1) % num_cities]].getLocation().Y, 2.0));
                        }
                        if (local_length < brute_force_length || brute_force_length ==
-1)
                        {
                            brute_force_length = local_length;
                            brute_forceTmpList.Clear();
                            for (int i = 0; i < v.Count; i++)
                            {
                                brute_forceTmpList.Add(v[i]);
                            }
                        }
                    }
                    else
                    {
                        for (int i = start; i < n; i++)
                        {
                            int tmp = v[i];

                            v[i] = v[start];
                            v[start] = tmp;
                            permute(total,v, start + 1, n);
                            v[start] = v[i];
                            v[i] = tmp;
                        }
                    }
                }
                //recursion for n!
                public int fact(int n)
                {
                    if (n == 0) return 1;
                    else return n * fact(n - 1);
```

```
            }

/*************************************************************************
             * brute_force_button_Click: initiate brute force function

*************************************************************************/
            private void brute_force_button_Click(object sender, EventArgs e)
            {
                counter = 0;
                int temp_city;
                string s = number_Cities_Box.Text;
                bool result = int.TryParse(s, out temp_city);
                if (result && city_list.Count != 0 && !backgroundWorker2.IsBusy)
                {
                    Graphics g = Graphics.FromImage(myCanvas);
                    num_cities = int.Parse(number_Cities_Box.Text);
                    brute_force_list = new List<int>();
                    brute_forceTmpList = new List<int>();
                    for (int i = 0; i < num_cities; i++)
                    {
                        brute_force_list.Add(i);
                    }
                    backgroundWorker2.RunWorkerAsync();
                }
            }


/*************************************************************************
             * backgroundWorker2_DoWork: draw brute force solution

*************************************************************************/
            private void backgroundWorker2_DoWork(object sender, DoWorkEventArgs e)
            {
                bruteWatch.Start();
                int total = fact(num_cities);
                permute(total,brute_force_list, 0, num_cities);
            }
            private void backgroundWorker2_ProgressChanged(object sender,
ProgressChangedEventArgs e)
            {
                progressBar1.Value = e.ProgressPercentage;
            }

/*************************************************************************
             * backgroundWorker2_RunWorkerCompleted: draw brute force solution

*************************************************************************/
            private void backgroundWorker2_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
            {
                Graphics g = Graphics.FromImage(myCanvas);

                for (int i = 0; i < brute_forceTmpList.Count; i++)
                {
                    g.DrawLine(new Pen(Color.Red),
city_list[brute_forceTmpList[i]].getLocation(),
                        city_list[brute_forceTmpList[(i + 1) %
num_cities]].getLocation());
                }
                brute_length_label.Text = Math.Round(brute_force_length,
2).ToString();
```

```
                bruteWatch.Stop();
                totalTimeBrute.Text = bruteWatch.Elapsed.TotalSeconds.ToString();
                bruteWatch.Reset();
                brute_force_length = -1;
                brute_forceTmpList.Clear();
                DrawingPanel.Invalidate();
            }


        }


/*****************************************************************************
        * DoubleBufferPanel: a child class of the Systems.Windows.Forms.Panel
        *                    Allows double buffering
        *
*****************************************************************************/
        public class DoubleBufferPanel : Panel
        {

            public DoubleBufferPanel()
            {
                // Set the value of the double-buffering style bits to true.
                // ControlStyles.UserPaint -- allows user to control painting w/o
passing off the work to the operating system
                //ControlStyles.AllPaintingInWmPaint--optimize to reduce flicker
but only use it if ControlStyles.UserPaint is true
                this.DoubleBuffered = true;
                this.SetStyle(ControlStyles.DoubleBuffer | ControlStyles.UserPaint
|
                 ControlStyles.AllPaintingInWmPaint, true);// | evaluates all
conditions even if condition 1 is true
                this.UpdateStyles();//forces style to be applied
            }

        }
    }
```

ДОДАТОК Д

Слайди презентації

Рисунок Д.1 – Слайд № 1

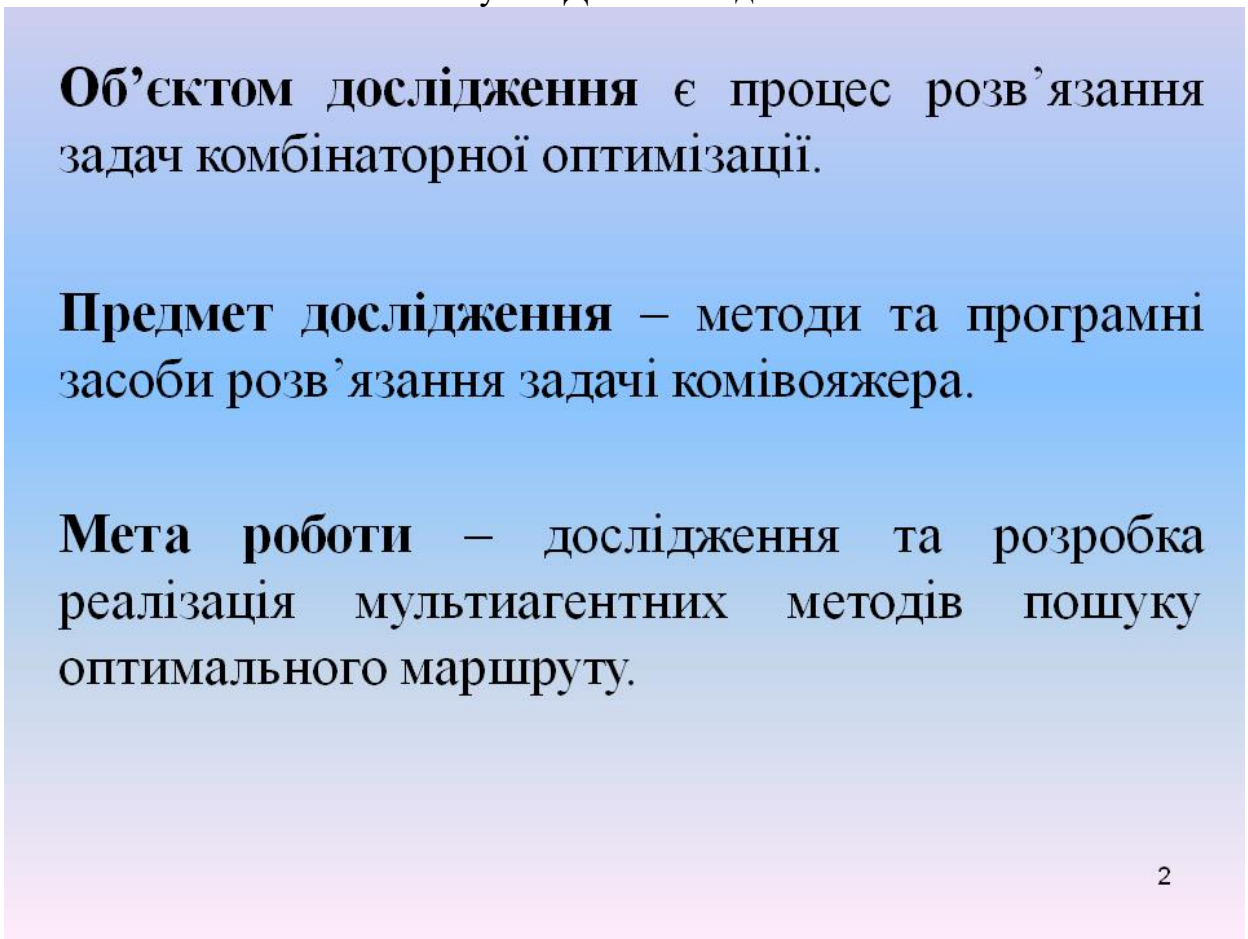


Рисунок Д.2 – Слайд № 2

ЗАДАЧІ РОБОТИ

– аналіз методів розв'язання задачі комівояжера;
– розробка технічного завдання для реалізації програми;
– вибір середовища розробки програмного забезпечення;
– розробка модифікованого мультиагентного методу для розв'язання задачі комівояжера;
– конструювання програмного забезпечення пошуку оптимального маршруту на основі мультиагентного підходу;
– тестування розробленого програмного забезпечення.

3

Рисунок Д.3 – Слайд № 3

МУЛЬТИАГЕНТНІ МЕТОДИ МУРАШИНИХ КОЛОНІЙ

Критерій	Метод			
	Ant System	ASrank	Ant Colony System	Max-min Ant System
Додавання феромонів	Здійснюється після отримання рішення		Відбувається в процесі створення рішення	
Розв'язувані завдання	TSP, QAP, JSP, VRP, SCSP	TSP	TSP, JSP	TSP, QAP
Принцип елітної стратегії	кожен агент бере участь в оновленні шляхів	в оновленні беруть участь (w-1) локально кращих агентів і глобально кращий агент	Оновлення виконує кращий (глобально або локально) агент	

4

Рисунок Д.4 – Слайд № 4

МОДИФІКОВАНИЙ МУЛЬТИАГЕНТНИЙ МЕТОД ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА

У розробленому модифікованому мультиагентному методі для розв'язання задачі комівояжера запропоновано додавати у множину агентів допоміжні «жадібні» мурахи.

Правило, яке визначає ймовірність переходу *k*-ої мурахи з міста *i* у місто *j*:

$$P_{ij,k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[n_{ij}]^{\beta}}{\sum_{l\in J_{i,k}}[\tau_{il}(t)]^{\alpha}\cdot[n_{il}]^{\beta}}, & j\in J_{i,k};\\ 0, & j\notin J_{i,k}.\end{cases}$$

Кількість відкладеного агентом феромону може бути задано у вигляді:

$$\Delta\tau_{ij,k}(t)=\begin{cases}\dfrac{Q}{L_k(t)}, & (i,j)\in T_k(t);\\ 0, & (i,j)\notin T_k(t).\end{cases}$$

5

Рисунок Д.5 – Слайд № 5

ВИБІР ІНСТРУМЕНТАРІЮ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Критерії порівняння	C#	Borland Delphi 7.0	Microsoft Visual Basic
Синтаксис	Легко у використанні	простий, легко вивчаємий	простий
Компілятор	проводить статичну перевірку типів, але на додаток до цього робить перевірку коректності під час завантаження	вбудований, високопродуктивний	повноцінний
Зручність для використання	зручний	зручний	не дуже зручний
Робота з базами даних	взаємодіє з базами даних через мову SQL	інтегроване ядро процесора баз даних Borland Database Engine (BDE)	набор об'єктних бібліотек, що входять в.NET Framework (ADO.NET)
Сприйняття коду	гарне	гарне	погане
Мова програмування	C	на базі Object Pascal	Basic
Наявність бібліотек для математичних обчислень	велика кількість	достатньо стандартних компонентів	невелика кількість стандартних компонентів

6

Рисунок Д.6 – Слайд № 6

ДІАГРАМА КЛАСІВ РОЗРОБЛЕНОЇ ПРОГРАМИ

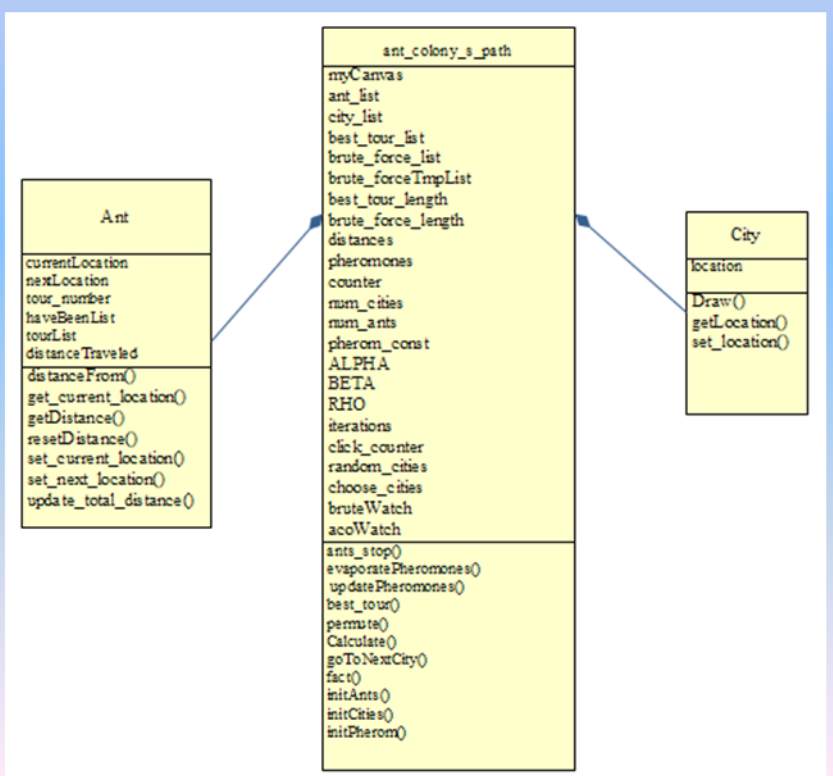


Рисунок Д.7 – Слайд № 7

ДІАГРАМИ СТАНІВ ТА ДІЯЛЬНОСТІ РОЗРОБЛЕНОЇ ПРОГРАМИ

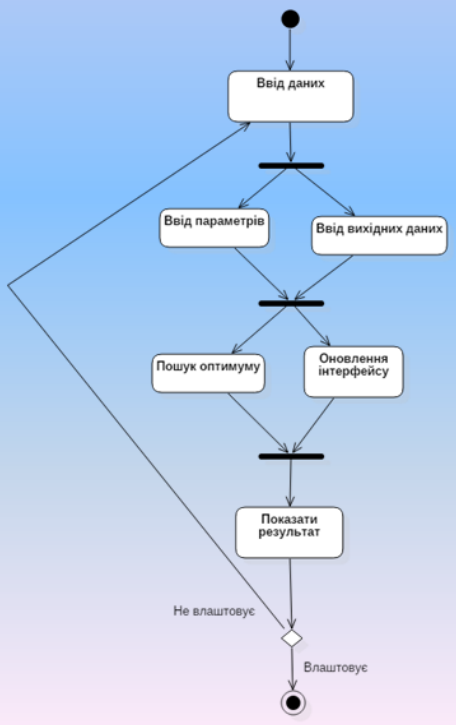
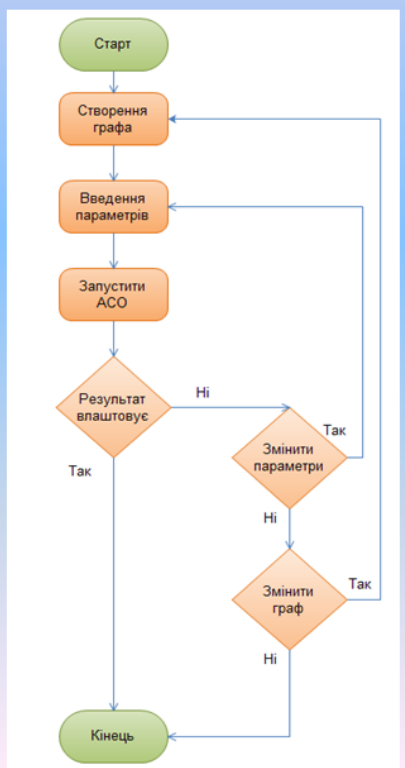


Рисунок Д.8 – Слайд № 8

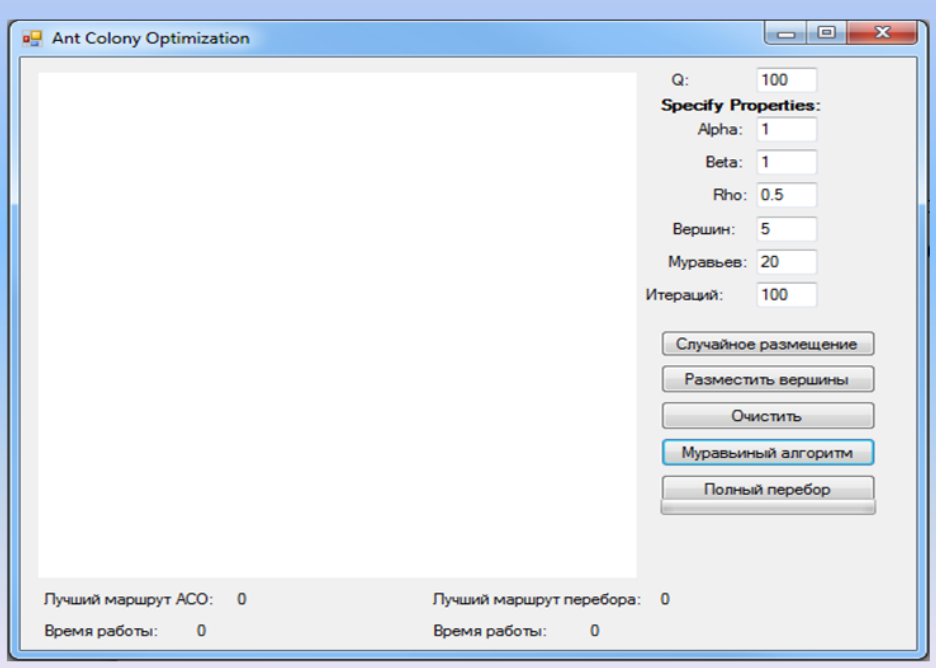
СХЕМА ФУНКЦІОНУВАННЯ РОЗРОБЛЕНОЇ ПРОГРАМИ



9

Рисунок Д.9 – Слайд № 9

ПРИКЛАД РОБОТИ З ПРОГРАМОЮ. ГОЛОВНЕ ВІКНО ПРОГРАМИ



10

Рисунок Д.10 – Слайд № 10

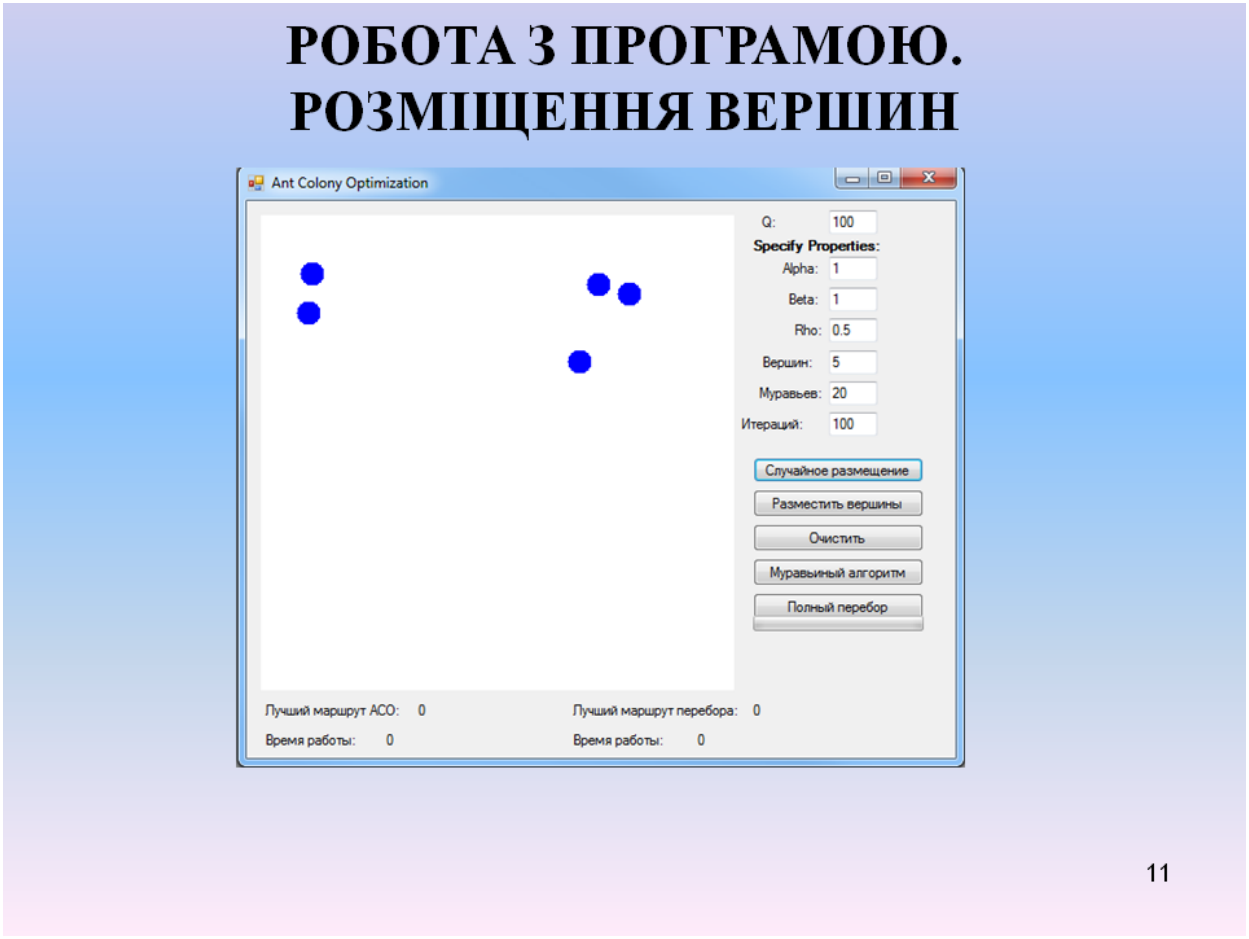


Рисунок Д.11 – Слайд № 11

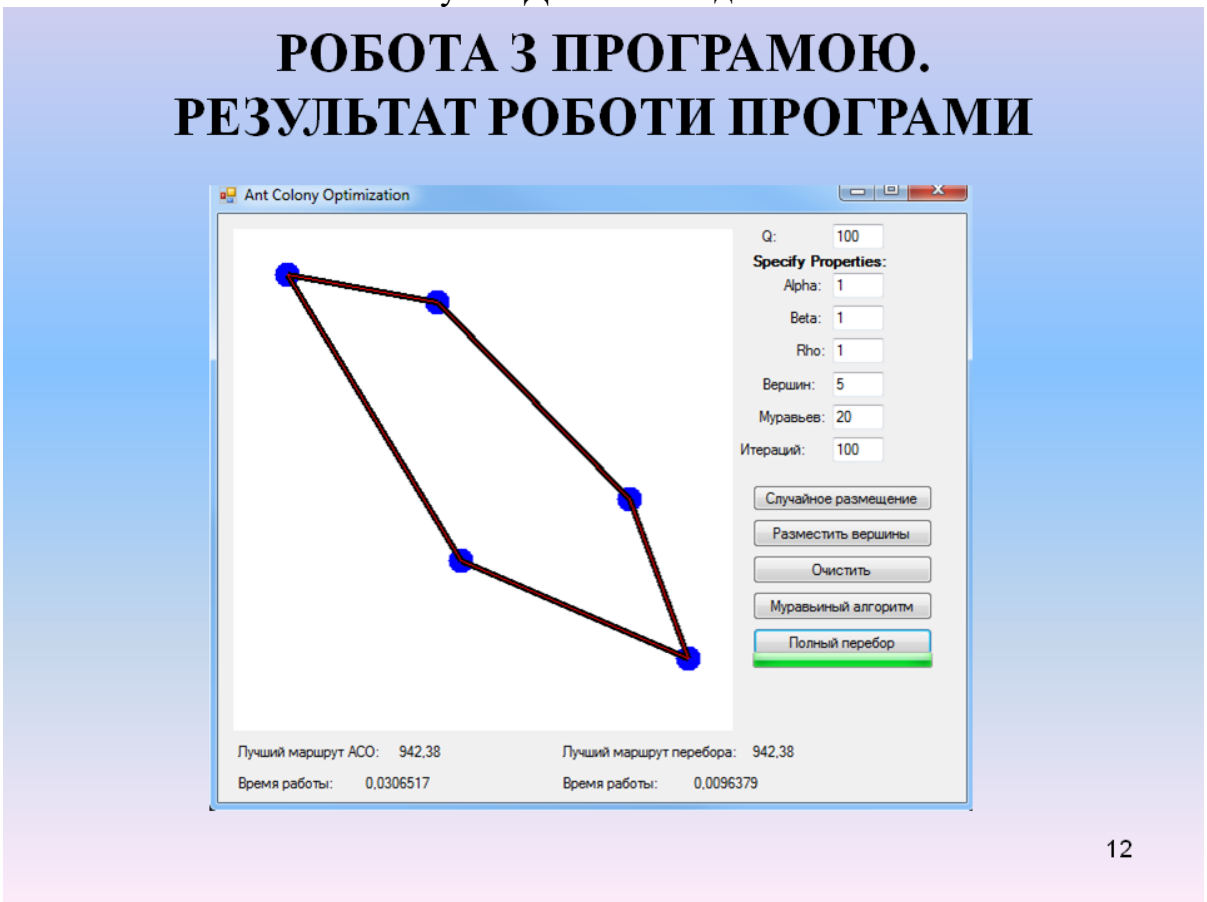


Рисунок Д.12 – Слайд № 12

ПОРІВНЯННЯ РОБОТИ МЕТОДІВ

№ експерименту	Кількість вершин	Найкращий маршрут ACO	Час виконання ACO	Найкращий маршрут повного перебору	Час виконання повного перебору
1	5	987,33	0,0423	987,33	0,00817
2	6	978,29	0,04232	978,29	0,01275
3	7	762,19	0,02717	762,19	0,024973
4	9	892,74	0,0417179	892,74	2,185017
5	10	1019,1	0,041566	1019,17	26,03718
6	11	949,81	0,04908	949,81	345,31919
7	15	1406,16	0,14606	--	--
8	50	2704,99	0,6448843	--	--
9	100	6201,73	2,5835	--	--

13

Рисунок Д.13 – Слайд № 13

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи магістра було досліджено та програмно реалізовано мультиагентні методи пошуку оптимального маршруту.

Вирішено такі завдання:

– аналіз методів розв'язання задачі комівояжера;

– розробка технічного завдання для реалізації програми;

– вибір середовища розробки програмного забезпечення;

– розробка модифікованого мультиагентного методу для розв'язання задачі комівояжера;

– конструювання програмного забезпечення пошуку оптимального маршруту на основі мультиагентного підходу;

– тестування розробленого програмного забезпечення.

14

Рисунок Д.14 – Слайд № 14