НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«На правах рукопису»
УДК _____

«До захисту допущено»
Завідувач кафедри
_____ Ролік О. І.
(підпис) (ініціали, прізвище)
“___”_____2019 р.

Магістерська дисертація
на здобуття ступеня магістра

зі спеціальності 126 – Інформаційні системи та технології
(код і назва спеціальності)

_____

на тему: Система побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів

Виконав: студент 6 курсу, групи ІА-з72мп
(шифр групи)

Кравчук Микола Петрович _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник к.т.н, доцент, Кравець П.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»

Факультет (інститут) інформатики і обчислювальної техніки
(повна назва)

Кафедра автоматики та управління в технічних системах
(повна назва)

Рівень вищої освіти – другий (магістерський)
(код і назва)

Спеціальність 126 – Інформаційні системи та технології
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О. І. Ролік
(підпис) (ініціали, прізвище)
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Кравчуку Миколі Петровичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Система побудови маршрутів кур'єрських перевезень з використанням мурашиних алгоритмів
науковий керівник дисертації Кравець Петро Іванович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 20___ р. №_____

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження: мурашиний алгоритм
_____

4. Предмет дослідження: способи та моделі застосування методу мурашиних колоній для вирішення задачі комівояжера
_____ __

5. Перелік завдань, які потрібно розробити: проаналізувати існуючі способи пошуку маршрутів; дослідити та реалізувати алгоритм мурашиних колоній для використання в програмній системі формування маршрутів для перевезень; здійснити підбір необхідних засобів та технологій для створення системи; здійснити проектування та розробку програмної системи, яка дозволить виконувати побудову маршрутів перевезень за допомогою мурашиного алгоритму; дослідити якість та швидкість розробленої функціональності _

6. Орієнтовний перелік ілюстративного матеріалу: структурна схема системи, діаграма варіантів використання системи, діаграма класів, діаграми послідовностей програмного забезпечення, діаграма бази даних системи, діаграма сутність-зв'язок, алгоритм побудови маршрутів з використанням мурашиних алгоритмів.

7. Орієнтовний перелік публікацій _____

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапу магістерської дисертації	Примітка
1	Отримання завдання та узгодження вихідних даних	20.10.2019	
2	Огляд досліджуваного об'єкту	29.10.2019	
3	Розробка архітектури системи	10.11.2019	
4	Реалізація функціональності системи	12.11.2019	
5	Тестування розробленої функціональності	19.11.2019	
6	Оформлення необхідної документації	28.11.2019	
7	Представлення роботи до попереднього захисту	03.12.2019	

Студент _____ М.П. Кравчук
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ П.І. Кравець
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація на тему «Система побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів» містить 113 аркушів пояснювальної записки, 39 таблиць, 33 рисунки, 8 додатків та 41 посилань на використані джерела.

Метою даної роботи є розробка системи для побудови оптимальних маршрутів з використанням алгоритму мурашки для компаній, які займаються кур’єрськими перевезеннями.

Об’єкт дослідження – мурашиний алгоритм.

Предмет дослідження – способи та моделі використання мурашиного алгоритму для знаходження найвигідніших рішень при вирішенні задачі комівояжера.

Актуальність дисертації обумовлена стрімким темпом використання систем побудови маршрутів, які використовуються в компаніях що займаються логістикою та кур’єрськими перевезеннями.

У процесі створення системи використовувались наступні технології розробки програмних продуктів: .NET, ASP.NET MVC та Entity Framework.

Ключові слова: мурашиний алгоритм, граф, маршрут, система, перевезення, задача комівояжера.

ABSTRACT

This master's thesis: “Routes construction system for courier transportations using ant colony optimization algorithm” contains 113 sheets of explanatory note, 39 tables, 33 pictures, 8 attachments and 41 references on used sources.

The purpose of the work is to develop a system for building optimal routes using ant colony optimization algorithm for courier companies.

Object of research – ant colony optimization algorithm.

Subject of research – methods and models of using ant algorithm for finding the best solutions when solving a salesman's problem.

The urgency of the dissertation is due to the rapid pace of use of route-building systems used by logistics and courier companies.

The following software development technologies were used in the development of the system: .NET, ASP.NET MVC and Entity Framework.

Key words: ant colony optimization algorithm, graph, route, system, transportation, travelling salesman problem.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9
1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПОШУКУ МАРШРУТІВ 11
1.1 Алгоритм А * 11
1.2 Метод гілок і меж 14
1.3 Метод найближчого сусіда 16
1.4 Алгоритм пошуку в глибину (ширину) 17
1.5 Алгоритм Дейкстри 19
1.6 Алгоритм Джонсона 21
1.7 Мурашиний алгоритм 23
Висновки до розділу 23
2 МУРАШИНИЙ АЛГОРИТМ В СИСТЕМІ ПОБУДОВИ МАРШРУТІВ 25
2.1 Основні поняття 25
2.1.1 Біологічна основа алгоритму 25
2.1.2 Початкова популяція 29
2.1.3 Переміщення мурахи 31
2.1.4 Приклад ітерації 33
2.1.5 Різновиди мурашиного алгоритму 37
2.2 Побудова маршрутів з використанням мурашиного методу 40
2.2.1 Структуризація та алгоритм реалізації методу мурашки 40
2.2.2 Побудова алгоритму пошуку 44
2.2.3 Порівняння вдосконаленого алгоритму з аналогом 48
Висновки до розділу 55
3 ОПИС АРХІТЕКТУРИ ТА РЕАЛІЗАЦІЯ СИСТЕМИ 57
3.1 Архітектури системи 57
3.1.1 Вибір засобів для реалізації системи 58
3.1.1.1 Вибір мов програмування 59
3.1.1.2 Вибір інтегрованого середовища розробки (IDE) 60

3.1.1.3 Фреймворк ASP.NET MVC 61
3.1.1.4 Технологія Entity Framework 62
3.1.1.5 СУБД Microsoft SQL Server 62
3.1.2 Проектування підсистем 63
3.1.2.1 Опис моделі бази даних 63
3.1.2.2 Опис структури серверної частини 64
3.1.2.3 Опис структури клієнтської частини 66
3.2 Реалізація системи 67
3.2.1 Реалізація структури бази даних 67
3.2.2 Реалізація серверної функціональності 70
3.2.3 Реалізація клієнтської функціональності 74
3.3 Тестування розробленої функціональності 78
Висновки до розділу 83
4 ДОСЛІДЖЕННЯ СИСТЕМИ ПОБУДОВИ МАРШРУТІВ 84
Побудована візуалізація результатів проведених експериментів наведена на рисунку 4.9. 90
Висновки до розділу 91
5 РОЗРОБКА СТАРТАП-ПРОЕКТУ 92
5.1 Опис ідеї проекту 92
5.2 Технологічний аудит проекту 93
5.3 Аналіз ринкових можливостей запуску стартап-проекту 93
5.4 Розроблення ринкової стратегії проекту 102
5.5 Розроблення маркетингової програми стартап-проекту 105
Висновки до розділу 108
ВИСНОВКИ 109
СПИСОК ПОСИЛАНЬ 110
Додаток А. Вихідний код функції пошуку з використанням мурашиного алгоритму
Додаток Б. Вихідний код контролера автентифікації

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ГА — генетичний алгоритм

ЕМА — евристичний мурашиний алгоритм

ЕОМ — електронна обчислювальна машина

МА — мурашиний алгоритм

МГГ — метод гілок і границь

МНВ — метод найменших відрізків

ПЗ — програмне забезпечення

РК — розрахунок концентрації

СО — самоорганізація

СУБД — система управління базами даних

ТМА — точний мурашиний алгоритм

ACOR — Ant Colony Optimization for continuous domain

API — Application Programming Interface

AS — Ant System

PDF — Probability Density Function

TSP — Traveling Salesman Problem

ВСТУП

Зростаючі з кожним днем темпи інформатизації суспільства підвищують важливість обчислювальної техніки в управлінських та інших типах процеів. Використання можливостей сучасних обчислювальних засобів для автоматизації процесу обробки інформації дає можливість підвищити продуктивність праці, та прискорити обмін різними типами інформації. Підприємства активно використовують обчислювальну техніку для ведення обліків, контролю за виконанням угод, проведенням різних операцій над діловими документами, управління процесами продаж тощо.

Зокрема найуспішніші компанії з продажу товарів різних типів не були б такими ефективними без організації доставки товару до споживача. В сучасному світі все більша кількість компаній та жителів різних країн користується послугами служб кур’єрських перевезень.

Внаслідок бурхливого розвитку обчислювальної техніки такі служби все частіше використовують засоби, які допоможуть їм вирішити проблеми раціонального розподілення вантажів, використання витратних ресурсів та побудови оптимальних маршрутів руху. Варто зауважити, що проблема використання витратних ресурсів тісно зв’язана з оптимальними маршрутами руху, так як при використанні максимально короткого шляху зменшується величина їх використання та економляться часові ресурси.

Проблема пошуку найкоротших шляхів є загальновідомою і важливою для різних сфер. Існує ряд алгоритмів для вирішення цього завдання. Останнім часом ця проблема інтенсивно вивчається для графів складної багаторівневої структури. У даній роботі розглядається задача пошуку найкоротших шляхів на графі від початкової вершини до певної кінцевої.

Актуальність дисертації зумовлена широким впровадженням інформаційних технологій в процеси перевезень, які проходять через колії, міста, складні транспортні розв'язки тощо. В таких місцях існує висока ймовірність зміни умов середовища, які будуть мати вплив на результат надання послуги і відповідно на кінцевого споживача.

В результаті цього з’являється потреба обійти негативний вплив при виникненні таких випадків, тому тематика даної роботи є досить актуальна.

Метою даної магістерської дисертації є розробка програмної системи побудови маршрутів кур’єрських перевезень з використанням мурашиного алгоритму.

Для досягнення поставленої мети у роботі необхідно виконати низку завдань:

- проаналізувати та оцінити існуючі способи пошуку маршрутів;
- дослідити та реалізувати алгоритм мурашиних колоній для використання в програмній системі формування маршрутів для перевезень;
- здійснити підбір необхідних засобів та технологій для створення високоякісної та швидкої системи;
- здійснити проектування та розробку програмної системи, яка дозволить виконувати побудову маршрутів перевезень за допомогою мурашиного алгоритму;
- дослідити якість та швидкість розробленої функціональності.

Об’єктом дослідження виступає мурашиний алгоритм, який використовується для пошуку та побудови оптимальних маршрутів на графах в умовах існування альтернативних рішень.

Предмет дослідження — способи та моделі застосування методу мурашиних колоній для знаходження найвигідніших рішень при вирішенні задачі комівояжера.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПОШУКУ МАРШРУТІВ

В даному розділі будуть розглянуті існуючі алгоритми пошуку шляхів на графах. Маршрут, або шлях є послідовністю ребер в неорієнтованому графі, в якому кінець кожного ребра збігається з початком наступного ребра. Число ребер маршруту називається його довжиною [7, с. 63].

1.1 Алгоритм A *

Пошук A * є алгоритмом пошуку по першому найоптимальнішому збігу, який в процесі виконання шукає маршрут мінімальної вартості з початкової вершини до кінцевої.

Послідовність проходження по вершинах обчислюється за допомогою евристичної функції, «відстань + вартість» (позначається як f (x)). Ця функція являє собою суму двох інших: функції вартості досягнення розглянутої вершини (x) з початкової (зазвичай позначається як g (x) і може бути як евристичною, так і неевристичною) і евристичної оцінки відстані від розглянутої вершини до кінцевої (позначається як h (x)) [32].

Функція h (x) повинна бути припустимою евристичною оцінкою, тобто не повинна переоцінювати відстані до цільової вершини. Наприклад, для завдання маршрутизації h (x) може являти собою відстань до цілі по прямій лінії, так як це фізично найменша можлива відстань між двома точками.

Опис алгоритму

A* покроково проходить всі шляхи, що ведуть від початкової вершини до кінцевої до моменту знаходження мінімального. Як і всі інформовані алгоритми пошуку, він переглядає спочатку ті маршрути, які потенційно ведуть до мети. Від інших алгоритмів, які також є алгоритмами пошуку по першому кращому збігу його відрізняє те, що при виборі вершини враховується, крім іншого, весь пройдений до неї шлях (складова g (x) - це вартість шляху від початкової вершини, а не від попередньої, як це відбувається в жадібному алгоритмі).

На початку роботи проглядаються вузли, суміжні з початковим; вибирається той з них, який має мінімальне значення f (x), після чого цей вузол розкривається. На кожному етапі алгоритм обробляє велику кількість шляхів з початкової точки до всіх ще не розкритих (листових) вершин графа («безліччю приватних рішень»), які розміщуються у черзі з пріоритетом. Пріоритет шляху визначається наступним чином: f (x) = g (x) + h (x). Алгоритм продовжує свою роботу до тих пір, поки значення f (x) цільової вершини не виявиться меншим, ніж будь-яке значення в черзі або поки все дерево не буде переглянуто. З множинних рішень вибирається рішення з найменшою вартістю.

Безліч вже пройдених вершин можна опустити (при цьому алгоритм перетвориться на пошук по дереву рішень), якщо відомо, що рішення існує, або якщо алгоритм пошуку оптимального маршруту вміє відсікати цикли.

Властивості

Алгоритм A * при існуванні рішення в будь якому випадку забезпечує його знаходження.

Якщо евристична функція h допустима, тобто ніколи не переоцінює дійсну мінімальну вартість досягнення мети, то A * сам є допустимим (або оптимальним), також за умови, що не будуть відсіктися пройдені вершини. Якщо ж така умова виконується, то для оптимальності алгоритму потрібно, щоб h (x) була ще й монотонною евристикою. Властивість монотонності означає, що якщо існують шляхи A -B -C і A- C (не обов'язково через B), то оцінка вартості шляху від A до C повинна бути менше, або дорівнювати сумі оцінок шляхів A -B і B -C. (Монотонність також відома як нерівність трикутника: одна сторона трикутника не може бути довше, ніж сума двох інших сторін). Математично, для всіх шляхів x, y (де y - нащадок x) виконується[31]:

$$g(x) + h(x) \leq g(y) + h(y) \qquad (1.1)$$

Алгоритм A * також оптимально ефективний для заданої евристики h. Це означає, що будь-який інший алгоритм досліджує не менше вузлів, ніж A * (за

винятком випадків, коли існує кілька приватних рішень з однаковою евристикою, точно відповідної вартості оптимального шляху).

Окремі випадки

У загальному випадку, пошук в глибину і пошук в ширину – це два окремі випадки алгоритму A *. Для пошуку в глибину береться глобальна змінна – лічильник, ініціалізований деяким великим значенням. Кожен раз при розкритті вершини присвоюється значення лічильника суміжним вершинам, при цьому зменшуючи його на одиницю після кожної операції присвоювання. В результаті виходить, що чим раніше буде відкрита вершина, тим більше значення h (x) вона отримає, а значить, буде переглянута в останню чергу. Якщо взяти h (x) = 0 для всіх вершин, то в результаті отримується ще один спеціальний випадок – алгоритм Дейкстри.

Особливості реалізації

Існує також декілька особливостей реалізації і прийомів, які можуть в значній мірі впливати на ефективність алгоритму. Перше, на що слід звернути увагу – це те, як в черзі з пріоритетом обробляються зв'язки між вершинами. Якщо вершини будуть додаватися за принципом «перший прийшов – останній вийшов», то в разі вершин з однаковою оцінкою алгоритм A * піде в глибину. Якщо ж при додаванні вершин буде виконуватися принцип «перший прийшов – перший вийшов», то для вершин з однаковою оцінкою алгоритм буде виконувати пошук в ширину [21]. У багатьох випадках ця обставина може істотно впливати на продуктивність.

Після того, як алгоритм відпрацював і ми отримали маршрут, разом з кожною вершиною зазвичай зберігається посилання на батьківський вузол. За допомогою цих посилань з’являється можливість реконструювання оптимального маршруту. В такому випадку важливо, щоб одна і та ж вершина не зустрічалася в черзі двічі. В більшості випадків для вирішення цієї проблеми перед додаванням відбувається перевірка, чи немає запису про дану вершину в черзі. Якщо така знайдеться, то запис оновлюють так, щоб вона відповідала мінімальній вартості з двох. Для пошуку вершини в кроні дерева багато стандартних алгоритмів потребують часу O (n). Якщо

вдосконалити дерево з використанням хеш – таблиці, то з'явиться можливість зменшити цей час.

Оцінка складності

Складність алгоритму A * залежить від евристики. У гіршому випадку, число вершин, досліджуваних алгоритмом, зростає експоненціально в порівнянні з довжиною оптимального шляху, але складність стає поліноміальною, коли евристика задовольняє наступній умові [20]:

$$\left|h(x) - h^*(x)\right| \leq O(\log h^*(x)) \qquad (1.2)$$

де h * - оптимальна евристика, тобто точна оцінка відстані з вершини x до мети. Іншими словами, помилка h (x) не повинна збільшуватися швидше, ніж логарифм від оптимальної евристики.

Отже, до переваг такого алгоритму можна віднести гарантоване знаходження кращого розв'язку при його наявності та можливість розкриття найменшої кількості вершин на відміну від інших алгоритмів пошуку.

Недоліками даного алгоритму є описана вище тимчасова складність, залежність від точності евристичного алгоритму та надмірне споживання алгоритмом ресурсів пам'яті, при яких з'являється потреба зберігати в ресурсах пам'яті експоненціальну кількість вузлів. Саме по причині занадто великої степені споживання ресурсів даний алгоритм не підходить для системи, яка реалізується.

1.2 Метод гілок і меж

Метод гілок і меж є універсальним алгоритмічним методом для знаходження кращих рішень для різноманітних задач оптимізації, особливо комбінаторної та дискретної задач. Даний алгоритмічний метод є комбінаторним (використовує перебір), тобто дозволяє обходити підмножини допустимих рішень, в яких відсутні оптимальні рішення.

Загальна ідея методу може бути описана на прикладі пошуку мінімуму і максимуму функції f (x) на множині допустимих значень x. Функції f і x можуть бути довільної природи. Для методу гілок і меж виконуються дві процедури: розгалуження і знаходження оцінок (меж) [9].

Процедура розгалуження полягає в розбитті області допустимих рішень на підобласті менших розмірів. Процедуру можна рекурсивно застосовувати до підобласті. Отримані підобласті утворюють дерево, яке називають деревом пошуку або деревом гілок і меж. Вузлами цього дерева є попередньо побудовані підобласті.

Процедура знаходження оцінок полягає в пошуку верхніх і нижніх меж для оптимального значення на підобласті допустимих рішень.

В основі методу гілок і меж лежить наступна ідея (для задачі мінімізації): якщо нижня межа для підобласті A дерева пошуку є більшою, ніж верхня межа якої-небудь раніше переглянутої підобласті B, то A може бути виключена з подальшого розгляду (правило відсіву). Зазвичай мінімальну з отриманих верхніх оцінок зберігають у глобальну змінну m. Будь-який вузол дерева пошуку, нижня межа якого більше значення m, може бути виключений з подальшого розгляду.

Якщо нижня межа для вузла дерева збігається з верхньою межею, то це значення є мінімумом функції і досягається на відповідній підобласті. [2]

Перевагою такого методу є достатньо висока ймовірність отримання хорошого результату пошуку.

Недоліками даного алгоритму є:

- трудомісткість – кількість вузлів у гілці може набувати занадто великих значень;
- відсутність критерію оптимальності – критерій оптимальності буде знайдений тільки після вичерпання списку завдань. Тобто рішення можна отримати ще до зупинки роботи алгоритму, але немає можливості це виявити.

1.3 Метод найближчого сусіда

Алгоритм найближчого сусіда вважається одним з найпростіших евристичних методів рішення задачі комівояжера. Даний алгоритм відносять до категорії жадібних алгоритмів.

Формулюється наступним чином:

Пункти обходу послідовно включаються в маршрут, причому, кожен черговий, що включається пункт повинен бути найближчим до останнього вибраного пункту серед всіх інших, ще не включених до складу маршруту.

Алгоритм простий в реалізації та швидко виконується, але, як і інші жадібні алгоритми, може видавати неоптимальні рішення. Одним з евристичних критеріїв оцінки рішення є правило: якщо шлях, пройдений на останніх кроках алгоритму, порівняти з шляхом, пройденим на початкових кроках, то можна умовно вважати знайдений маршрут прийнятним, інакше, ймовірно, існують більш оптимальні рішення. Інший спосіб оцінки рішення полягає у використанні алгоритму нижньої граничної оцінки (lower bound algorithm).

Переваги методу:

- простота реалізації та отримання результатів;
- в результаті отримується найкраще з можливих рішень.

Недоілки:

- метод допускає такий підбір початкової вершини та значень відстаней між вершинами графу, який спричинить отримання найгіршого рішення з можливих;
- метод використовується в основному для задач з невеликою кількістю змінних та класів;
- неможливість визначення перебігу проходження до кінцевого рішення.

1.4 Алгоритм пошуку в глибину (ширину)

Пошук в глибину (DFS) - це один з відомих методів обходу графа. Алгоритм пошуку описують наступним чином: для кожної непройденої вершини необхідно знайти всі непройдені суміжні вершини і повторити пошук для них. Використовуються як підпрограми в алгоритмах пошуку одно- і двозв'язних компонент, топологічного сортування.

Приклад порядку обходу дерева в глибину наведений на рис. 1.1.

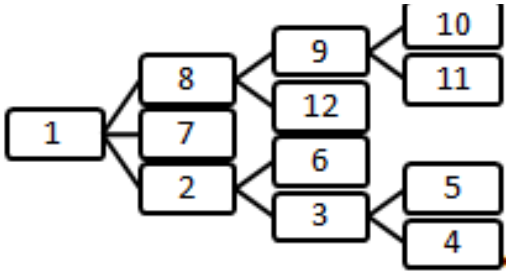

Рисунок 1.1 – Порядок обходу дерева в глибину [11]

Опис

Нехай заданий граф G = (V, E), де V - безліч вершин графа, E - безліч ребер графа. Можна припустити, що в початковий момент часу всі вершини графа пофарбовані в білий колір.

Виконаємо наступні дії:

1) з безлічі всіх білих вершин виберемо будь-яку вершину, позначимо її v1.

2) виконуємо для неї процедуру DFS (v1).

3) перефарбовуємо її в чорний колір.

4) повторюємо кроки 1-3 доти, поки безліч білих вершин не порожньо.

Процедура DFS (параметр – вершина $u \in V$):

1) перефарбувати вершину u в сірий колір.

2) для всякої вершини w, суміжної з вершиною u, виконуємо наступні два кроки:

1) якщо вершина w пофарбована у білий колір, виконуємо процедуру DFS (w).

2) зафарбуємо w в чорний колір.

Недоліком алгоритму є потенційна можливість повторювання кроків та відсутності знайходження цільового вузла.

Пошук в ширину (BFS) - інший метод обходу та розмітки вершин графа. Пошук в ширину виконується в наступному порядку: на початку обходу s приписується мітка 0, суміжним з нею вершинам - мітка 1. Потім по черзі розглядається оточення всіх вершин з мітками 1, і кожної з вхідних в ці оточення вершин приписуємо мітку 2 і т. д. Приклад порядку обходу дерева в глибину представлений на рисунку 1.2.

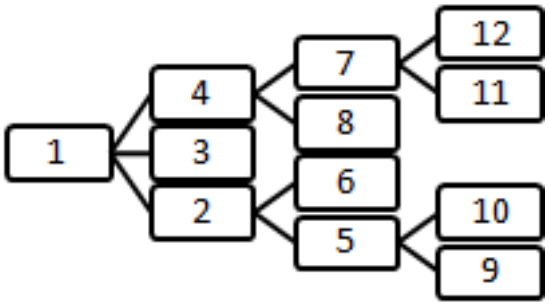

Рисунок 1.2 – Порядок обходу дерева в ширину [11]

Якщо вихідний граф зв'язний, то пошук в ширину позначить всі його вершини. Дуги виду (i, i+1) породжують остовий безконтурний орієнтований граф, який містить в своєму складі остове орієнтоване дерево, зване пошуковим деревом.

Легко побачити, що за допомогою пошуку в ширину можна також виконувати нумерування вершини, нумеруючи спочатку вершини з міткою 1, потім з міткою 2 і т. д.

Двонаправлений пошук в ширину

Алгоритм часто покращує простий пошук в ширину тим, що функціонують два одночасних пошуки в ширину зі стартового і кінцевого вузлів і зупиняється, коли вузол з одного фронту пошуку знаходить сусідній вузол з іншого фронту. Подібний пошук може значно покращити простий пошук в ширину (зазвичай в 2 рази).

Найскладнішим випадком для двонаправленого пошуку є задача, в якій для перевірки мети заданий тільки неявний опис деякої (можливо дуже великої) множини цільових станів. При зворотному пошуку доцільно було б створити компактні описи всіх станів, які дозволяють наприклад поставити мат за допомогою деякої кількості ходів, і ці описи потрібно було б звіряти з станами, сформованими при прямому

пошуку. Загальноприйнятого способу ефективного вирішення такої проблеми не існує, і це є одним з недоліків даного алгоритму.

Найбільш значущим недоліком пошуку в ширину є надмірне використання оперативної пам’яті пристрою, в результаті алгоритм може використати всю пам’ять за кілька хвилин. Це спричинено необхідністю збереження всіх рівнів дерева для створення наступних.

1.5 Алгоритм Дейкстри

Алгоритм Дейкстри – графовий алгоритм, який був винайдений Е. Дейкстрою. Даний алгоритм дозволяє знайти найкоротшу відстань від однієї з вершин графа до всіх інших. При цьому працює тільки для графів без ребер негативної ваги. Алгоритм широко і дуже часто застосовують на практиці, в програмуванні і технологіях. Наприклад, його використовує протокол OSPF для усунення кільцевих маршрутів. Відомий також під назвою найкоротший шлях – перший (Shortest Path First) [30].

Формулювання завдання

Дано простий зважений граф G (V, E) без петель і дуг негативної ваги. Потрібно знайти найкоротші шляхи від деякої вершини графа G до всіх інших вершин цього графа.

Ініціалізація

Кожній вершині з V зіставляється мітка – мінімальна відома відстань від цієї вершини до a. Алгоритм функціонує в покроковому режимі – на кожному кроці він проходить одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані. Мітка самої вершини a встановлюється рівною 0, мітки інших вершин – нескінченності. В цього випливає момент, що відстані від a до інших вершин поки невідомі. Всі вершини графа позначаються як не відвідані.

Крок алгоритму

У випадку, якщо всі вершини відвідані, алгоритм завершується. В іншому випадку із ще не відвіданих вершин вибирається вершина u, яка містить мінімальну

позначку. Розглядаються всілякі маршрути, в яких u є передостаннім пунктом. Вершини, з'єднані з вершиною u ребрами, можна назвати сусідами цієї вершини. Для кожного сусіда розглядається нова довжина шляху, що дорівнює сумі поточної мітки u і довжині ребра, що з’єднує u з цим сусідом. Якщо отримана довжина менше мітки сусіда, проводиться заміна мітки цією довжиною. Після розгляду всіх сусідів, позначається вершина u як відвідану і повторюється крок.

Алгоритм

Позначення:

- V - безліч вершин графа

- E - безліч ребер графа

- w [ij] - вага (довжина) ребра ij

- a - вершина, відстані від якої шукаються

- U - безліч відвіданих вершин

- d [u] - по закінченню роботи алгоритм дорівнює довжині найкоротшого шляху з a до вершини u

- p [u] - по закінченні роботи алгоритм містить найкоротший шлях з a в u

Псевдокод:

привласнимо $d[a] \leftarrow 0,\ p[a] \leftarrow a$

для всіх $u \in V$ відмінних від a

присвоїмо $d[u] \leftarrow \infty$

поки $\exists V \notin U_c d[v] < \infty$

нехай $v \notin U$ — вершина з мінімальним d[v] (3)

додамо вершину v до U

для всіх $u \notin U$ таких, що $uv \in E$

якщо d[u] > d[v] + w[vu] тоді

замінимо $d[u] \leftarrow d[v] + w[vu]$

замінимо $p[u] \leftarrow p[v], u$

Опис.

У найпростішій реалізації для зберігання чисел d [i] можна використовувати масив чисел, а для зберігання даних елемента множині U - масив логічних змінних.

На початку алгоритму відстань для початкової вершини визначається рівною нулю, а всі інші відстані встановлюються великим позитивним числом (більше максимального можливого шляху в графі). Масив прапорів заповнюється нулями. Потім запускається основний цикл [27].

На кожному кроці циклу шукається вершину з мінімальною відстанню і прапором рівним нулю. Потім встановлюється в неї прапор в 1 і перевіряються всі сусідні з нею вершини. Якщо в ній відстань більше, ніж сума відстані до поточної вершини і довжини ребра, тоді його потрібно зменшити. Цикл завершується, коли прапори всіх вершин дорівнюють 1, або коли у всіх вершин з прапором 0 $d[i]=\infty$. Такий випадок має місце у випадку, якщо граф G незв'язаний.

Перевагою алгоритму є облік довжини шляху від вихідної і-тої вершини до наступної j-тої вершини (як це було в алгоритмі найближчого сусіда) та невисока обчислювальна складність.

До основних недоліків можна віднести: складна програмна реалізація в порівнянні з способами, які використовують матриці; неможливість роботи тільки з графами, які містять невід’ємні довжини дуг.

1.6 Алгоритм Джонсона

Алгоритм Джонсона знаходить шляхи за найменшою відстанню між усіма парами вершин зваженого орієнтованого графа. Даний алгоритм може працювати у випадках, коли в графі містяться ребра з позитивною або негативною вагою, але відсутні цикли з негативною вагою.

Алгоритм

Дано граф $G = (V, E)$ з ваговою функцією $\omega: E \to R$

Якщо ваги всіх ребер ω в графі невід'ємні, є можливість знайти найкоротші шляхи між усіма парами вершин, використавши алгоритм Дейкстри один раз для кожної з вершин. Якщо граф містить ребра з негативною вагою, але відсутні цикли з

негативною вагою, тоді можна обчислити нову безліч ребер з невід'ємними вагами, що дозволяє скористатися попереднім методом. Нова безліч, що складається з ваг ребер ω має задовольняти наступним властивостям [6].

Для всіх ребер (u, ν) нова вага ω (u , ν) > 0.

Для всіх пар вершин (u, ν) ∈ V шлях p є найкоротшим шляхом з вершини υ в вершину ν з використанням вагової функції ω тоді і тільки тоді, коли p - також найкоротший шлях з вершини υ в вершину ν з ваговою функцією ω.

Збереження найкоротших шляхів

Нехай дано зважений орієнтований граф G = (V, E) з ваговою функцією ω: E → R , і нехай h: E → R - довільна функція, яка відображає вершини на дійсні числа. Для кожного ребра (u, ν) ∈ E визначимо:

$$\widehat{\omega}(u,v) = \omega(u,v) + h(u) - h(v) \qquad (1.3)$$

Нехай p = < ν0, ν1, ... , νk > - довільній шлях з вершини ν0 в вершину νk p є найкоротшім шляхом з ваговою функцією ω тоді і тільки тоді, коли він є найкоротшім шляхом з ваговою функцією ω, тобто рівність ω (p) = δ (ν0, νk) є рівносильною рівності ω (p) = δ (ν0, νk). Крім того, граф G містить цикл з негативною вагою з використаних вагової функції ω тоді и тільки тоді, коли він містить цикл з негативною вагою з використаних вагової функції ω.

Зміна ваги

Для даного графа створюється новий граф G '= (V ', E '), де V ' = V ∪ { s }, для деякої нової вершини s ∉ V, а E '= E ∪ { (s , υ) : υ ∈ V }.

Виконується розширення вагової функції ω таким чином, щоб для всіх вершин υ ∈ V виконувалась рівність ω (s, υ) = 0.

Далі визначається для всіх вершин υ ∈ V величина h (υ) = δ (s, υ) і нові ваги для всіх ребер ω (u, υ) = ω (u, υ) + h (u) - h (υ) ≥ 0.

Недоліком цього алгоритму є те, що в загальному випадку робота алгоритму може зайняти значний проміжок часу. [3]

1.7 Мурашиний алгоритм

Основна задумка мурашиного алгоритму (методу мурашиних колоній) полягає в моделюванні поведінки колонії мурах, які здатні знаходити найкоротшій шлях від мурашника до джерела їжі і пристосовуватися до мінливих умів зовнішнього середовища, знаходячи новий найкоротшій шлях, якщо попередні умови пересування є вже недоступними. Під час руху мураха залишає феромон, відмічаючи таким чином пройдений шлях, і цю інформацію використовують інші мурахи. В результаті для взаємодії між агентами (мурахами) застосовується непряма комунікація. [2]

Якщо змоделювати переміщення мурах на деякому графі, ребра якого являють собою можливі шляхи переміщення мурах, то крім позитивного зворотного зв'язку (відкладання феромону), існує також негативний зворотній зв'язок, який проявляється в вигляді такого явища як випаровуваності феромону. Присутність негативного зворотного зв'язку гарантує, що мурахи будуть шукати інші шляхи від колонії до джерела їжі, тобто процес знаходження локального оптимального рішення не є єдиним. Рішенням задачі буде найбільший вміст феромону на ребрах графа, який є оптимальним.

Висновки до розділу

В процесі аналізу існуючих алгоритмів знаходження кращого маршруту були визначені переваги та недоліки кожного з методів.

Згідно виконаного аналізу алгоритм A* є не ефективним при наявності кількох локальних рішень з однаковою евристикою, яка відповідна вартості оптимального шляху, однак для попередньо визначеної евристики він є кращим.

Метод гілок і меж застосовується для знаходження оптимальних рішень різних задач оптимізації, особливо дискретної і комбінаторної оптимізації. Але такий метод при кожному використанні працює з відсівом підмножин допустимих рішень, які не містять оптимальних рішень, внаслідок чого ускладнюється перебір та значно збільшується час пошуку рішення.

Метод найближчого сусіда є жадібним алгоритмом, якого відносять до найпростіших евристичних методів рішення задачі комівояжера. Однак похибка результатів у більшості випадків занадто велика, що не допустимо при рішенні більшості практичних задач.

Пошук в ширину та глибину використовують як складові в алгоритмах пошуку одно- і двозв'язних компонент, топологічного сортування, що дуже звужує перелік вирішуваних задач.

Алгоритм Дейкстри знаходить найкоротшу відстань від однієї з вершин графа до всіх інших, але при цьому коректно функціонує тільки для графів позитивної ваги, що також зменшує перелік виконуваних задач.

Алгоритм Джонсона дозволяє знайти найкоротші шляхи між усіма парами вершин зваженого орієнтованого графа, однак не працює при відсутності циклів з негативною вагою та займає значний проміжок часу для знаходження рішення.

Мурашиний алгоритм реалізується по принципу колонії мурах, які здатні знаходити найкоротшій шлях від мурашника до джерела їжі і пристосовуватися до змін стану середовища, знаходячи новий найкоротшій шлях, якщо попередні умови виявилися недоступними. При цьому взаємодія між мурахами відбувається за допомогою непрямої комунікації.

Отже, для формування шляхів в системі для кур’єрських перевезень був вибраний мурашиний алгоритм. Окрім того, що такий метод дозволяє отримувати рішення не гірше переглянутих раніше, основними критеріями є його ефективне функціонування в умовах неперервної зміни параметрів в часі, та можливістю не зациклюватись на локальних оптимальних рішеннях [4].

Зокрема при сучасному інтенсивному дорожньому трафіку є ймовірність створення заторів, тимчасових перекриттів доріг, аварій та інших непередбачуваних ситуацій які впливають на швидкість та якість надаваного кур’єрською компанією сервісу. В процесі неперервного руху по автошляхах такі переваги є доцільними тому, що при стрімкій зміні умов на маршруті, алгоритм дозволить знайти інше найкраще рішення.

2 МУРАШИНИЙ АЛГОРИТМ В СИСТЕМІ ПОБУДОВИ МАРШРУТІВ

2.1 Основні поняття

В даному підрозділі проводиться загальний огляд вибраного алгоритму та механізм його функціонування.

2.1.1 Біологічна основа алгоритму

Як відомо, комахи, включаючи мурах різних видів, термітів і деякі види бджіл і ос, живуть в колоніях, складених з великої кількості індивідуумів, які взаємодіють між собою. Колонії комах дають змогу вирішувати різні завдання оптимізації, які жодна комаха окремо не була б здатна вирішити (наприклад, знаходження найкоротших шляхів у процесі добування харчів, вирішення задачі при призначенні робочої сили і кластеризація при організації гнізд).

Для множини комах потрібна деяка форма взаємозв'язку, яка дозволить співпрацювати при вирішенні повсякденних завдань. Цей зв'язок між індивідуумами колонії може бути прямий, в залежності від різновиду. Коли бджола знаходить джерело продовольства, то вона повідомляє напрямок і відстань до місця розташування, де воно було знайдене іншим бджолам виконуючи характерний танець. Це приклад прямого зв'язку, оскільки інші бджоли повинні відчувати танець, який виконує одна бджола, щоб визначити місцезнаходження джерела продовольства. Інші форми прямого зв'язку: фізичний контакт, обмін продовольством або деякою рідиною.

Непрямий зв'язок між особинами колонії вимагає, щоб представник виду змінив навколишнє середовище таким чином, щоб це змінило поведінку особин, які будуть взаємодіяти з цим зміненим навколишнім середовищем в майбутньому [20].

Одним із прикладів непрямого зв'язку є утворення слідів феромону, які виконують деякі різновиди мурах. Мураха при добуванні харчів позначає доріжку, залишаючи певну кількість феромону на своєму шляху, в результаті чого спонукає слідувати по такому шляху інші особини, які також виконують добування харчів.

Принцип зміни навколишнього середовища в якості засобу зв'язку для зміни поведінки називають stigmergy. Це є основним методом в організації в мурашиних колоніях, який дозволяє їм самоорганізовуватися.

Термін самоорганізація (СО) використовується, щоб описати складну поведінку, яка виникає при взаємодії порівняно простих агентів. За допомогою СО мурашки здатні вирішити складні завдання, з якими вони стикаються щодня. Вигода СО при вирішенні завдань особливо очевидна через її розподілений та ефективний характер. Мурашина колонія може ефективно підтримувати осмислену поведінку, навіть якщо велика кількість мурашок нездатна до сприяння.

Для кращого розуміння механізму та здатності мурашиних колоній сходитися на оптимальних та хороших рішеннях при пошуку найкоротшого шляху від гнізда до джерела продовольства були проведені деякі експерименти [2]. При проведенні таких експериментів колонії мурах з Аргентини Linep - ithema humile давали два шляхи ідентичної довжини, і після того, як минув певний час, було зафіксовано, що мурашки сходилися до однієї з доріжок, після чого фактично були виключені альтернативні шляхи. Окрім того, щоб остаточно перевірити, чи виберуть особини даного виду мурах найбільш короткий з безлічі шляхів, був проведений подвійний експеримент моста, де мурахи були змушені вибирати двічі між коротким і довгим шляхами (рисунок 2.1).

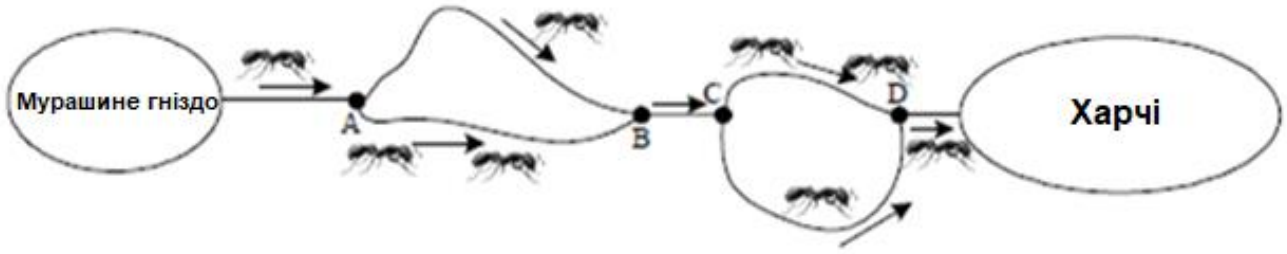

Рисунок 2.1 – Подвійний експеримент моста [12]

Мурахи з Аргентини є фактично сліпими, тому вони не мають ніяких безпосередніх засобів визначення найкоротшого шляху. Однак, незважаючи на такий недолік, як виявилось з результатів експерименту, колонія мурашок здатна до

виявлення найкоротшого шляху, що з'єднує гніздо з ділянкою, яка містить продовольство.

Спочатку, всі мурахи розташовані на ділянці гнізда. Безліч мурашок відправляється від гнізда в пошуку продовольства, кожна особина залишає феромон в процесі проходження шляху, і досягає першого пункту в точці А. Так як мурашки не володіють жодною інформацією про те, який шлях обрати, тобто ніяка інша мураха не проходила тут раніше і не залишила за собою слід феромону, кожна особина буде вибирати йти чи вправо або вліво з однаковою ймовірністю.

Через деякий час приблизно одна половина мурах вибере більш короткий шлях, решта – більш довгий маршрут до перетину з В.

Мурахи, які пройдуть по коротшому шляху, досягнуть цього перетину швидше, і повинні будуть вирішити, який шлях вибрати, щоб повернутися. Тоді в мурах знову виникає ситуація коли немає жодної інформації, щоб використовувати її для орієнтування, тому одна частина мурах, яка досягла перетин В, повернеться до гнізда, в той час як інша буде продовжувати переміщення до ділянки, в якій міститься продовольство.

Мурахи, які опинилися на більш довгій ділянці між перетинами А і В досягають перетину В і також розділяються, однак, оскільки вміст сліду феромону, який знаходиться на шляху назад до гнізда приблизно вдвічі більше, ніж сліду феромону, що досягає області з продовольством, більша частина мурах повернеться до гнізда, прибуваючи туди в той же час як і інші мурахи, які обрали довгий шлях. Слід зазначити, що так як більше мурах тепер йшли по короткій ділянці між перетинами А і В в порівнянні з довгим, наступні мурахи, що залишають гніздо тепер вже будуть надавати перевагу короткому шляху, який є першим вдалим вибором при пошуку найкоротшого шляху [19].

Поведінка мурах на наступній ділянці між пунктами С і D фактично ідентична поведінці, яка була розглянута на першому мосту між перетинами А і В. У кінцевому рахунку, безліч мурашок досягне кінцевого пункту і збере деяку кількість харчів, щоб принести до гнізда.

Досягнувши пункту D, мурахи нададуть перевагу коротшому шляху по тому ж самому принципу, що й мурахи, які скоріше почали переміщення, і така ж сама ситуація відбудеться знову в пункті B.

Так як кількість феромону між пунктами A і C на зворотному шляху до гнізда приблизно дорівнює сумі кількості феромону на цих двох ділянках від гнізда, то це буде найбільш ймовірним найкоротшим зворотнім шляхом.

Оскільки мурахи безперервно розподіляють феромон в процесі свого переміщення, короткий шлях безперервно посилюється все більшою кількістю мурах, поки величина феромону, який позначає шлях не стане наскільки великою, що це затьмарить інші можливі шляхи і фактично всі мурашки будуть використовувати найкоротший шлях, тобто система зійдеться до найкоротшого шляху через самозміцнення [13].

Слід звернути увагу, що феромон, який використовується мурахами, повільно випаровується через деякий час. Дійсно, довжини шляхів, які не були вибрані для проходження протягом деякого часу, є досить великими, і ці шляхи не міститимуть майже ніяких слідів феромону внаслідок їх випаровування після певного проміжку часу, і як результат, збільшується ймовірність вибору мурахами коротких шляхів. Детальніша схема роботи мурашиного алгоритму представлена на рисунку 2.2.

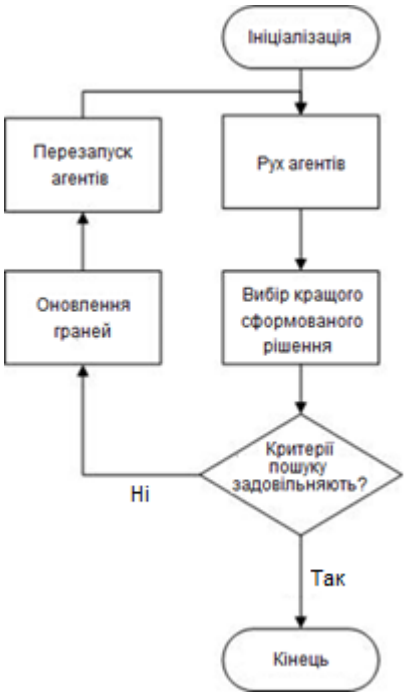

Рисунок 2.2 – Схема роботи мурашиного алгоритму [14]

Одним із перших завдань, для яких застосовувався метод мурашиних колоній (алгоритм мурашки), були задачі комівояжера (Traveling Salesman Problem, TSP). Основною причиною, через яку була обрана дана задача, є те, що в такому типі задач необхідно знаходити найкоротший шлях між пунктами, тому аналогія методу мурашиних колоній легко пристосовується для вирішення даної задачі.

Для вирішення даної задачі було розроблено декілька різних методів, які засновані на оптимізації за допомогою мурашиних колоній.

Першим методом був метод мурашиних систем (Ant System - AS). Надалі такий метод послужив основою для багатьох інших методів, що працюють на принципі мурашиних колоній.

У методі мурашиних систем агент формує своє рішення в процесі переміщення від одного вузла до іншого на графі рішень. Метод працює до виконання t_max ітерацій. На кожній ітерації агенти формують свої рішення за n кроків, на кожному з яких використовується правило вибору наступного вузла – правило вибору агентом, що знаходяться на деякому вузлі наступного вузла для переміщення.

Спочатку Марком Дориго було запропоновано три методи мурашиних систем, різних між собою способом оновлення шляхів – ребер. Ці методи мали наступні назви: щільнісний (ant - density), кількісний (ant - quantity) і циклічний (ant - cycle) методи мурашиних систем. У щільнісному і кількісному методах агенти залишали феромони в процесі формування рішення, в той час як в циклічному методі агенти залишали феромони після закінчення переміщення, тобто після вибору рішення.

Проведені експерименти за рішенням тестових завдань показали, що циклічний метод видавав значно кращі результати в порівнянні з іншими двома. У зв'язку з цим, два менш ефективні методи були відкинуті. Тому, надалі, під методом мурашиних колоній розуміється саме циклічний метод мурашиних систем. [5]

2.1.2 Початкова популяція

Після створення популяції мурахи порівну розподіляються по вузлах мережі. Необхідно порівну розділити мурах між вузлами, щоб всі вузли мали однакові шанси

стати відправною точкою. Якщо всі мурашки почнуть рух з однієї точки, це означатиме, що дана точка є оптимальною для старту, але насправді нам це невідомо.

Для кожної особини перехід з пункту i в пункт j залежить від трьох складових: пам'яті мурахи (TabuList), видимості та віртуального сліду феромону.

TabuList (пам'ять мурахи) - це список, в якому зберігається бітовий масив, за допомогою якого визначаються відвідані і не відвідані вузли. Таким чином, агент повинен проходити через кожен вузол тільки один раз. Вузли в списку поточного проходження (Path) розташовуються в тому порядку, в якому агент їх відвідував. Пізніше список використовується для визначення протяжності шляху між вузлами.

Ясно, що список Tabu зростає при виконанні маршруту та очищується при кожній ітерації алгоритму. Позначимо через J (i, k) список міст, які ще необхідно відвідати мурасі k, що знаходиться в місті i.

Видимість є величиною, зворотною відстані: $\eta_{ij} = 1 / L_{ij}$, де Lij - відстань між пунктами i та j. Видимість – це локальна статична інформація, що виражає необхідність відвідати найближчий пункт з наступних. Використання тільки такої величини як видимість, неодмінно, є недостатньою умовою для знаходження найоптимальнішого маршруту.

Віртуальний слід феромону на ребрі (ij) являє собою підтверджене мурашиним досвідом бажання відвідати пункт j з пункту i. Цей параметр характеризує перевагу вибору даної грані в порівнянні з іншими в процесі переміщення. Інформація про феромони граней змінюється в процесі формування рішень. При цьому кількість феромонів, що генерується агентами, пропорційна якості рішення, складеного відповідним агентом: чим коротший шлях, тим більше буде залишено феромону, і навпаки, чим більша довжина шляху, тим в меншій кількості буде залишено феромону на відповідних ребрах. Такий підхід дозволяє забезпечити безпосередній пошук в процесі знаходження кращого рішення.

На відміну від видимості слід феромону є більш глобальною і динамічною інформацією – вона змінюється після кожної ітерації алгоритму, відображаючи при цьому надбаний мурахами досвід. Кількість віртуального феромону на ребрі (ij) на ітерації t позначимо через $\tau_{ij}(t)$. [6]

2.1.3 Переміщення мурахи

Переміщення мурах ґрунтується на одному ймовірнісно-пропорційному правилі, яке визначає ймовірність переходу k-ї особини з пункту i в пункт j, якщо мураха ще не закінчила свій шлях, тобто не відвідала усі вузли мережі:

$$\begin{cases} P_{ij,k}(t) = \dfrac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{l \in J_{i,k}} [\tau_{ij}(t)]^{б} \cdot [з_{ij}(t)]^{в}} < Rand(1), \text{якщо } j \in J_{i,k}, \\ P_{ij,k}(t) = 0, \text{якщо } j \notin J_{i,k} \end{cases} \quad (2.1)$$

де: P(ij, k) (t) - ймовірність того, що мураха k переміститься з вузла i в вузол j;

Rand (1) - випадкове число в інтервалі (0;1);

б - параметр, що описує вагу сліду феромону при виборі маршруту. При б = 0 буде обрано найближче місто, що відповідає жадібному алгоритму в класичній теорії оптимізації;

в - параметр, що задає вагу видимості. Якщо в = 0, тоді працює лише феромон посилення, що провокує швидке виродження маршрутів одного субоптимального рішення.

Варто звернути увагу, що правило (2.1) визначає лише ймовірність вибору того чи іншого пункту. Власне вибір пункту здійснюється за принципом колеса рулетки: кожен пункт має свій сектор з площею, пропорційною ймовірності (2.1). Для вибору пункту необхідно кинути кульку на рулетку – згенерувати випадкове число, і визначити сектор, на якому ця кулька зупиниться.

Зауважимо, що хоча правило (2.1) не змінюється протягом ітерації, значення ймовірностей P (ij, k) (t) для двох мурах в одному і тому ж пункті можуть відрізнятися, оскільки P (ij, k) (t) - функція від J (i, k) списку ще відвіданих міст мурахою k.

Пройдений мурахою шлях закінчується тоді, коли мураха відвідає всі вузли графа. Слід також мати на увазі, що цикли заборонені, оскільки в алгоритм включений список TabuList. По завершенні довжина шляху може бути обчислена – вона дорівнює сумі всіх граней, по яких переміщувалася мураха. Рівняння (2.2) показує кількість

ферменту, який був залишений на кожній грані шляху для мурахи k. Змінна Q є константою.

$$\Delta\phi_{ij,k}(t) = \frac{Q}{L_k(t)} = \begin{cases} \frac{Q}{L_k(t)}, \text{якщо } (ij) \in T_k(t) \\ 0, \text{якщо } (ij) \notin T_k(t) \end{cases}, \quad (2.2)$$

де Tk (t) – маршрут, пройдений мурахою k на ітерації t;

Lk (t) - довжина цього маршруту;

Q - регульований параметр, значення якого повинне бути одного порядку з довжиною оптимального маршруту.

Результат рівняння можна використати як засіб вимірювання шляху. Короткий шлях характеризується високою концентрацією ферменту, а більш довгий – більш низькою. Потім отриманий результат використовується в рівнянні (2.3), для збільшення кількості ферменту вздовж кожної грані пройденого мурахою k-го шляху.

$$\phi_{ij,k}(t+1) = \phi_{ij}(t) + c \cdot \sum_{k=1}^{N_{ij}} \Delta\phi_{ij}(t), \quad (2.3)$$

де c - коефіцієнт випаровування феромону, з ∈ [0,1];

i та j - вузли, що утворюють грані, які відвідала k-та мураха;

Nij - загальна кількість агентів, які відвідали грань.

Слід звернути увагу, що дане рівняння застосовується для всього шляху, при цьому кожна грань позначається ферментом пропорційно довжині шляху. Тому слід дочекатися, поки мураха закінчить переміщення і тільки потім оновити значення кількості ферменту, в іншому випадку дійсна довжина шляху залишиться невідомою.

На початку шляху у кожній грані є шанс бути обраною. Щоб поступово видалити грані, які входять в гірші шляхи в мережі, до всіх граней застосовується процедура випаровування ферменту (Pheromone evaporation). Використовуючи константу з рівняння (2.3), можна отримати формулу (2.4).

$$\phi_{ij}(t) = \phi_{ij}(t) \cdot (1 - c) \quad (2.4)$$

На початку оптимізації кількість феромону приймається рівною невеликому позитивному числу (initialPheromone). Загальна кількість мурах в колонії залишається постійною на протязі роботи алгоритму. Численна колонія призводить до швидкого посилення субоптимальних маршрутів, а коли мурах невелика кількість, виникає небезпека втрати кооперативності поведінки через обмежену взаємодію і швидке випаровування феромону. Звичайно кількість мурах повинна дорівнювати кількості пунктів, оскільки кожна мураха починає маршрут зі свого міста.

По завершенню проходження мурахою шляху, грані вже оновлені відповідно до довжини шляху і відбулося випаровування ферменту на всіх гранях, алгоритм запускається повторно. Список табу очищується, і довжина шляху онуляється. Мурашкам дозволяється переміщуватися по мережі, вибираючи грань відповідно до рівняння (2.1).

Такий процес може виконуватися для незмінної кількості шляхів або до моменту, коли протягом декількох запусків не було відзначено повторних змін. Потім визначається кращий шлях, який і являє собою рішення. [7]

2.1.4 Приклад ітерації

Розберемо функціонування алгоритму на простому прикладі, щоб побачити, як працюють описані раніше рівняння. Розглянемо простий сценарій з двома мурашками, які обирають два різних шляхи для досягнення однієї мети. На рис. 2.3 показано такий приклад з двома гранями між двома вузлами (V0 та V1).

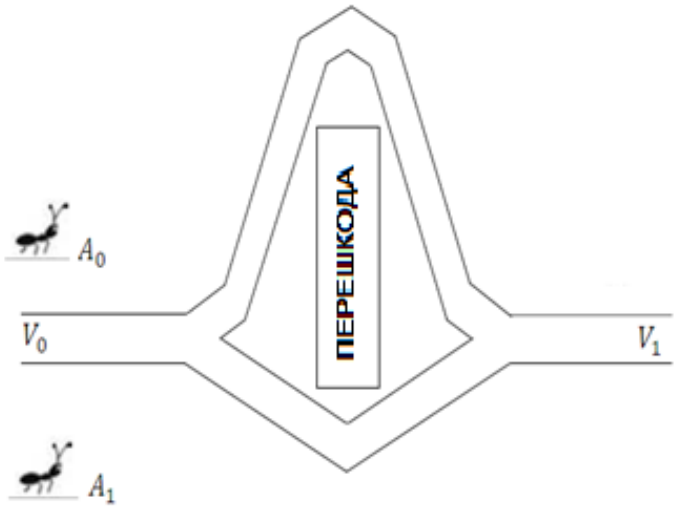

Рисунок 2.3 – Початковий стан алгоритму [22]

Кожна грань ініціалізується і має однакові шанси щоб бути обраною. Короткий шлях становить 10 кроків, довгий шлях складає 20 кроків.

В якості параметрів алгоритму визначимо змінні:

- вага сліду феромону, б = 3.0;
- вага видимості, в = 1.0;
- коефіцієнт випаровування феромону, з = 0.6;
- регульований параметр Q = 10;
- початкове значення феромона initialPheromone = 0.1.

Дві мурахи знаходяться у вузлі V0 і позначаються як A0 і A1. Так як ймовірність вибору будь-якого шляху однакова, в цьому циклі ми проігноруємо рівняння вибору шляху. На рисунку 2.4 кожна мураха вибирає собі шлях (мураха A0 іде по верхньому шляху, а мураха A1 – по нижньому).

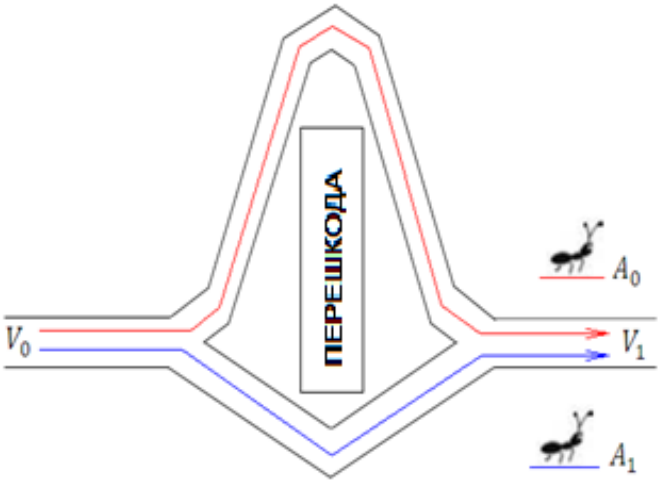

Рисунок 2.4 – Вибір шляху [22]

Відповідно до рівняння (2.2) обчислюється кількість ферменту, що буде залишений:

$$\Delta\phi_{V_0 V_1,0}(t) = \frac{10}{20} = 0.5$$

$$\Delta\phi_{V_0 V_1,1}(t) = \frac{10}{10} = 1$$

Далі за рівнянням (2.3) розраховується кількість ферменту, що буде застосовано. Для мурах A0 і A1 результат відповідно становить:

$$\phi_{V_0 V_1,0}(t+1) = 0.1 + 0.5 \cdot 0.6 = 0.4$$
$$\phi_{V_0 V_1,1}(t+1) = 0.1 + 1.0 \cdot 0.6 = 0.7$$

Далі за допомогою рівняння (2.4) визначається, яка частина ферменту випарується і, відповідно, скільки залишиться. Результати (для кожного шляху) складають:

$$\phi_{V_0 V_1,0}(t) = 0.4 \cdot (1.0 - 0.6) = 0.16$$
$$\phi_{V_0 V_1,1}(t) = 0.7 \cdot (1.0 - 7) = 0.28$$

Такі значення представляють нову кількість ферменту для кожного шляху (верхнього і нижнього, відповідно). Тепер перемістимо мурах назад у вузол V0 і скористаємося ймовірнісним рівнянням вибору шляху (2.1), щоб визначити, який шлях повинні вибрати мурахи.

Ймовірність того, що мураха вибере верхній шлях (представлений кількістю ферменту 0.16, становить:

$$P_{V_0 V_1,0}(t) = \frac{[0.16]^3 \cdot [0.05]^1}{[0.16]^3 \cdot [0.05]^1 + [0.28]^3 \cdot [0.1]^1} = 0.085.$$

Ймовірність того, що мураха вибере нижній шлях (який представлений кількістю ферменту 0.28), становить:

$$P_{V_0 V_1,1}(t) = \frac{[0.28]^3 \cdot [0.1]^1}{[0.16]^3 \cdot [0.05]^1 + [0.28]^3 \cdot [0.1]^1} = 0.915.$$

При зіставленні двох ймовірностей обидві мурахи виберуть нижній шлях, який є найбільш оптимальним.[8]

У таблиці 2.1 вказані основні параметри алгоритму, що характеризують даний приклад.

Таблиця 2.1 – Характеристики пройденого шляху

	A_0	A_1
Пройдена відстань, $L_{V_0 V_1}$	20	10
Рівень ферменту / пройдена відстань, $\Delta\phi_{V_0 V_1}$	0.5	1.0
Кількість ферменту, який було застосовано, $\phi_{V_0 V_1}$	0.4	0.7
Кількість ферменту, який випарувався, $\phi_{V_0 V_1}$	0.16	0.28
Імовірність вибору верхнього / нижнього шляху	0.085/0.915	0.085/0.915

2.1.5 Різновиди мурашиного алгоритму

Поведінка мурах може мати різний математичний опис, тому були створені розширення алгоритму мурахи (званого також методом мурашиних систем). Серед таких виділяють: метод, який заснований на ранжуванні (ASrank); метод мурашиних систем, який використовує елітну стратегію; метод системи мурашиних колоній; максимальний метод мурашиних систем (MAX - MIN AS - MMAS).

Найпершим розширенням методу мурашиних систем була елітна стратегія. Такий підхід ґрунтується на збільшенні концентрації феромонів, що дозволить знайти кращий глобальний шлях в деякий момент часу t. За таких обставин процес додавання феромону для дуг, які містять на даний момент часу шлях, проводиться ще раз. Концентрація феромону при таких умовах обчислюється у відповідності до довжини оптимального шляху.

Пізніше з'явився метод мурашиних систем, який заснований на ранжуванні (ASrank). Такий метод є вдосконаленням елітної стратегії і полягає в наступному: агенти упорядковані відповідно до довжини сформованих ними ж шляхів, в результаті чого на глобально кращому шляху кількість феромонів підвищується з деякою вагою w, також збільшення феромонів виробляється тільки для дуг, що були вибрані кращими агентами (w - 1). При цьому k-й кращий агент буде додавати феромон з вагою (wk) відповідно до (2.5):

$$\phi_{ij}(t+1) = c \cdot \phi_{ij}(t) + w \cdot \Delta\phi_{ij,gl}(t) + \sum_{k=1}^{w-1}(k-w) \cdot \Delta\phi_{ij,k}(t), \qquad (2.5)$$

де $\Delta\phi_{ij,gl}(t) = \frac{1}{L_g}(t), L_g$ – довжина кращого глобального шляху.

Метод системи мурашиних колоній (Ant Colony System) дає змогу покращити мурашиний алгоритм шляхом використання інформації, яка була отримана мурахами, які вже виконали обхід, для дослідження простору пошуку. Це досягається за допомогою двох механізмів. По-перше, використовується сувора елітна стратегія при оновленні феромонів на гранях. По-друге, агенти вибирають наступне місто для

переміщення, використовуючи так зване, псевдовипадкове пропорційне правило: з ймовірністю q0 агент переміщується в пункт u, для якого добуток кількості феромонів і евристичної інформації є максимальним (2.6):

$$u = \arg max_{u \in J^i} \left\{ \phi_{ij} \cdot з_{ij}(t)^{\epsilon} \right\} ds \qquad (2.6)$$

в той час як з імовірністю 1 - q0 буде застосований базовий підхід при визначенні наступного пункту для переходу. Значення q0 є константою. При цьому якщо q0 прагне до 1, то використовується тільки псевдовипадкове пропорційне правило, коли ж q0 = 0, тоді метод мурашиних колоній працює за принципом методу мурашиних систем.

В процесі оновленні шляхів, як було сказано раніше, застосовується сувора елітна стратегія, відповідно до якої тільки агент, що склав оптимальніше рішення, виробляє феромон на шляху свого переміщення. Тоді кількість феромонів на гранях змінюється відповідно до формули 2.7:

$$\phi_{ij}(t+1) = c \cdot \phi_{ij}(t) + (1-c) \cdot \Delta\phi_{ij,best}(t) \qquad (2.7)$$

В якості кращого агента може використовуватися агент, який отримав краще рішення в ході виконання поточної ітерації, або глобально кращий агент, який отримав найкраще рішення на всіх ітераціях від початку роботи методу.

Ще однією перевагою методу системи мурашиних колоній є те, що агенти оновлюють кількість феромонів в процесі складання рішення (подібно до щільнісного та кількісного методів мурашиних систем). Такий підхід призводить до зменшення ймовірності вибору однакових шляхів усіма агентами. За рахунок цього знижується ймовірність зациклення в локальному оптимумі.

Максимальний метод мурашиних систем (MAX - MIN AS - MMAS) дозволяє ввести нижню і верхню межу для можливих значень феромонів на межі, а також даний метод відрізняється підходом до опису їхнього значення при виконанні

початкової ініціалізації. Для MMAS застосовують перелік значень феромонів, обмежений ϕ_{min} та $\phi_{max}, \forall \phi_{ij}$, тобто $\phi_{min} \leq \phi_{ij} \leq \phi_{max}$.

У загальному вигляді відмінності між різновидами методу мурашиних колоній можна відобразити в таблиці 2.2.

Таблиця 2.2 – Відмінності між різновидами методу мурашиних колоній

Критерій	AS	ASrank	ACS	MMAS
Додавання феромону	Після отримання рішення	Після отримання рішення	У процесі отримання рішення	У процесі отримання рішення
Динамічна зміна коефіцієнта Q	Відсутня	Відсутня	Відсутня	Відсутня
Правило вибору наступного пункту	Традиційний підхід	Псевдовипадкове пропорційне правило	Традиційний підхід	Традиційний підхід
Застосування елітної стратегії	Всі агенти беруть участь в оновленні шляхів	Оновлення виконують (w-1) локально кращих агентів і глобально кращий агент	Оновлення виконується тільки кращим агентом	
Застосування обмежень для різних параметрів	Відсутні	Обмеження кількості мурах, що виконують оновлення шляхів	Відсутня	Використовується інтервал значень феромонів
Застосування локальної оптимізації	Відсутні	Відсутні	Використовуються традиційні методи локальної оптимізації	Відсутня

Продовження таблиці 2.2

Завдання, які вирішуються	Завдання комівояжера, квадратична задача про призначення, транспортна задача	Завдання комівояжера	Завдання комівояжера, задача календарного планування	Завдання комівояжера, квадратична задача про призначення
Вплив кількості агентів на знаходження результату	Сильне	Середнє	Слабке	Слабке

Кількість феромонів при ініціалізації на гранях задається рівним нижній межі інтервалу, що забезпечує краще дослідження простору рішень. У MMAS, також як і в ACS, тільки кращий агент виконує додавання феромонів після кожної ітерації методу. На основі результатів обчислень можна побачити, що кращі результати виходять в тому випадку, якщо оновлення феромонів виконується з використанням глобально кращого рішення. Для MMAS є можливість використати локальний пошук, що дозволить покращити його характеристики. [9]

2.2 Побудова маршрутів з використанням мурашиного методу

Розглянемо процес використання мурашиного алгоритму для побудови маршрутів перевезень.

2.2.1 Структуризація та алгоритм реалізації методу мурашки

Мурахи є соціальними істотами, які живуть в колективі – колонії. Кількість мурах в одній колонії може досягати декількох мільйонів, а сама колонія може бути розподілена по території в десятки і сотні кілометрів. При тому, в колонії не має централізованого управління, а обмін інформацією між особинами здійснюється з

використанням непрямого обміну, завдяки феромону, який мурахи залишають у процесі свого пересування в просторі [40].

З одного боку, чим більше мурах проходить по деякому шляху, тим більша на ньому концентрація феромону. З іншого боку, при виборі одного шляху з кількох альтернативних, найбільш висока ймовірність вибору мурахою того шляху, концентрація феромону на якому вище.

Описані принципи життєдіяльності колонії мурах лежать в основі розглянутого методу. Сам метод і реалізація цього алгоритму можуть бути ефективно використані при вирішенні завдання комівояжера. По відношенню до методу, комівояжер – це мураха, якій необхідно за найкоротшим маршрутом відвідати всі мурашники, жодного разу не повернувшись в той, де вона вже була.

Етапи алгоритму

1. введення вихідних даних і ініціалізація параметрів налаштовування та змінних алгоритму.

2. визначення ймовірності переходу із одного мурашника в інший.

3. генерація випадкового маршруту переміщення кожної мурахи з останнього місцезнаходження з урахуванням розрахованих в п. 2 ймовірностей переходу; визначення результуючої довжини кожного маршруту (критерій оптимізації); збереження в пам'яті варіантів маршрутів з найкращим значенням критерію.

4. розрахунок зміни концентрації феромону між двома мурашниками і нового значення концентрації феромону.

5. повторення процедури з п. 2 до виконання одного або декількох обраних умов закінчення.

6. виведення варіантів маршрутів, що забезпечують оптимальне значення критерію оптимальності.

Опишемо етапи докладніше.

Введення вихідних даних і ініціалізація параметрів настроювання і змінних алгоритму.

Вихідними даними є:

- кількість мурашників (M);

- кількість мурах, що припадають на один мурашник (n);
- координати мурашників X_i, Y_i.

Параметрами налаштування алгоритму є:

- ступінь впливу феромону (α);
- ступінь впливу відстані (β);
- коефіцієнт, що враховує випаровування феромону (φ).

Змінними алгоритму є:

- загальна кількість мурах (N) [41]:

$$N = nM \tag{2.8}$$

- відстань між мурашниками (D ij):

$$D_{ij} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2} \tag{2.9}$$

- середня відстань між мурашниками (D):

$$D = \frac{2\sum_{i=1}^{M-1}\sum_{j=i+1}^{M} D_{ij}}{M(M-1)} \tag{2.10}$$

- параметр видимості j-го мурашника з і-го мурашника (ν ij):

$$\nu_{ij} = \frac{D}{D_{ij}} \tag{2.11}$$

- кількість феромону на шляху з і-го в j-й мурашник:

$$F_{ij} = 1 \quad \forall i = \overline{1, M-1};\ j = \overline{i, M} \tag{2.12}$$

Визначення ймовірності переходу із одного мурашника в інший

Ймовірність переходу з мурашника і в мурашник j залежить від кількості феромону, що знаходиться на шляху між цими мурашниками, і видимості мурашника (визначається запахом, що доноситься з самого мурашника), в який відбувається перехід, з урахуванням ступенів впливу феромону і відстані між мурашниками [40]:

$$P_{ij} = \frac{F_{ij}^{\ \alpha} \cdot v_{ij}^{\ \beta}}{\sum\limits_{\substack{k=1 \\ k \neq i}}^{M} F_{ik}^{\ \alpha} \cdot v_{ik}^{\ \beta}} \tag{2.13}$$

Оскільки на першій ітерації розрахунку прийнято одиничне значення феромона, однакове для всіх варіантів переходу, на ній зазначимо тільки відстань між містами.

Генерація випадкового маршруту переміщення мурах

Для кожної N-ї мурахи генерується маршрут руху з останнього місцезнаходження випадковим чином, з урахуванням ймовірностей переходу, отриманих на етапі 2. Для кожного з отриманих маршрутів розраховується цільова функція загальної протяжності маршруту.

Розрахунок зміни концентрації феромону.

Зміна концентрації феромону на кожній ділянці залежить від протяжності цієї ділянки. У загальному випадку, чим коротше ділянка, тим більша кількість мурах встигнуть пройти по ній за один і той же проміжок часу, і на ній залишиться більша кількість феромону, в порівнянні з ділянкою, яка має більшу довжину.

Разом з тим, необхідно враховувати випаровування феромону на ділянці з плином часу. Таким чином, якщо за одну ітерацію роботи алгоритму ділянка не використовувалася, то кількість феромону на ній зменшиться.

Зміна концентрації феромону за одну ітерацію обчислюється за співвідношенням:

$$F_{ij}^{(t+1)} = \varphi F_{ij}^{(t)} + D(M-1)(1-\varphi)\sum_{q=1}^{N_q} \frac{1}{W_{ij\,q}^{(t)}} \qquad (2.14)$$

де $W_{ij\,q}^{(t)}$ - протяжності маршрутів, в які входить відрізок шляху між мурашниками i, j на етапі розрахунку t;

q - індекс маршруту, що містить відрізок шляху між мурашниками i, j;

N_q - кількість маршрутів, що містять відрізок шляху між мурашниками i, j.

Умови закінчення обчислювальної процедури і виведення результатів:

1) виконання заданої граничної кількості ітерацій розрахунку;

2) відсутність поліпшення цільової функції на заданій граничній кількості ітерацій розрахунку поспіль.

Рішенням або рішеннями завдання є такі маршрути, отримані за весь час роботи алгоритму, які забезпечують мінімум цільової функції.

2.2.2 Побудова алгоритму пошуку

Розглянемо рішення детермінованого завдання оптимізації. Детерміноване завдання оптимального рішення формулюється наступним чином:

$$\min_{S \in D \subset R^n} F(S) = F(S^*),$$
$$D = \left\{ S \middle| s_j^- \le s_j \le s_j^+, j \in [1:n] \right\}, \qquad (2.15)$$

де R^n - n-мірний простір;

D - область допустимих значень вектора варійованих параметрів;

S - n-мірний вектор варійованих параметрів;

S * - оптимальне значення вектора змінних параметрів;

F (S) - цільова функція (критерій оптимальності);

F (S *) - оптимальне значення критерію оптимальності [37].

Алгоритм ACOR (Ant Colony Optimization for continuous domain) успішно замінює дискретний алгоритм ACO без необхідності вносити будь які зміни в його схему. Інші алгоритми безперервної оптимізації хоч і засновані на алгоритмі ACO (continuous-ACO, API, алгоритм безперервно взаємодіючих колоній мурашок (CIAC)), але в більшості своїй не повторюють його схему.

Основна ідея ACOR – приріст компонентів вектора варійованих параметрів, отриманого на залежному (від феромонів) ймовірному виборі компонентів. Це досягається шляхом заміни дискретного розподілу ймовірностей неперервною функцією, яка називається функцією щільності ймовірностей (Probability Density Function (PDF)), також відомою як розподіл Гауса.

Алгоритм ACOR використовує ядро Гауса (Gaussian Kernel) для зваженого підсумовування декількох функцій щільності імовірності. Ядро Гауса - G^i (S) визначається за формулою [9]

$$G^i(S) = \sum_{l=1}^{k} \omega_l g_l(S) = \sum_{l=1}^{k} \omega_l \frac{1}{\sigma_l^i \sqrt{2\pi}} e^{-\frac{\left(s_l^i - \mu_l^i\right)^2}{2\sigma_l^{i\,2}}}, \qquad (2.16)$$

де k - число функцій щільності ймовірності;

i – вимірювання;

ω_l - вагова функція;

μ_l^i - вектор середніх значень (вектор математичних очікувань);

σ_l^i - вектор дисперсій.

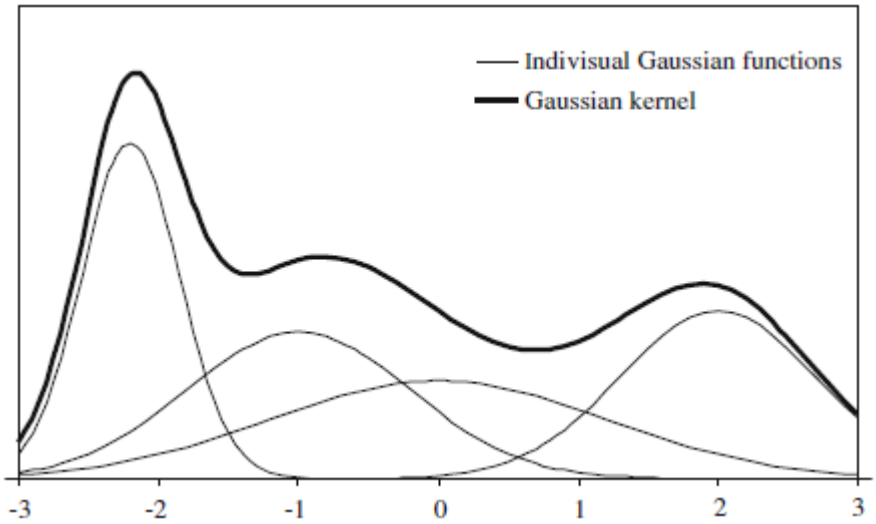

Рисунок 2.5 – Приклад ядра Гауса, що складається з чотирьох функцій Гауса [34]

Модель феромонів ACOR визначається через ранжирування архіву рішень T. На кожній ітерації, отриманий набір рішень додається до набору рішень T і впорядковується за критерієм оптимальності. В наборі рішень T завжди знаходиться k рішень, в результаті чого на кожній ітерації безліч найгірших рішень повинні бути вилучені. Це процедура імітує процес оновлення феромонів в дискретних алгоритмах ACO. Мета цього процесу полягає в тому, щоб змістити процес пошуку в сторону кращих рішень, знайдених при оптимізації.

Схема алгоритму

Для зручного відображення рішень використовують масив рішень, представлений в таблиці 2.3. У таблиці рішень, кожне рішення зберігаються згідно рангу (s_1^i - найкраще), ω_l - вага кожної функції щільності ймовірності, $\omega_1 \geq \omega_2 \geq \cdots \geq \omega_n$, ядро Гауса для i-го кроку - G^i (S), яке обчислюється, використовуючи тільки i-ий компонент всіх k рішень в архіві T.

Таблиця 2.3 – Таблиця рішень [34]

s_1^1	s_1^2	...	s_1^i	...	s_1^n	ω_1
⋮	⋮	...	⋮	...	⋮	⋮
s_l^1	s_l^2	...	s_l^i	...	s_l^n	ω_l
⋮	⋮	...	⋮	...	⋮	⋮
s_k^1	s_k^2	...	s_k^i	...	s_k^n	ω_k
G^1	G^2	...	G^i	...	G^n	

Для отримання рішення, мураха на кожному кроці i = 1, ..., n, вибирає значення рішення S^i в n-мірній задачі оптимізації:

1) для k-мурах випадковим чином отримують рішення s_1, ..., s_n;

2) впорядковується за значенням цільової функції, де ранг рішення l = 1 є кращим;

3) обчислюється вага ω_l для кожного рішення

$$\omega_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, l \in [1:k], \tag{2.17}$$

де q - коефіцієнт.

При фіксованому k, мале значення q (~0) означає, що тільки функція щільності ймовірності кращого рішення буде використана для створення нового рішення, тоді як при великому значенні q, виходить більш рівномірна ймовірність. При великому значенні q, зменшується швидкість отримання кінцевого результату.

4) обчислюється ймовірність кожного рішення

$$p_l = \frac{\omega_l}{\sum_{r=1}^{k} \omega_r}, l \in [1:k]. \quad (2.18)$$

5) методом рулетки, випадково вибирають одне рішення Sl, використовуючи розраховану ймовірність;

6) вважають, що математичне сподівання μ_l^i одно s_l^i;

7) обчислюють дисперсію (відхилення від s_l^i) в і-му вимірі за формулою 2.19;

$$\sigma_l^i = \xi \sum_{e=1}^{k} \frac{|s_e^i - s_l^i|}{k-1}, \quad (2.19)$$

де i [1: n];

ξ - коефіцієнт, який визначає випаровування феромонів (ξ> 0).

8) отримують рішення, використовуючи генератор випадкових чисел і розподіл ймовірностей, отриманих за допомогою ядра Гауса;

9) обчислюють значення цільової функції кожного рішення;

10) додають отримані рішення в архів рішень T;

11) впорядковують отримані рішення;

12) зберігають k рішення в архіві T;

13) якщо краще рішення задовольняє критеріям пошуку, завершують пошук, інакше переходять на третій крок.

2.2.3 Порівняння вдосконаленого алгоритму з аналогом

Мурашиний алгоритм розглядається як багатоагентна система, в якій кожен агент (мураха) функціонує автономно відповідно до певних правил.

На противагу до майже примітивної поведінки агентів, поведінка всієї системи виходить досить розумною [1]. Прикладом підтвердження оптимальності поведінки

мурашиних колоній служить той факт, що мережа їх мурашника близька до мінімального кістяка графа.

Реалізація для пошуку на графі, основне правило поведінки мурашок можна описати наступною формулою [5]:

$$P_{ij}(t) = \frac{\tau_{ij}(t)^{\alpha} \cdot \left(\frac{1}{d_{ij}}\right)^{\beta}}{\sum \tau_{ij}(t)^{\alpha} \cdot \left(\frac{1}{d_{ij}}\right)^{\beta}} \qquad (2.20)$$

де $P_{ij}(T)$ - імовірність переходу з вершини *i* в вершину *j* на теперішній момент часу;

τ_{ij} *(T)* - концентрація феромону (сліду, залишеного мурахами), яка визначає перевагу руху по даному шляху;

d_{ij} - його довжина, а α і β - деякі константи, що визначають роботу алгоритму, що задаються користувачем.

На кожній ітерації алгоритму, зміни концентрації феромону відбуваються відповідно до наступної формули [14]

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \frac{Q}{L_k(t)} \qquad (2.21)$$

де *Q* - параметр, що містить значення порядку оброблюваних довжин;

L_k – Довжина маршруту; *p* - коефіцієнт випаровування феромону.

Суть цієї формули полягає в тому, що кожного разу, коли мураха проходить по деякому відрізку, то залишає слід феромону в розмірі, обернено пропорційному довжині поточного відрізку. Чим більшу кількість разів мурахи пройдуть по деякого шляху, тим вищою буде концентрація феромону, і в результаті з більшою ймовірністю вони будуть знову вибирати такий шлях.

Покращити ефективність роботи алгоритму можна шляхом введення так званого експортування елітних мурах, які будуть підсилювати ребра кращого поточного маршруту, знайденого з початку роботи алгоритму. Тоді якщо в колонії є *e* елітних мурах, то ребра маршруту будуть отримувати додаткову кількість феромону за наведеною нижче формулою

$$\tau_{ij}(t) = \tau_{ij}(t) + e\frac{Q}{L} \qquad (2.22)$$

де *e* - кількість елітних мурах;

L - довжина найкращого поточного маршруту.

Тобто після кожної ітерації збільшується концентрація феромону на тих ребрах, на яких прокладений найефективніший маршрут. Таким чином, прискорюється збіжність до оптимуму внаслідок збільшення концентрації феромону на окремо взятих ребрах. За результатами проведених досліджень, найбільш оптимальною кількістю елітних мурах є 3, дане число поєднує в собі баланс швидкості збіжності та точності одержуваних рішень. При e = 1 точність найбільш висока, а при збільшенні цього значення вимагається все менша кількість ітерацій для отримання локального глухого кута, коли подальша робота алгоритму не приводить до поліпшення результату [12].

Підібрати оптимальні значення параметрів α і β із формули (2.20) можна експериментальний шляхом. Для вищеописаного алгоритму було проведено дослідження впливу цих параметрів на ефективність отриманих рішень. Для довільно згенерованих 80 вершин графа в табл. 2.4 відображені отримані значення, і кращі поєднання виділені кольором.

Таблиця 2.4 – Залежність ефективності рішення від вибраних параметрів на прикладі задачі з 80 точками

Параметри	Краще рішення	Гірше рішення	Середній результат
α = 1, β = 1			
100 ітерацій	6501	7384	6999
1000 ітерацій	4126	4996	4649
α = 2, β = 1			
100 ітерацій	3657	4675	4289
1000 ітерацій	3506	3814	3708
α = 5, β = 1			
100 ітерацій	4321	4909	4556
1000 ітерацій	4249	4337	4296
α = 0, β = 1			
100 ітерацій	10097	11188	10657
1000 ітерацій	9857	10481	10185
α = 1, β = 2			
100 ітерацій	4015	4464	4219
1000 ітерацій	3569	3762	3653
α = 1, β = 5			
100 ітерацій	3509	3582	3554
1000 ітерацій	3471	3493	3482
α = 1, β = 0			
100 ітерацій	15551	15578	15563
1000 ітерацій	12750	13402	13152

В таблиці можна побачити, що зміна значень параметрів α і β призводить до істотних змін в отриманих рішеннях. Кращі результати були отримані при використанні наступних значень параметрів: (A = 2, β = 1) і (α = 1, β = 5).

Збільшення значення першого параметра (збільшення залежності одержуваного рішення від концентрації феромону) призводить до збільшення розкиду рішень, в результаті чого робота мурашиного алгоритму носить скоріше евристичний характер. Збільшення значення другого параметра (збільшення залежності одержуваного рішення від довжини) призводить до зменшення розкиду рішень, і алгоритм починає працювати, насамперед, як точний метод. Далі проводиться дослідження роботи зазначених варіантів алгоритму. Для зручності умовно назвемо МА з ($\alpha = 2$, $\beta = 1$) евристичним мурашиним алгоритмом (ЕМА), а МА з ($\alpha = 1$ і $\beta = 5$) точним мурашиним алгоритмом (ТМА).

Для МА були розроблені наступні модифікації:

1) попередній розрахунок початкової концентрації феромону (РК);

2) розрахунок початкового шляху методом найменших відрізків (МНВ);

3) згладжування методом гілок і границь (МГГ).

Під першою модифікацією мається на увазі, що початкова концентрація феромону розраховується з матриці відстаней ще до роботи основного циклу. Ця модифікація подібна до роботи великої кількості елітних мурах, коли підсилюється швидкість збіжності, але падає точність обчислень. Друга модифікація припускає, що початкове наближення шляху розраховується методом найменших відрізків, який завдяки елітним мурашкам стане пріоритетним на початкових етапах ітерацій, тобто також зросте швидкість збіжності в рази, але впаде точність. Третя модифікація вступає в силу після застосування елітних мурах на кожній ітерації, але тільки в другій половині основного циклу алгоритму, для того щоб не витрачати часові ресурси на пошук явно неефективних рішень. Згладжування ведеться методом гілок і меж для кожної з п'яти точок на всьому шляху, тобто дана модифікація дозволяє прискорити не тільки процес збіжності, а й підвищити точність знайденого рішення [11].

Для порівняльного аналізу запропонованих модифікацій на основі отриманих ТМА і ЕМА потрібно побудувати таблицю для тих же 80 точок, що і в табл. 2.4, і також додати результати роботи для 30 і 150 вершин з кількістю ітерацій 1000.

Таким чином, можна зробити висновки, що модифікації, такі як РК і МНВ, доцільно використовувати для швидкого отримання ефективного вирішення, так як збіжність в даному випадку дуже висока і необхідність у великій кількості ітерацій відпадає, однак імовірність отримання оптимального рішення для великої кількості точок мінімальна.

Максимальну ефективність дані модифікації показують при використанні мурашиного алгоритму з $\alpha > \beta$, так як в ТМА і так вже використовуються точні обчислення завдяки великому значенню β (довжина відрізків значною мірою впливає на ймовірність переходу мурахи з однієї вершини в іншу) і використання інших традиційних алгоритмів тільки погіршує його роботу. Модифікація методом гілок і меж завжди покращує роботу будь-якого МА з незначною втратою швидкості роботи, тому оптимальні рішення були отримані саме з цією модифікацією в ТМА (вони виділені в табл. 2.5).

Таблиця 2.5 – Порівняльний аналіз модифікованого і не модифікованого МА

Назва	30 точок		80 точок		150 точок	
	Краще рішення	Гірше рішення	Краще рішення	Гірше рішення	Краще рішення	Гірше рішення
ТМА	2236	2236	3471	3493	4820	4930
ЕМА	2271	2449	3506	3814	5479	5885
ТМА з РК	2236	2236	3483	3532	5043	5220
ЕМА з РК	2236	2236	3488	3634	5203	5244
ТМА з МНВ	2236	2236	3499	3539	4949	5011
ЕМА з МНВ	2253	2367	3634	3736	5153	5280
ТМА з МГГ	2236	2236	3469	3489	4808	4911
ЕМА з МГГ	2243	2312	3499	3812	5649	5967
ТМА з усіма модифікаціями	2236	2236	3475	3486	5037	5061

Продовження таблиці 2.5

ЕМА з усіма модифікаціями	2236	2236	3485	3642	5146	5251

Результати роботи (довжини знайдених шляхів) досліджуваних алгоритмів для 100-140 точок відображені на рис. 2.6.

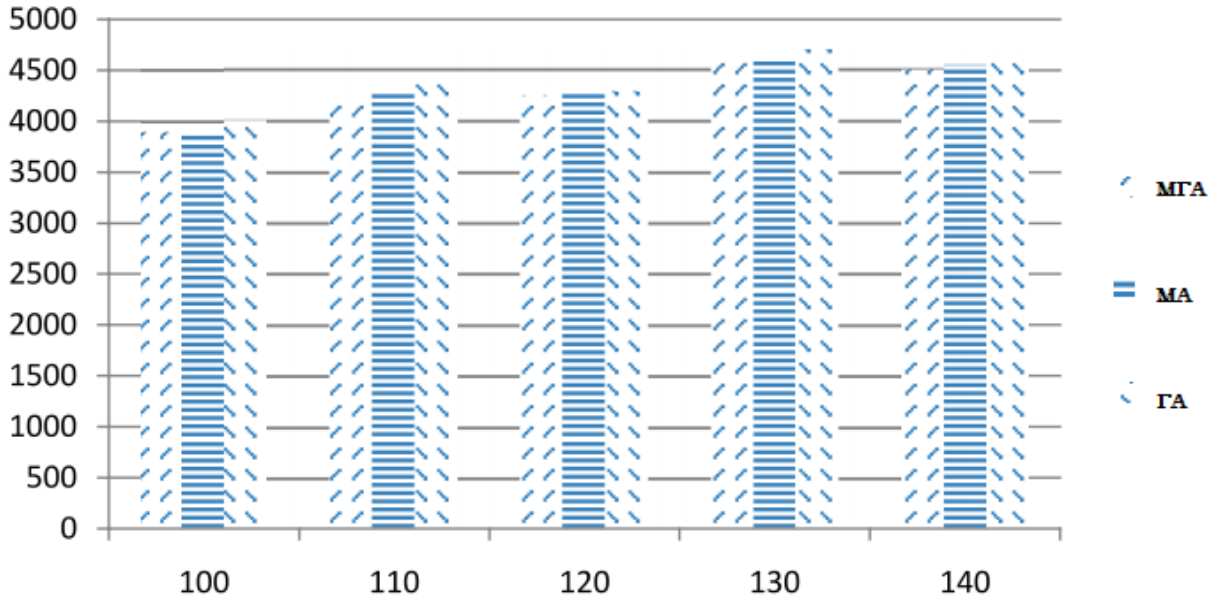

Рисунок 2.6 – Порівняльний аналіз отриманих рішень для 100-140 точок

На рис. 2.6 видно, що до 140 точок алгоритм незначно поступається іншим алгоритмам в точності знайденого рішення.

На рис. 2.7 відображені результати роботи алгоритмів для 300-340 точок.

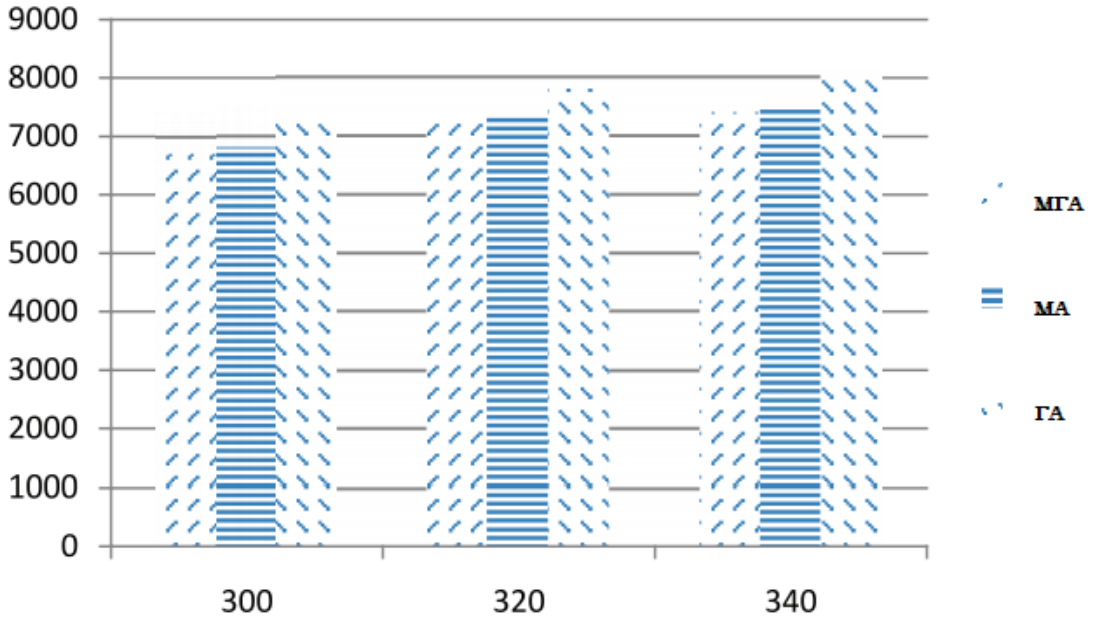

Рисунок 2.7 – Порівняльний аналіз отриманих рішень для 300-340 точок

В даному випадку модифікований алгоритм значно поступається гібридному мурашиному алгоритму в точності визначення довжини знайденого шляху. Тому можна зробити висновок, що його не слід застосовувати в задачах з великою кількістю вихідних даних – для відносно швидкого пошуку рішень доцільно використовувати мурашиний або мурашино-генетичний алгоритм.

Висновки до розділу

В процесі виконання другого розділу магістерської дисертації розглядалися основні засади функціонування алгоритму, описувалися біологічні основи та описувалася математична модель переміщення.

Після цього було проведено побудову алгоритму пошуку на графах в умовах великої кількості альтернатив.

Визначено структуризацію та алгоритм реалізації методу мурашки, наведено основні етапи побудови алгоритму. До їх складу варто віднести: введення вихідних даних і ініціалізація параметрів налаштовування і змінних алгоритму; визначення ймовірності переходу із одного мурашника в інший; генерація випадкового маршруту руху кожної мурахи з останнього місцезнаходження; визначення результуючої

довжини кожного маршруту (критерій оптимізації); збереження в пам'яті варіантів маршрутів з найкращим значенням критерію; розрахунок зміни концентрації феромону між двома мурашниками і нового значення концентрації феромону; повторення процедури до виконання одного або декількох обраних умов закінчення; виведення варіантів маршрутів, що забезпечують оптимальне значення критерію оптимальності.

Основна ідея алгоритму – приріст компонентів вектора варійованих параметрів, отриманого на залежному (від феромонів) ймовірному виборі компонентів. Це досягається шляхом заміни дискретного розподілу ймовірностей неперервною функцією, яка називається функцією щільності ймовірностей (Probability Density Function (PDF)), так само відомою як розподіл Гауса.

За для більш повного рішення поставленого завдання розроблюваний алгоритм було модифіковано за допомогою додавання функції, де за кожен крок виконання алгоритму до знайденої частини маршруту додається нове ребро.

Особливістю алгоритму є не конвергентність, тобто навіть після багатьох ітерацій одночасно досліджується безліч різних рішень, що дозволяє не зупинятися в локальних оптимумах.

3 ОПИС АРХІТЕКТУРИ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектури системи

Розроблена система реалізована як веб-застосування, діаграма варіантів застосування системи зображена у додатку В.

Архітектура системи побудована на стандартній клієнт-серверній архітектурі та складається з таких частин як:

- сервер баз даних та сама база даних;
- серверна частина;
- клієнтська частина.

База даних є безпосередньо місцем, в якому містяться дані, та з якої можна отримати дані при відповідному запиті.

На серверній частині описана проектна бізнес-логіка, яка є проміжною ланкою між сховищем даних та клієнтською частиною, що дозволяє надавати користувачу тільки ті дані, на які йому наданий дозвіл.

Клієнтська сторона є саме тим місцем, в якому працює користувач системи. На ній відображається реалізований інтерфейс і також певна визначена логіка. Оскільки клієнтська частина в даному випадку це веб застосування, то перевізникам та операторам для роботи потрібно використовувати такий тип програмного забезпечення як веб переглядач.

Узагальнена схема архітектури системи зображена на рис. 3.1

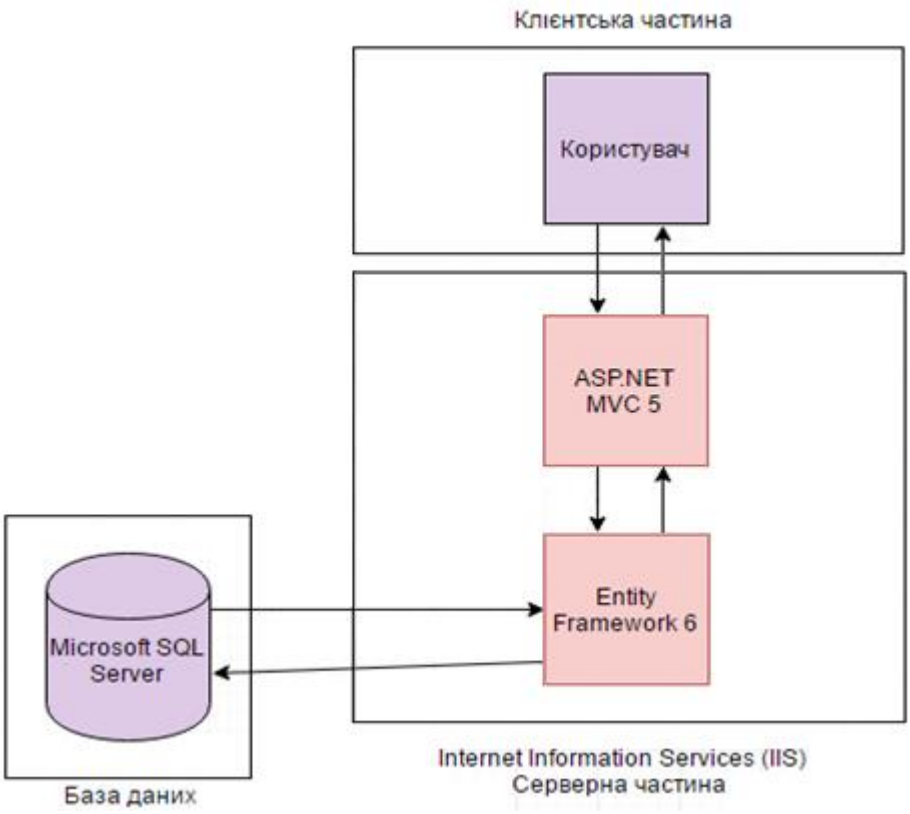

Рисунок 3.1 — Архітектура системи

Відповідно до спроектованої архітектури веб-браузер оператора або кур’єра обмінюється даними з фреймворком ASP.NET MVC за допомогою відповідних запитів. Для отримання, оновлення та видалення даних зі сховища використовується ORM технологія Entity Framework, яка працює в термінах сутностей, що в свою чергу дозволяє не заглиблюватися в деталі роботи реляційних баз даних. Окрім того, така технологія в архітектурі має місце тому, що вона виступає посередником між СУБД Microsoft SQL Server та бібліотеками, які знаходяться на сервері, це в свою чергу дає змогу здійснювати підтримку системи без додаткових маніпуляцій. Детальна структурна схема системи зображена на додатку Г.

3.1.1 Вибір засобів для реалізації системи

Оберемо необхідні технології та бібліотеки для розробки швидкої та ефективної системи побудови маршрутів.

3.1.1.1 Вибір мов програмування

Для розробки системи були вибрані мови програмування JavaScript для реалізації клієнтської частини і C# та платформа Microsoft .NET для серверної.

Мову програмування C# та платформу .NET було обрано через наступні причини:

- об’єктно-орієнтований підхід – дозволяє описати абстракції та взаємодію між ними;
- висока швидкість та продуктивність роботи;
- красивий, інтуїтивно зрозумілий та надійний синтаксис і стиль написання програмного коду;
- дозволяє швидше ніж будь яка інша мова розробляти програмні рішення;
- строга типізація – дає змогу уникати помилок при роботі з типами даних;
- платформа містить велику кількість готових шаблонів та бібліотек відразу (для створення віконних, мобільних, веб застосувань тощо), що дозволяє зекономити час на пошук та встановлення їх з різних джерел;
- зрозуміла та докладна документація та велика кількість навчальних матеріалів на офіційних джерелах.

Окрім мови C# з платформи Microsoft .NET для реалізації системи були використані технології Entity Framework та ASP.NET MVC.

Реалізація клієнтської частини реалізована з використанням трійки основних технологій для створення веб-застосувань, до них відносяться:

- HTML (Hypertext Markup Language) – для розмітки веб-додатку;
- CSS (Cascading Style Sheets) – для оформлення стилю системи;
- JavaScript – є мовою, яка дозволяє маніпулювати веб-сторінкою, в системі вона була використана для написання самого алгоритму побудови маршруту та іншої логіки на клієнтській стороні.

3.1.1.2 Вибір інтегрованого середовища розробки (IDE)

Для написання будь якого застосунка можна використовувати навіть звичайний текстовий редактор, але це дуже незручно і займає багато часу. Тому були розроблені такі типи допоміжних програмних рішень як інтегровані середовища розробки або скорочено IDE (Integrated development environment), які містять окрім текстового редактора коду ще інтерпретатор або компілятор, відлагоджувальник, засоби автозаповнення коду та інші корисні доповнення.

Для створення рішення було вибране середовище Microsoft Visual Studio, яке дозволяє створювати цілий набір різних типів застосунків, таких як: веб-додатки, мобільні застосунки, служби, програми і графічним інтерфейсом тощо.

Для розробки застосування дане середовище було вибране завдячуючи наступним перевагам:

- просте створення проекту та підключення необхідних додаткових бібліотек як із внутрішнього набору, так і з зовнішніх джерел за допомогою менеджера пакетів nuget;
- зручна інтелектуальна функція IntelliSense, яка відображає можливі наступні інструкції ще до початку введення та перехоплює помилки і показує вірні варіанти;
- можливість створювати і вставляти готові шаблони коду як за допомогою гарячих клавіш, так і через вибір з інтерфейсу;
- широкі можливості відлагоджування коду (встановлення точок зупинок, відображення стану пам’яті, центрального процесора та інших ресурсів на певних ділянках коду);
- вбудований веб-сервер, який дає змогу запускати такі веб-додатки як розроблюване безпосередньо в середовищі без можливості підключення пристроїв зі сторонніх мереж;
- вбудовані засоби ефективного рефакторингу сирцевого коду;
- можливість роботи з системами контролю версій без використання додаткового програмного забезпечення;

– функція для роботи з системою управління базами даних яка дозволяє переглядати та змінювати структуру таблиць, керувати записами та інші.

3.1.1.3 Фреймворк ASP.NET MVC

Технологія ASP.NET MVC – це набір засобів та бібліотек, які дозволяють створювати веб-сайти та застосунки в контексті шаблону проектування MVC (модель – представлення - контролер).

Для реалізованої системи така технологія підходить по кількох причинах:

– можливість інтегрування з JavaScript бібліотеками без складнощів;
– повний контроль над HTML розміткою, яка дозволяє відображати елементи інтерфейсу;
– розширюваність за рахунок реалізації абстрактних типів;
– можливість створювати додатки, що легко тестуються та супроводжуються;
– відкритість інфраструктури;
– сучасна система маршрутизації.

На рисунку 3.2 представлений шаблон MVC в архітектурі системи побудови маршрутів.

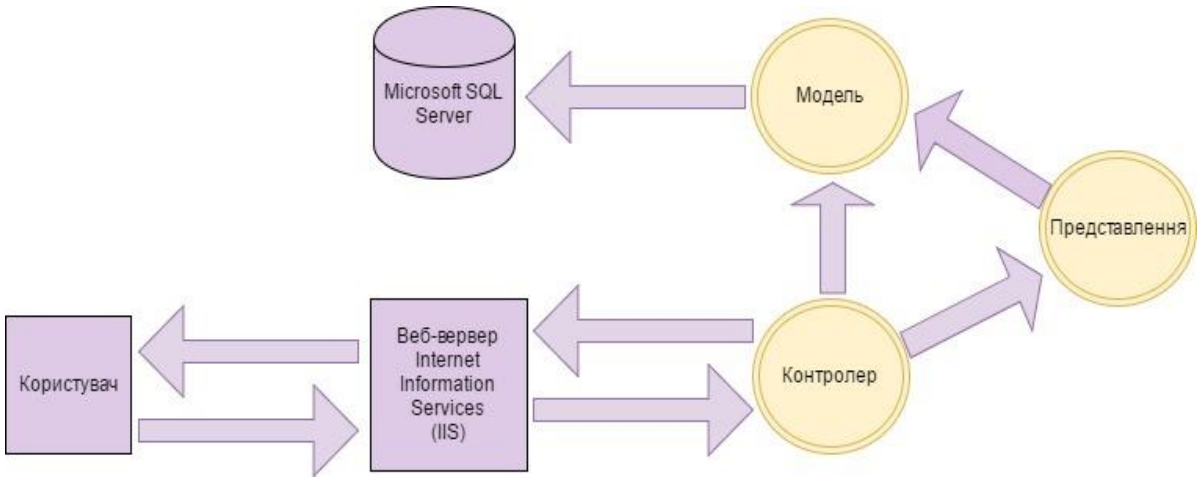

Рисунок 3.2 — Шаблон проектування MVC в архітектурі системи

Паттерн MVC розділяє застосунок та три основні частини:

- представлення (View) – це безпосередньо користувацький інтерфейс, з яким взаємодіє користувач;
- контролер (Controller) – проміжний клас, який зв’язує користувацький інтерфейс з системою управління базою даних враховуючи реалізовані обмеження. Під час роботи отримує та обробляє запити від користувача та повертає представлення, файл, потік байтів та інші типи об’єктів;
- модель – частина, призначення якої полягає в описі даних, які обробляються.

3.1.1.4 Технологія Entity Framework

Entity Framework є object-relational mapping (ORM) технологією, яка дозволяє абстрагуватись від роботи з реляційною базою даних та працювати в контексті об’єктно-орієнтованої парадигми.

При роботі з використанням такої технології розробник по суті працює з об’єктами, а не з таблицями та ключами.

Основні переваги даної технології при реалізації системи:

- швидка робота;
- повний контроль над класами сутностей;
- робота з контекстом подібна до роботи з шаблоном репозиторія;
- проста робота для застосунків, в який активно додаються нові поля;
- ліниве завантаження, яке дозволяє вивантажувати дані тільки при зверненні до них.

3.1.1.5 СУБД Microsoft SQL Server

Microsoft SQL Server – це сучасна система управління для баз даних від корпорації Microsoft, яка дозволяє оперувати реляційною моделлю даних.

До переваг даної СУБД можна віднести:

- висока продуктивність;
- сучасні методи забезпечення безпеки даних;
- відмовостійкість;
- хороша масштабованість;
- простота використання з іншими продуктами та платформами корпорації Microsoft.

3.1.2 Проектування підсистем

В наступних підпунктах описані архітектурні особливості та складові основних частин, які функціонують в системі.

3.1.2.1 Опис моделі бази даних

Перш за все для побудови системи потрібно визначити об’єктну модель майбутньої системи, яка допоможе описати взаємозв’язок між даними та їхню структуру.

Виділимо основні сутності, які використовуються в системі та їхні основні атрибути:

- кур’єр (унікальний ідентифікатор, ім’я, прізвище, по батькові, номер телефону, електронна пошта, ім’я користувача, пароль, ознака активності);
- замовлення (унікальний ідентифікатор, дата початку, дата закінчення, загальна вартість, коментар, статус);
- товар (унікальний ідентифікатор, назва товару, штрих-код, кількість, вартість);
- замовник (унікальний ідентифікатор, ім’я, прізвище, по батькові, адреса електронної пошти, номер телефону, вулиця, номер будинку, номер квартири, номер кімнати);
- оператор (унікальний ідентифікатор, ім’я, прізвище, по батькові, ім’я користувача, пароль, електронна пошта);
- координати (унікальний ідентифікатор, довгота, широта, адреса).

Основною сутністю для створення екземплярів інших є сутність оператора, яка здатна створювати окремих представників для замовника, замовлення та кур’єра, і ще окрім того призначає координати замовнику. Кур’єр має відношення «один до багатьох» з сутністю замовлення і при цьому має тільки один екземпляр сутності координат, які описують його розташування на поточний момент часу. Діаграма сутність-зв’язок відображена на додатку Д.

3.1.2.2 Опис структури серверної частини

Оскільки в серверній частині застосовувалася бібліотека ASP.NET MVC, то відповідно до однойменного шаблону проектування, дані отримуються з моделі відповідно до описаних бізнес-правил, обробляються в відповідному контролері враховуючи маршрутизацію та відображаються в користувацький інтерфейс. Відповідно до інших запитів від клієнтської частини, в системі вибирається контролер-обробник та його певний метод, до якого пішов запит.

Обрання контролера та методу-обробника відбувається за допомогою вбудованої інфраструктури маршрутизації, яка дає змогу будувати співвідношення між адресою (URL) та деякою дією контролера без потреби існування на сервері файлу, який є кінцевою точкою відповідно до адреси. В розробленій системі визначені маршрути по замовчуванню відповідно до ролі авторизованого користувача.

Послідовність обробки вхідних запитів від клієнтської сторони та повернення відповіді зображена на рисунку 3.3.

Рисунок 3.3 — Схема обробки запитів

Відповідно до відображеної схеми виконуються наступні кроки:

1) після отримання запиту по вказаній адресі на веб-сервері з використанням системи маршрутизації виконується пошук відповідного маршруту та запускається ініціалізація;
2) зберігається інформація про запит з використанням компонента запитуючого застосування;
3) після знайдення маршруту вибирається контролер та метод-обробник за допомогою компонента маршрутизації;
4) перевіряються права доступу до контролера;
5) генерується екземпляр контролера;
6) в контролері шукається запитувальний метод і після знайдення визначається відповідність параметрів;
7) метод обробляє запит відповідно що переданих параметрів, генерує представлення та повертає його з відповідним HTTP статус-кодом.

Окрім фреймворку ASP.NET MVC для отримання даних з сховища даних була інтегрована ORM бібліотека Entity Framework, яка дозволяє уніфікувати роботу з

різними системами управління базами даних використовуючи при цьому один інтерфейс та шаблони проектування репозиторій Repository і Unit of work.

Шаблон Repository є зручним паттерном для роботи з даними, який призначений для розділення класів даних від іншої логіки застосування. В той же час шаблон Unit of work зазвичай використовується сумісно з репозиторієм і дає можливість узагальнити роботу з різними репозиторіями використовуючи при цьому єдиний контекст даних.

3.1.2.3 Опис структури клієнтської частини

Клієнтська частина є однією з найважливіших для користувача, оскільки користувач безпосередньо з нею взаємодіє не задумуючись про існування серверної частини та інших.

В розроблюваному застосуванні на даній частині був реалізований блок функціональності для самого водія та оператора (адміністратора), який має можливість призначати перевезення, керувати записами отримувачів, замовлень та кур’єрів.

Для користувачів системи були реалізовані наступні сторінки:

а) доступні для обох типів:

- сторінка реєстрації;
- сторінка входу;

б) доступні для водія:

- базова сторінка з інформацією про перевезення та мапою з побудованим маршрутом;

в) доступні для оператора:

- базова сторінка зі списком активних кур’єрів та місцями їхнього поточного знаходження;
- сторінка керування отримувачами (створення, редагування, видалення);
- сторінка керування кур’єрами (створення, редагування видалення);

- сторінка керування замовленнями (створення, редагування, видалення);
- сторінка керування обліковим записом оператора.

Для відображення маршруту був використаний сервіс «Карти Google», тому для реалізації побудови його на карті застосовувався прикладний програмний інтерфейс «Google Maps JavaScript API», який містить засоби, що дозволять виконувати маніпуляції з картами з використанням мови JavaScript. Діаграма послідовностей для оператора зображена на додатку Е, діаграма послідовностей для кур'єра зображена на додатку Ж, мурашиний алгоритм побудови маршрутів зображений на додатку И.

3.2 Реалізація системи

В даному підрозділі описуються нюанси реалізації серверної, клієнтської та підсистеми роботи з даними.

3.2.1 Реалізація структури бази даних

Оскільки для виконання операцій створення, редагування та видалення записів та таблиць в базі даних застосована спеціальна технологія Enity Framework відповідно до програмного коду. В такому випадку все виконується автоматично без безпосереднього написання SQL коду розробником. Надалі описана структура створених таблиць при використанні такого методу.

Специфікація таблиць бази даних в системі побудови маршрутів відображена на таблицях 3.1-3.6:

Таблиця 3.1 — Оператори (Users)

Назва	Тип	Опис	Обмеження
Id	nvarchar	Унікальний ідентифікатор запису	Primary key
UserName	nvarchar	Ім'я користувача (Логін)	Unique key
Password	nvarchar	Пароль	
PhoneNumber	nvarchar	Номер телефону	

Продовження таблиці 3.1

Email	nvarchar	Адреса електронної пошти користувача	
Name	nvarchar	Ім’я	
LastName	nvarchar	Прізвище	
SurName	nvarchar	По батькові	

В даній таблиці зберігаються дані операторів після проходження процедури реєстрації в системі.

Таблиця 3.2 — Перевізник (Courier)

Назва	Тип	Опис	Обмеження
Id	nvarchar	Унікальний ідентифікатор запису	Primary key
UserName	nvarchar	Ім’я користувача	
Password	nvarchar	Пароль	
Name	nvarchar	Ім’я	
LastName	nvarchar	Прізвище	
SurName	nvarchar	По батькові	
IsActive	bit	Ознака активності	
PhoneNumber	nvarchar	Номер телефону	
Email	nvarchar	Адреса електронної пошти	
CurrentCoordinateId	nvarchar	Посилання на координати поточного розміщення	Foreign key

В таблиці для збереження інформації про перевізників є дані для отримання доступу в систему і також зовнішній ключ на таблицю з координатами, який використовується для збереження координат поточного знаходження.

Таблиця 3.3 — Отримувач (Receiver)

Назва	Тип	Опис	Обмеження
Id	nvarchar	Унікальний ідентифікатор отримувача	Primary key
Name	nvarchar	Ім’я	

Продовження таблиці 3.3

LastName	nvarchar	Прізвище	
SurName	nvarchar	По батькові	
EmailAddress	nvarchar	Адреса електронної пошти	Unique key
MobilePhone	nvarchar	Номер мобільного	Unique key
StreetName	nvarchar	Назва вулиці	
HouseNumber	int	Номер дому	
FloorNumber	int	Номер поверху	
FlatNumber	int	Номер квартири	
CurrentCoordinateId	nvarchar	Посилання на розміщення отримувача	Foreign key

В даній таблиці присутні контактні дані, дані поточного місця проживання отримувача, які призначені для коректної доставки вантажу та координати для відображення на карті.

Таблиця 3.4 — Замовлення (Order)

Назва	Тип	Опис	Обмеження
Id	nvarchar	Унікальний ідентифікатор	Primary key
Date	datetime	Дата	
TotalPrice	decimal	Загальна вартість	
Comment	nvarchar	Коментар	
Address	nvarchar	Адреса замовлення	
ReceiverId	nvarchar	Посилання на отримувача	Foreign key
StatusId	nvarchar	Посилання на статус	Foreign key
CourierId	nvarchar	Посилання на перевізника, який виконує замовлення	Foreign key

Замовлення має дату, коментар, адресу, загальну суму і також посилання на кур’єра, який його виконує та клієнта отримувача.

Таблиця 3.5 — Одиниця товару (OrderItem)

Назва	Тип	Опис	Обмеження
Id	nvarchar	Унікальний ідентифікатор товару	Primary key

Продовження таблиці 3.5

ItemName	nvarchar	Найменування речі, яка перевозиться	
BarCode	nvarchar	Штрих-код продукту	Unique key
Price	decimal	Вартість товару	
Quantity	int	Кількість	
OrderId	nvarchar	Адреса по координатах	Foreign key

Одиниці товару містяться в замовленні за допомогою використання зовнішнього ключа. Для опису товару існують колонки назви, штрих-коду, вартості та кількості.

Таблиця 3.6 — Координата (Coordinate)

Назва	Тип	Опис
Id	nvarchar	Унікальний ідентифікатор координат
GeoLat	nvarchar	Широта
GeoLong	nvarchar	Довгота
Address	nvarchar	Адреса

В таблиці координат окрім первинного ключа ще є широта, довгота та адреса відповідно до координати. Детальна діаграма баз даних системи зображена на додатку К.

3.2.2 Реалізація серверної функціональності

В інтегрованому середовищі розробки рішення побудоване на багаторівневій архітектурі, яка дозволяє розділити застосування на різні незалежні один від одного рівні, які по суті являють собою проекти в середовищі. Такий принцип дозволяє розділити розробку по ролях, тобто дизайнер займається розробкою графічної частини (рівень представлення), прикладний розробник займається бізнес-логікою та організовує роботу з даними.

Схематично це представлено на рисунку 3.4.

Рисунок 3.4 — Трирівнева архітектура рішення

Рівень представлення відповідає безпосередньо за інтерфейс, з яким взаємодіє користувач. На даному рівні містяться сторінки, стилі, логіка побудови маршруту з використанням карт Google, стилі та інші складові.

На рівні бізнес-логіки – функціональність валідації з використанням атрибутів, обчислення, права доступу тощо. Даний рівень є посередником між сховищем даних та графічним інтерфейсом користувача.

Рівень доступу до даних містить сутності даних та виконує взаємодію з базою даних використовуючи ORM технології, в нашому випадку такою системою є Entity Framework. Також на даному рівні використані шаблони проектування репозиторій та unit of work.

Відповідно до цієї методики, в системі побудови маршрутів для кур’єрських перевезень структура рішення має вигляд, зображений на рисунку 3.5.

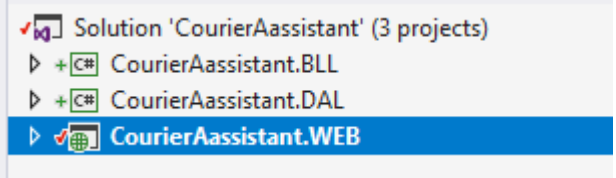

Рисунок 3.5 — Структура рішення на основі трирівневої архітектури

На рівні даних містяться сутності для користувача, ролі користувача, кур'єра, отримувача, замовлення, товару, координат та профілю користувача, в якому визначені загальні атрибути (ім'я, прізвище, дата народження тощо), що дозволяє уникнути дублювання однотипного коду. Для виконання операцій з даними через використання Entity Framework визначений клас для роботи з контекстом бази даних AppContext, що містить набори даних для кожної сутності.

Для інкапсуляції роботи логіки взаємодії з джерелами даних застосований паттерн репозиторій, зокрема був створений інтерфейс IRepository та реалізуючі його менеджери OrderManager, UserManager та ReceiverManager. Окрім того, щоб винести загальну функціональність для всіх репозиторіїв, був створений інтерфейс IUnitOfWork та клас EFUnitOfWork.

Отримана структура для рівня доступу до даних відображена на рисунку 3.6.

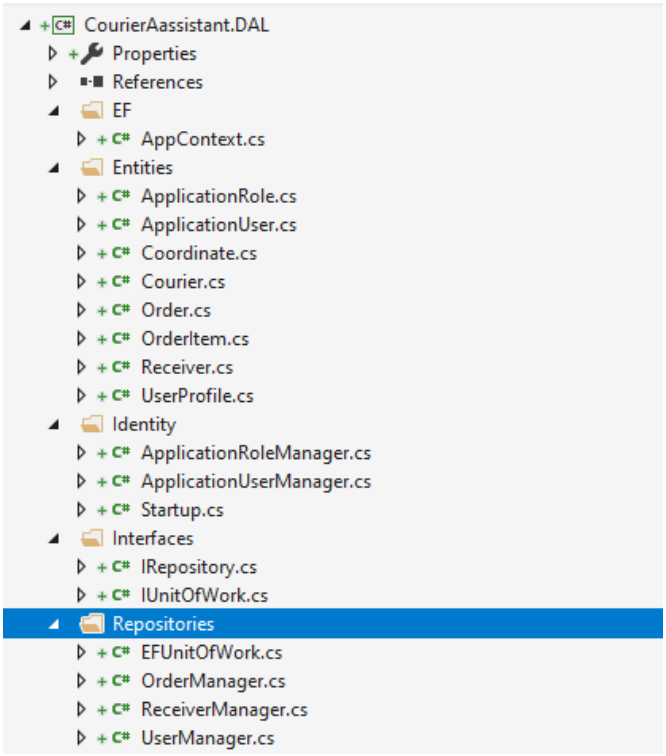

Рисунок 3.6 — Структура проекту для рівня доступу до даних

Рівень бізнес-логіки приймає дані з відповідного рівня, обробляє та повертає для відображення, і по такому ж принципу в зворотному порядку. Оскільки даний рівень є посередником між рівнем даних та представленням, то з'являється необхідність створення спеціальних проміжних сутностей, в яких відсутня будь яка поведінка, що дозволяє передавати дані між рівнями. Для цього був реалізований паттерн Data Transfer Object (DTO) та створено класи для кожної сутності.

Безпосередня взаємодія з рівнями відбувається за допомогою спеціалізованих класів-сервісів, для яких з'являється необхідність ввести бібліотеку інверсії залежностей. В даному випадку застосований сучасний продукт NInject.

Структура рівня бізнес-логіки представлена на рисунку 3.7.

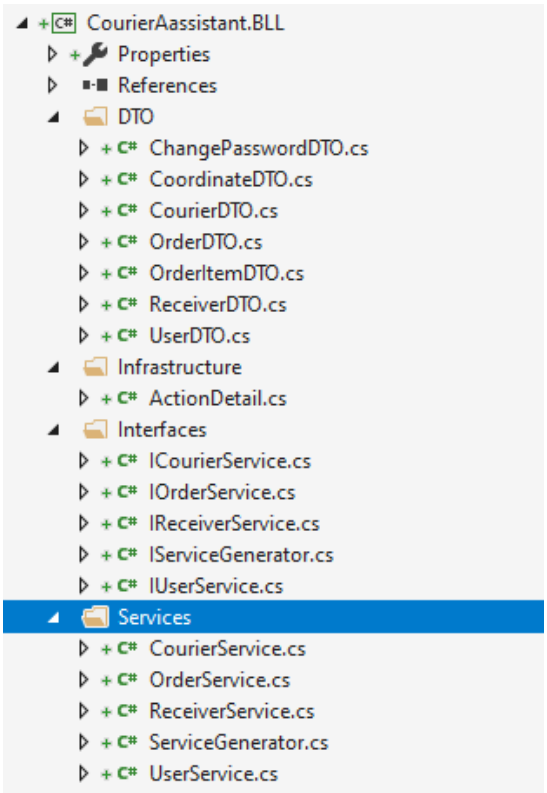

Рисунок 3.7 — Структура проекту для рівня доступу до даних

Основними складовими частинами на інтерфейсному рівні є контролери. В системі були створені контролери для роботи з даними користувача, кур'єрів, отримувачів та контролер основного робочого середовища оператора.

Для отримання даних з рівня бізнес-логіки та відображення їх користувачу потрібно визначити моделі представлення, об'єкти яких створюються за допомогою

спеціального засобу маппінгу. В даному випадку були визначені моделі для користувача та керування користувачем, перевізників, координат, отримувачів, замовлень та ідентифікації.

Окрім того до теки з скриптами для веб-переглядача додана бібліотека Google Maps JavaScript API для роботи з картами, та файл з реалізованим сценарієм побудови маршрутів.

Структура проекту для рівня представлення відображена на рисунку 3.8.

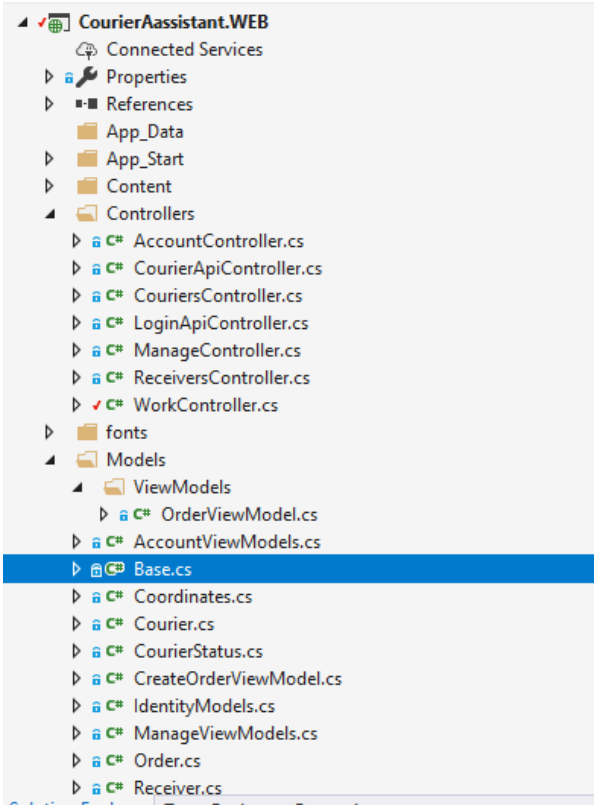

Рисунок 3.8 — Структура проекту для рівня представлення

Діаграма класів системи побудови маршрутів відображена на додатку Л.

3.2.3 Реалізація клієнтської функціональності

В залежності від типу користувача відкривається певна сторінка. Для оператора та кур’єра це є відповідне робоче середовище.

При переході по визначеній адресі ресурсу користувачу доступні 2 типи сторінок:

– сторінка входу;
– сторінка реєстрації.

Після введення даних авторизації оператору стають доступними наступні сторінки:

– робоче вікно з поточними розміщеннями кур’єрів з вмістом маршрутів;
– вікно додавання, редагування та видалення замовлень, кур’єрів, товарів та отримувачів;
– вікно налаштування основного облікового запису.
– сторінка управління довідковою інформацією системи.

Для водія після входу відкривається основне робоче вікно з картою, на якій відображений маршрут руху та необхідна інформація про поточне перевезення.

Детальний вміст та логіка реалізованих сторінок:

1) Вікно входу:

Початковою сторінкою для більшості інтернет сервісів з авторизацією є сторінка входу. На ній відображене поле електронної пошти, паролю та прапорець збереження даних входу (рисунок 3.9).

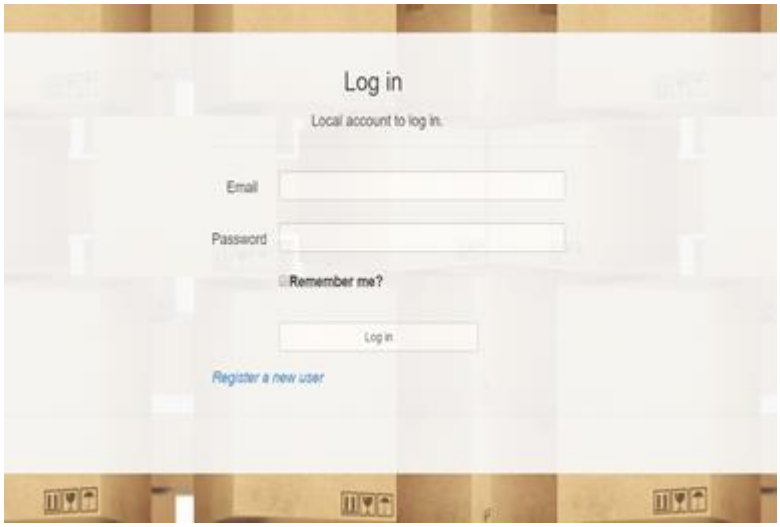

Рисунок 3.9 — Вікно входу

При введенні неактуальних або неправильних даних відображається підказка помилки та опису помилки (рисунок 3.10).

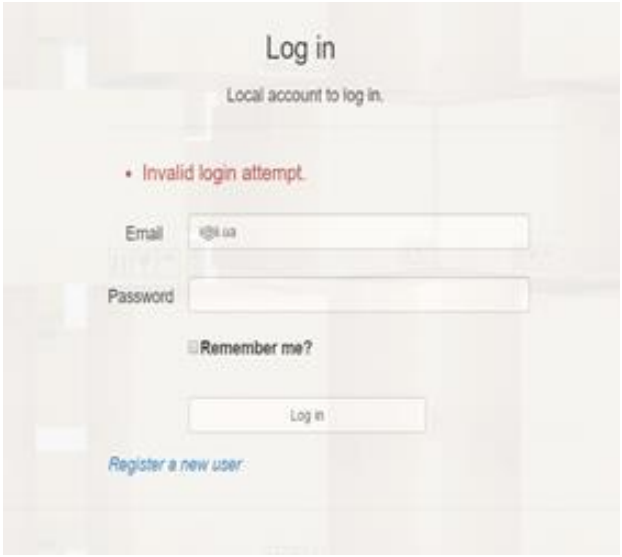

Рисунок 3.10 — Вікно входу при введенні помилкових даних

2) Форма реєстрації:

Перед входом в систему потрібно, щоб були актуальні авторизаційні дані в сховищі даних, для цього потрібно пройти процес реєстрації на відповідній сторінці. Після заповнення необхідних полів та натискання на кнопку реєстрації виконується запит на додавання нового користувача, перевіряється правильність введених даних та відсутність дублів. Якщо перевірки пройшли успішно, то виводиться повідомлення про успішну реєстрацію та виконується перехід на сторінку входу. Форма реєстрації відображена на рисунку 3.11.

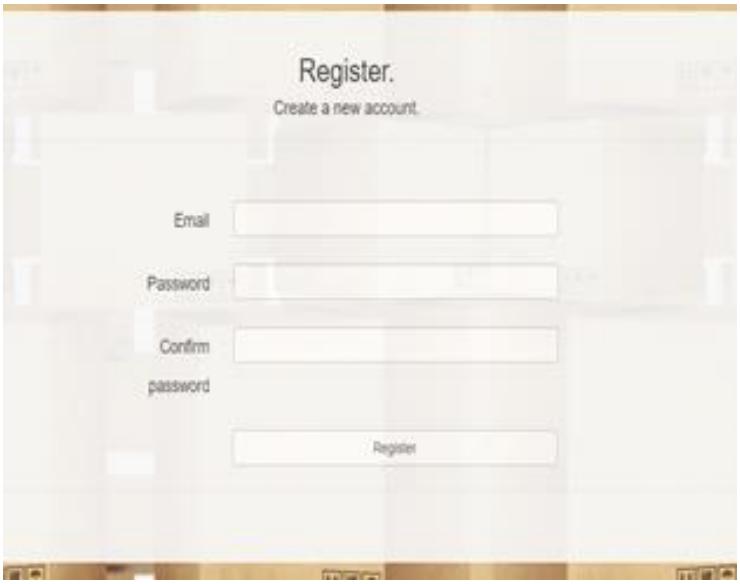

Рисунок 3.11 — Форма реєстрації нового користувача

3) Сторінка створення замовлення:

На сторінці створення нового замовлення є поля обрання кур’єра та отримувача замовлення, форма додавання товарів, текстові форми введення інформації та карта з маркером для встановлення місця доставки замовлення. Маркер можна встановити одноразовим натисканням на кінцевий пункт або пересуванням мишею. Сторінка зображена на рисунку 3.12.

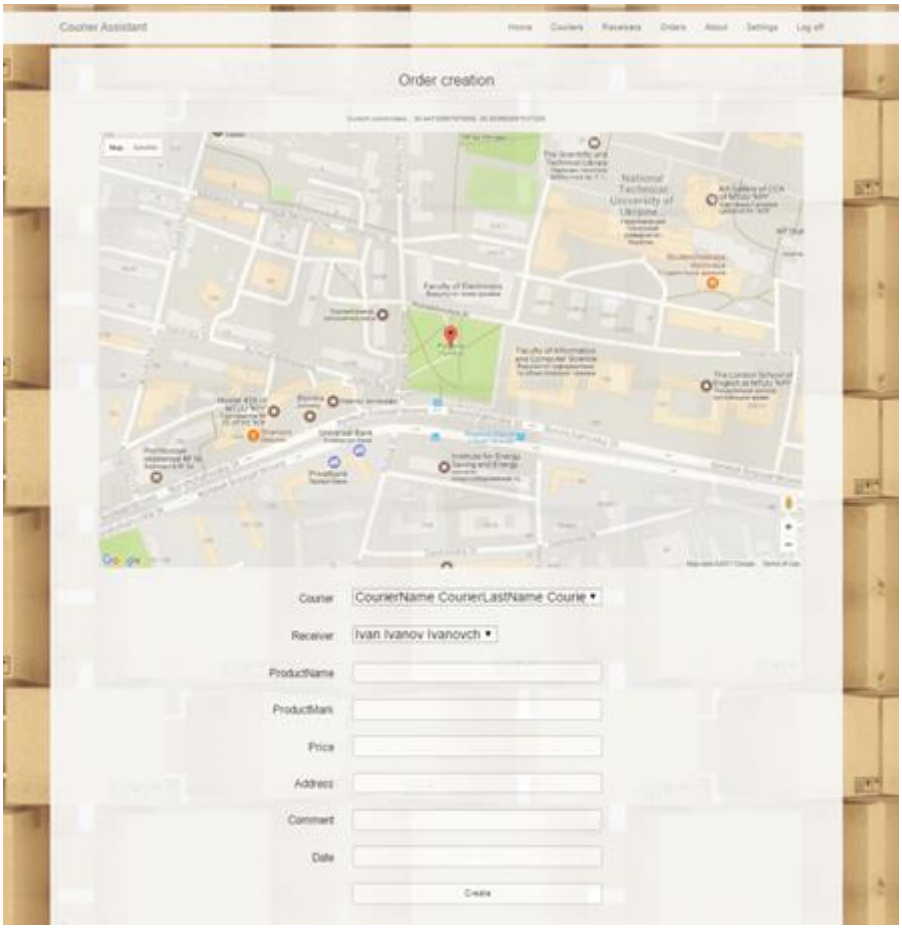

Рисунок 3.12 — Сторінка оператора для створення нового замовлення

4) Сторінка оператора з інформацією про поточні перевезення:

Після створення замовлення, призначення кур’єра та отримувача, додавання товарів та натискання на кнопку «Створити», оператор може слідкувати за процесом виконання за допомогою сторінки огляду активних перевезень (рисунок 3.13).

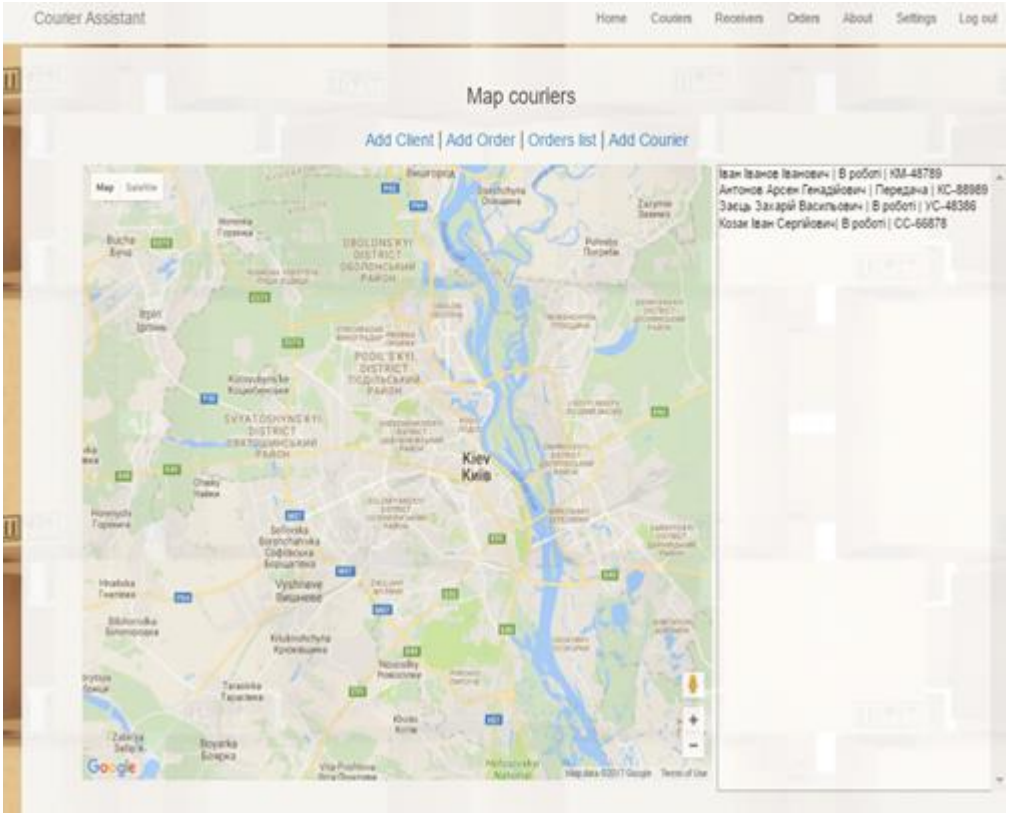

Рисунок 3.13 — Сторінка оператора для огляду поточних перевезень

Тут можна побачити актуальне на даний час розміщення кур’єра на карті, статус замовлення та інформацію про отримувача та товари. Також з цієї сторінки можна перейти до створення кур’єра, замовлення та отримувача.

3.3 Тестування розробленої функціональності

В даному підрозділі описаний процес ручного тестування реалізованих функцій системи застосовуючи тестові сценарії (тест-кейси). Тестові сценарії дозволяють описати послідовність дій та необхідних умов, які дадуть можливість перевірити функціональність розроблюваної системи.

На таблиці 3.7 описаний сценарій тестування функціональності входу.

Таблиця 3.7 — Тестовий сценарій входу в систему

Назва тесту	Тест входу в систему	
Функція	Вхід в систему	
Дія	Очікуваний результат	Результат
Передумова		
Перейти по головній адресі системи	Відкрита сторінка входу в систему	Пройшов
Кроки тестування		
Ввести неправильну адресу електронної пошти	З’являється спливаюче повідомлення з довідковою інформацією по помилку	Пройшов
Ввести електронну адресу та пароль, які не збережені в базі даних	З'являється спливаюче повідомлення про відсутність таких даних в системі	Пройшов
Ввести адресу пошти та пароль, які зареєстровані в системі	Виконується вхід на основну робочу сторінку	Пройшов
Післяумова		
Відкривається доступ до основного робочого середовища	Дається доступ на виконання основних дій в системі	Пройшов

На таблиці 3.8 протестована функціональність реєстрації нового кур’єра та оператора в системі перед початком виконання основних функцій в робочому вікні.

Таблиця 3.8 — Тестовий сценарій реєстрації користувача в системі

Назва тесту	Тест реєстрації
Функція	Додавання нового користувача в систему

Продовження таблиці 3.8

Дія	Очікуваний результат	Результат
Передумова		
Натиснути на пункт «Реєстрація нового користувача» на сторінці входу	Відкрита сторінка реєстрації нового користувача	Пройшов
Кроки тестування		
Вказати адресу електронної пошти, яка не відповідає формату електронних адрес	З’являється спливаюче повідомлення з інформацією про правильний формат електронних адрес	Пройшов
Вказати пароль, який не відповідає політиці формату паролів в системі	З'являється повідомлення з правилами формування паролю користувача	Пройшов
Ввести електронну адресу, яка відповідає формату та пароль, який відповідає політиці паролів	Виконується вхід в основне робоче середовище	Пройшов
Післяумова		
Відкривається доступ до основного робочого середовища	Дається доступ на виконання основних дій в системі	Пройшов
Створюється новий запис в сховищі даних з даними нового користувача	Дається можливість виконувати вхід в систему без повторної реєстрації	Пройшов

У таблиці 3.9 відображений тестовий сценарій додавання нового замовника.

Таблиця 3.9 — Тестовий сценарій додавання замовника

Назва тесту	Тест додавання замовника	
Функція	Створення замовника	
Дія	Очікуваний результат	Результат
Передумова		
Натиснути на пункт «Новий клієнт» на формі огляду поточних перевезень	Відображається сторінка з полями для нового клієнта	Пройшов
Кроки тестування		
Вказати контактний номер телефону, який не відповідає формату	З’являється повідомлення з актуальним форматом номеру телефону	Пройшов
Залишити незаповненими обов’язкові поля	Відображаються підказки про обов’язковість заповнення полів	Пройшов
Заповнити всі необхідні поля	Створюється запис нового отримувача	Пройшов
Післяумова		
Створюється запис нового отримувача	Повернення в основне робоче вікно оператора	Пройшов

На таблиці 3.10 проведене тестування створення нового замовлення з вибором кінцевого пункту на карті.

Таблиця 3.10 — Тестовий сценарій створення замовлення

Назва тесту	Тест додавання замовлення
Функція	Створення замовлення

Продовження таблиці 3.10

Дія	Очікуваний результат	Результат
Передумова		
Натиснути на пункт «Нове замовлення»	Відображається сторінка створення замовлення з картою	Пройшов
Кроки тестування		
Залишити невстановленим маркер на карті	З’являється повідомлення про обов’язковість встановлення маркеру	Пройшов
Залишити незаповненими обов’язкові поля	Відображаються підказки про необхідність заповнення обов’язкових полів	Пройшов
Встановити пункт на карті та заповнити інформацію про замовлення	Створюється запис нового замовлення з базі даних	Пройшов
Післяумова		
Створюється запис нового замовлення	Повернення в основне робоче вікно оператора з побудованим маршрутом до місця призначення	Пройшов

В останньому тесті виконане тестування побудови маршруту в робочому вікні перевізника (таблиця 3.11).

Таблиця 3.11 — Тестовий сценарій побудови маршруту для кур’єра

Назва тесту	Тест побудови маршруту
Функція	Побудова маршруту для кур’єра

Продовження таблиці 3.11

Дія	Очікуваний результат	Результат
Передумова		
Натиснути на пункт «Готовий» в середовищі кур’єра	Відображається інформація про товар та клієнта	Пройшов
Кроки тестування		
Натиснути на кнопку «Побудувати маршрут руху»	Виконається побудова маршруту до отримувача	Пройшов
Післяумова		
Відображається доріжка руху до кінцевого пункту	Маршрут руху побудований на карті	Пройшов

Висновки до розділу

В результаті проведеної розробки була реалізована система побудови маршрутів з використанням мурашиного алгоритму, побудована на стандартній клієнт-серверній архітектурі. Рішення у інтегрованому середовищі розробки було створене з використанням багаторівневого підходу, до якого входить рівень доступу до даних, бізнес-логіки та представлення. При проектуванні моделі реляційної бази даних були створені таблиці складових системи, визначені колонки та обмеження. У ході реалізації серверної частини реалізовувалися всі рівні архітектури. Зокрема для роботи з даними були реалізовані паттерни проектування, та використана технологія Entity Framework. На бізнес рівні були створені проміжні сутності та інтегрована інверсія залежностей. На клієнтському рівні були створені моделі представлення, контейнери, конфігурація маршрутизації та впроваджений робочий алгоритм побудови маршрутів з використанням мурашиного алгоритму. Клієнтська частина містить необхідні робочі сторінки для операторів системи та перевізників. Окрім того, були реалізовані форми входу та реєстрації в системі. В кінці розділу було проведене ручне тестування системи з використанням тестових сценаріїв.

4 ДОСЛІДЖЕННЯ СИСТЕМИ ПОБУДОВИ МАРШРУТІВ

В даному розділі описується дослідження якості та швидкості побудови шляхів при використанні мурашиного алгоритму в реалізованій системі. Для була використана побудова маршруту в розробленому застосуванні та стандартна функціональність побудови шляху сервісу Google Maps, який як відомо використовує алгоритм пошуку маршрутів A *.

Мета дослідження – довести, що мурашиний алгоритм в розробленій системі побудови маршрутів не поступається в ефективності алгоритму A *, який реалізований в функціональності продукту Google Maps.

Для дослідження в реальних умовах застосовується пошук на відомих маршрутах автошляхів міста Києва та порівнюється робота алгоритмів використовуючи 2 коротких та 2 довгих шляхи. Побудова на різних довжинах шляхів зумовлена тим, що алгоритми, які розглядаються, можуть показати різні результати якості та швидкості побудови.

В рамках перших двох експериментів розглянемо побудову коротких маршрутів (до 5 кілометрів).

Експеримент №1. Побудова шляху від станції метро Політехнічний інститут до площі Льва Толстого.

Для початку перевіримо побудову з використанням алгоритму мурашиних колоній (рисунок 4.1).

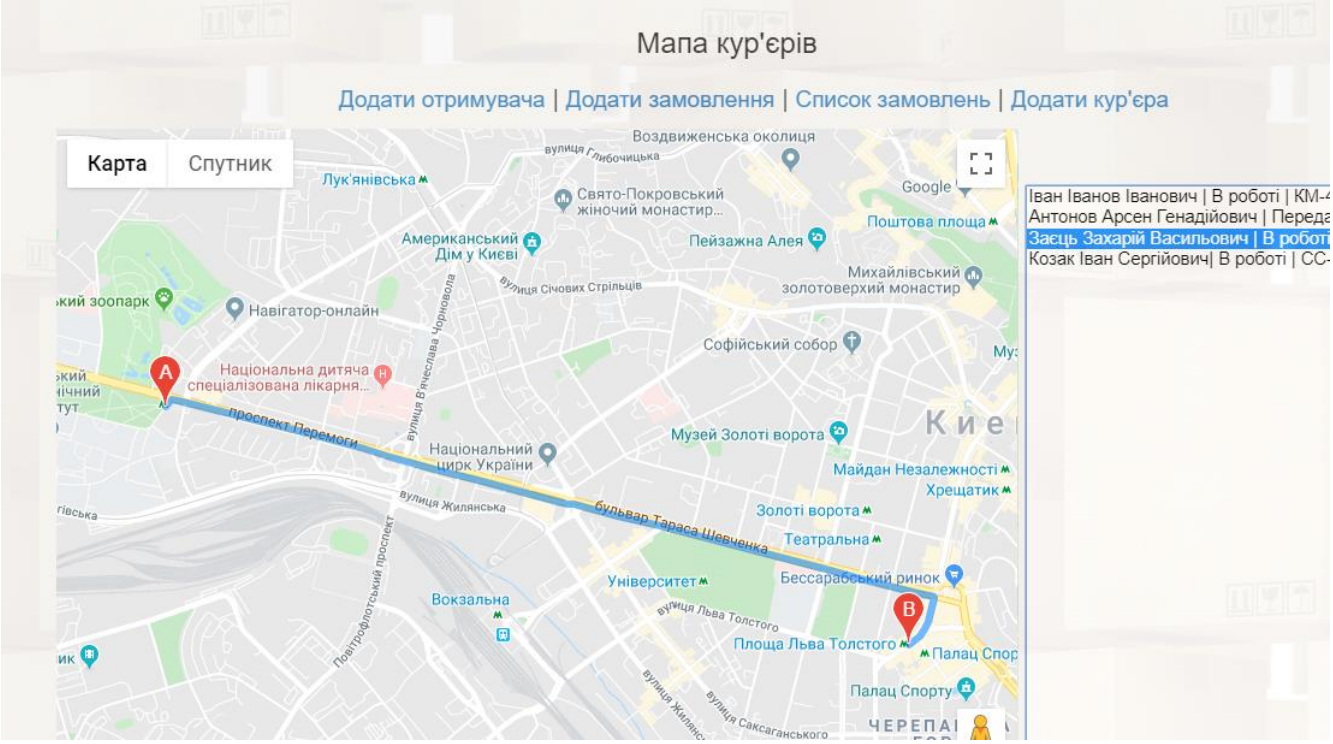

Рисунок 4.1 — Маршрут №1 в системі з використанням мурашиного алгоритму

В результаті побудови довжина маршруту – 4.3 кілометри, час побудови – 0.9 секунди. Тепер побудуємо для тих самих пунктів за допомогою продуку Google Maps, який використовує алгоритм A * (рисунок 4.2).

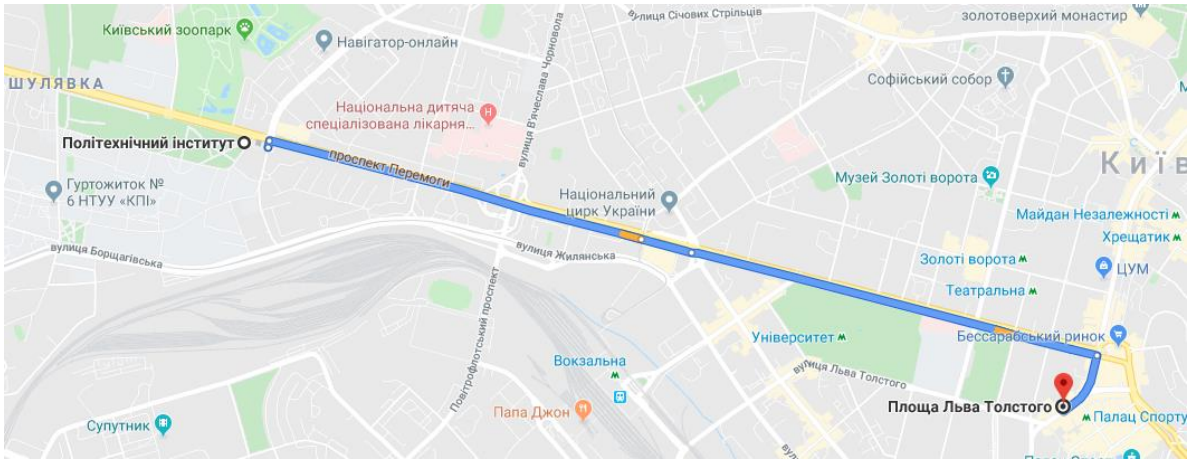

Рисунок 4.2 — Маршрут №1 при пошуку алгоритмом A *

Отримана довжина шляху при використанні A * дорівнює 4.3 кілометрам, швидкість побудови 0.8 секунди.

Як можна побачити, перший маршрут є простим, тому обидва алгоритми знайшли однаковий шлях та відпрацювали майже за однакову кількість часу.

Експеримент №2. Побудова шляху від Контрактової площі до вулиці Січових Стрільців (рисунок 4.3).

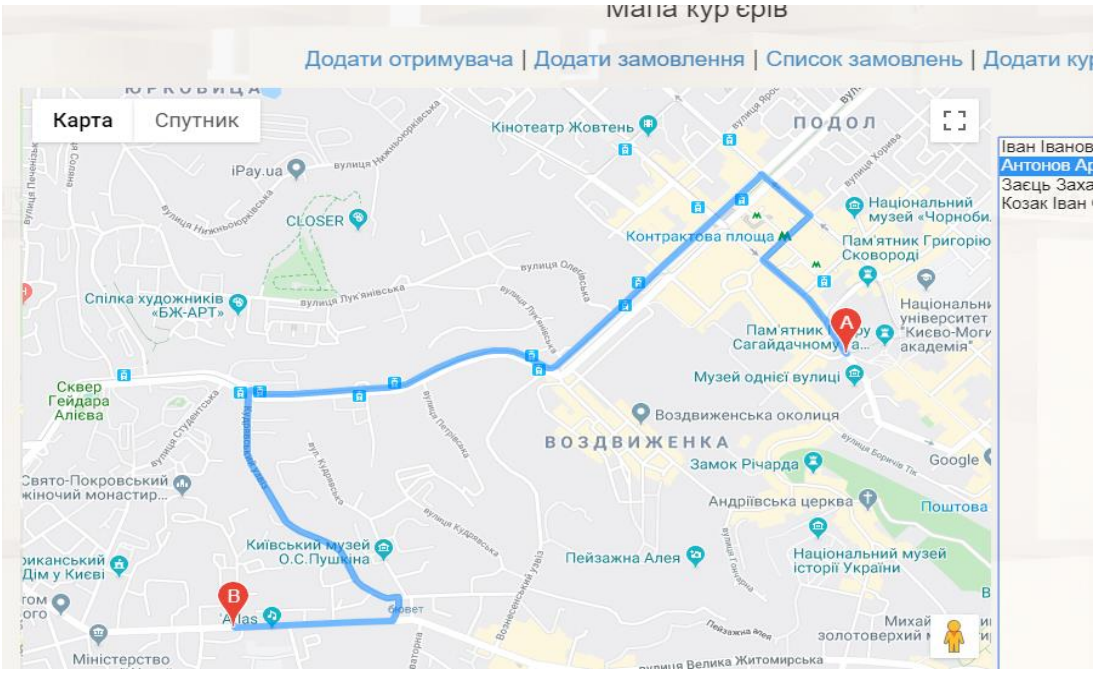

Рисунок 4.3 — Маршрут №2 в системі з використанням мурашиного алгоритму

Після побудови маршруту №2 мурашиним алгоритмом відстань дорівнює 3.455 кілометрам, при цьому швидкість знаходження рішення – 1.1 секунди. Застосуємо алгоритм A *, результат побудови якого зображений на рисунку 4.4.

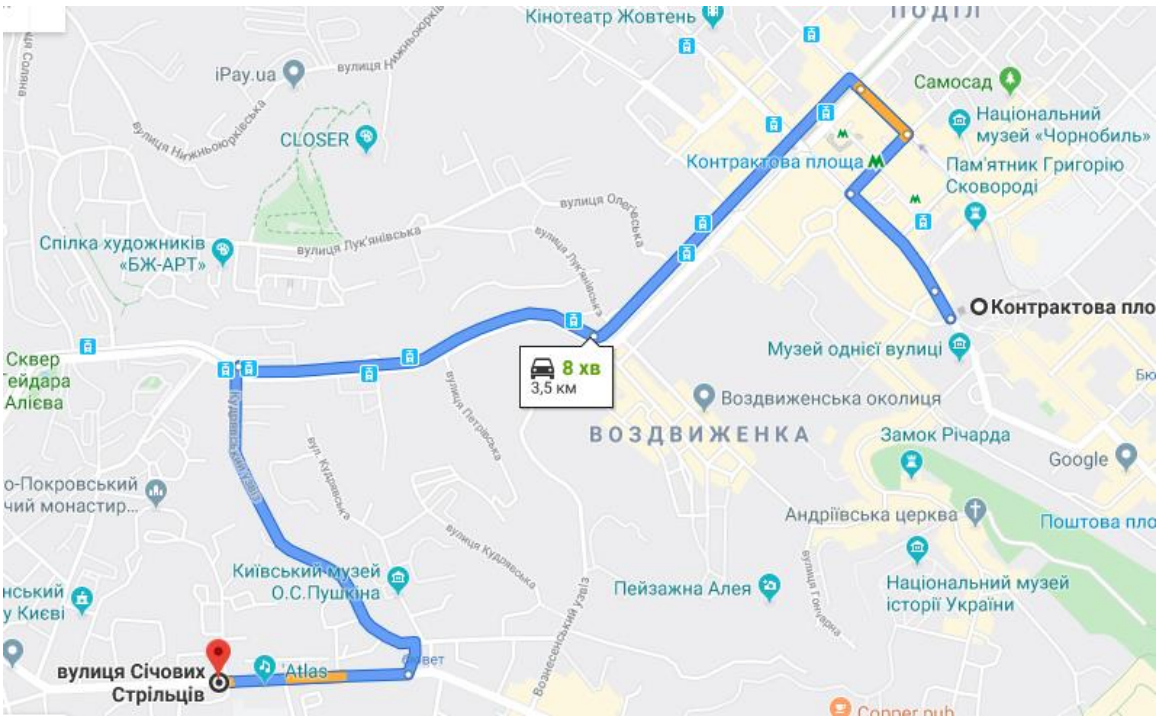

Рисунок 4.4 — Маршрут №2 при пошуку алгоритмом A *

В даному випадку довжина маршруту є однаковою як при побудові методом мурашиних колоній, однак швидкість побудови виявилася трохи меншою – 1.3 секунди.

Як можна побачити з результату побудови шляхів на коротких відстанях, алгоритми показали практично однакові результати по довжинах відстаней. Якщо дивитись по швидкості побудови, то при збільшенні кількості мурах в мурашиному алгоритмі обчислення виконувалося дещо повільніше, оскільки на короткому маршруті був швидко знайдений локальний оптимум і відповідно частина мурах виконувала рух по одному оптимальному маршруту не покращуючи при цьому результат. Тому була обрана оптимальна кількість, яка дорівнювала 20 і з таким значенням виконувався подальший пошук.

Для наступних досліджень були обрані вищі значення довжин шляхів.

Експеримент №3. Побудова шляху від станції Почайна до станції Васильківська (рисунок 4.5).

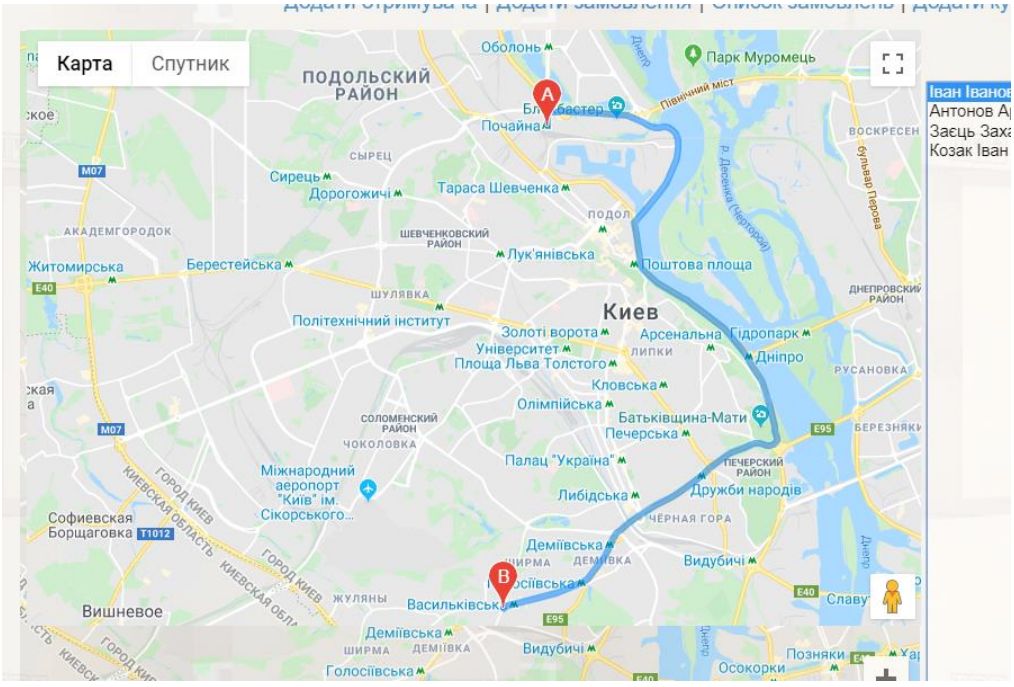

Рисунок 4.5 — Маршрут №3 в системі з використанням мурашиного алгоритму

При побудові автошляху для станцій метро від Почайної до Васильківської розроблена система сформувала маршрут з дистанцією – 17.7 км., час побудови при

цьому збільшилась до 1.5 секунди. Далі був побудований шлях для іншого алгоритму (рисунок 4.6).

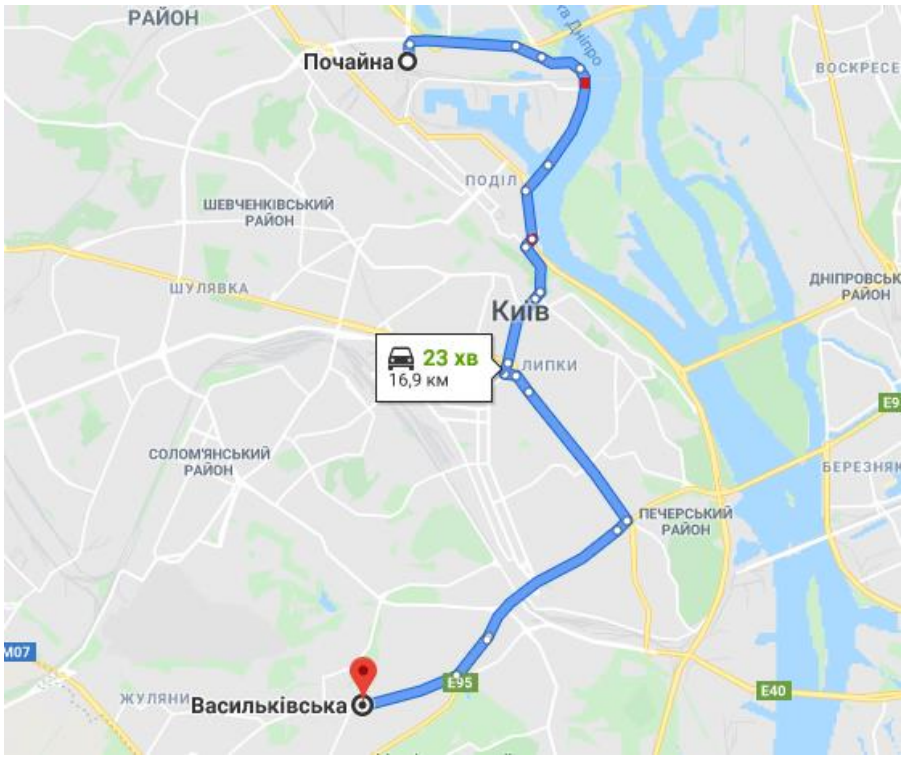

Рисунок 4.6 — Маршрут №3 при пошуку алгоритмом A *

Довжина шляху – 16.9 кілометрів, швидкість побудови – 1.7 секунди.

Як можна побачити, алгоритм A * показав кращу довжину шляху, проте така побудова зайняла трохи більше часу. Це зумовлено невеликою кількістю мурах на маршруті для алгоритму мурахи, при збільшенні кількості мурах та концентрації феромону маршрут стає аналогічним як при побудові алгоритмом A *, однак тривалість пошуку займає 2.9 секунд.

Експеримент №4. Побудова шляху від станції Позняки до станції Шулявська (рисунок 4.7).

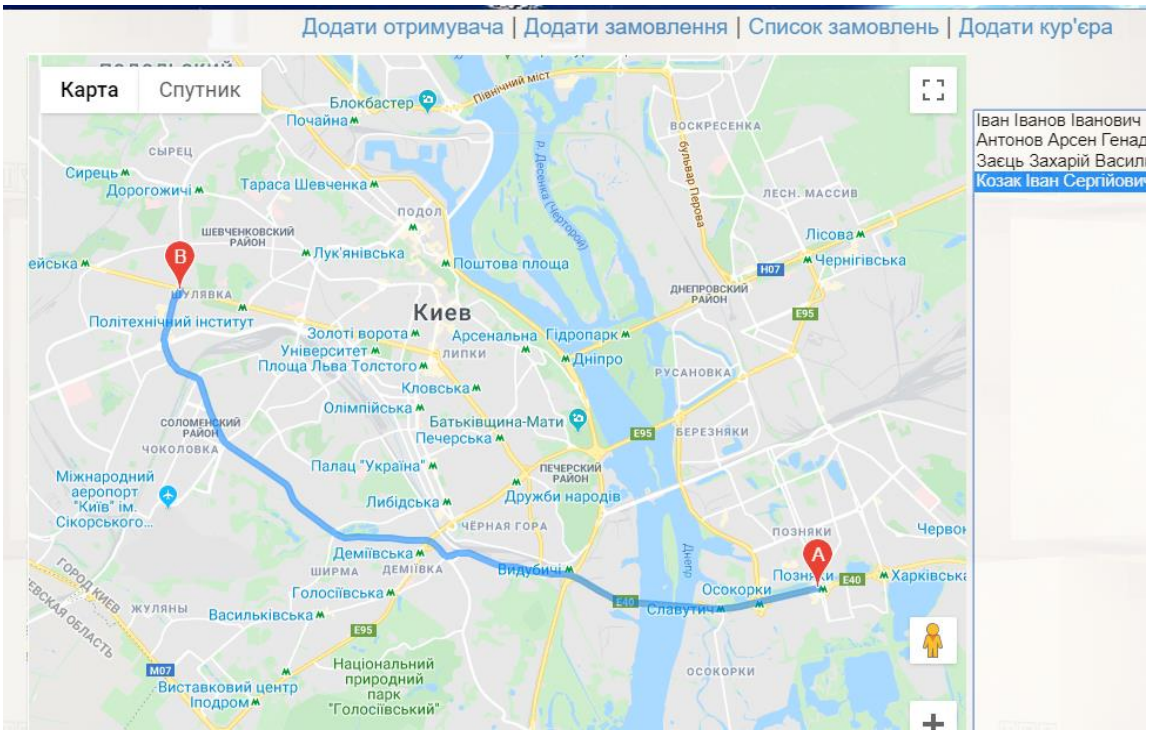

Рисунок 4.7 — Маршрут №4 в системі з використанням мурашиного алгоритму

В результаті отримуємо відстань – 20.8 кілометри, тривалість формування маршруту – 1.7 секунди.

Проведемо аналогічний експеримент для сервісу Google Maps, результат побудови зображений на рисунку 4.8.

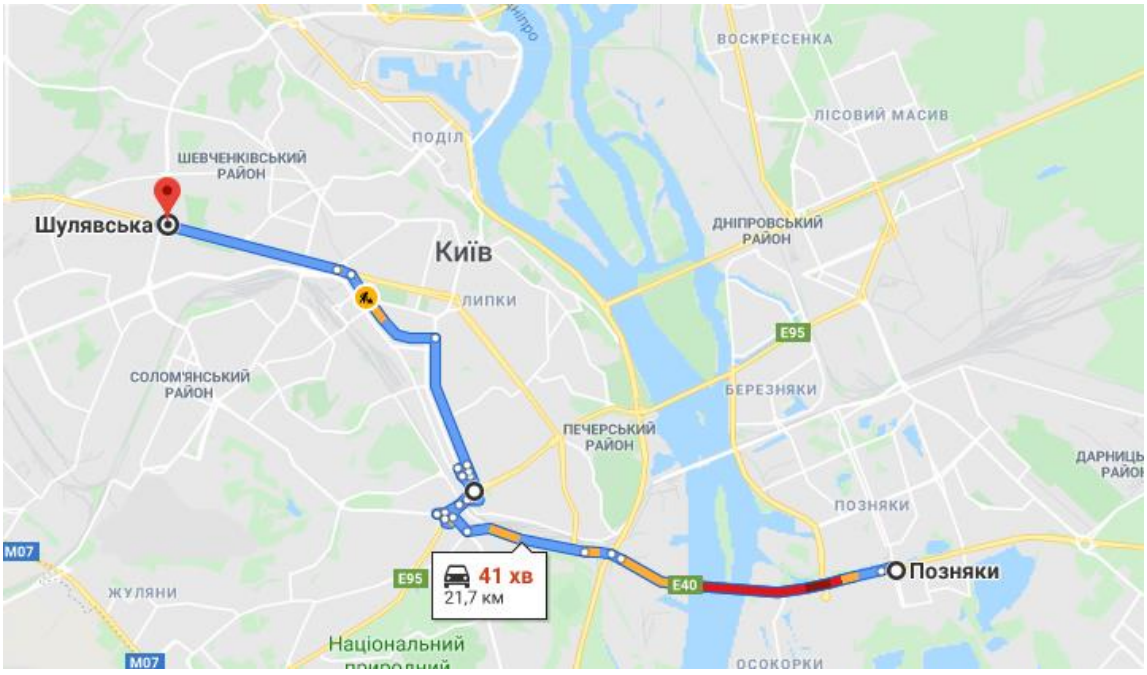

Рисунок 4.8 — Маршрут №4 при пошуку алгоритмом A *

В даному випадку довжина шляху дорівнює 21.7 км., швидкість – 1.6 секунди. Очевидно, що ситуація схожа на попередню. Для алгоритму мурашиної колонії необхідно налаштувати керуючі параметри, зокрема відрегулювати кількість мурах та феромону.

Узагальнені результати дослідження представлені на таблиці 4.1.

Таблиця 4.1 – Опис ідеї стартап-проекту

№	Назва алгоритму	Довжина шляху, км	Час виконання, с
	Мурашиний алгоритм	4,3	0,9
	Алгоритм А *	4,3	0,8
	Мурашиний алгоритм	3,455	1,1
	Алгоритм А *	3,455	1,3
	Мурашиний алгоритм	17,7	1,5
	Алгоритм А *	16,9	1,7
	Мурашиний алгоритм	20,8	1,7
	Алгоритм А *	21,7	1,6

Побудована візуалізація результатів проведених експериментів наведена на рисунку 4.9.

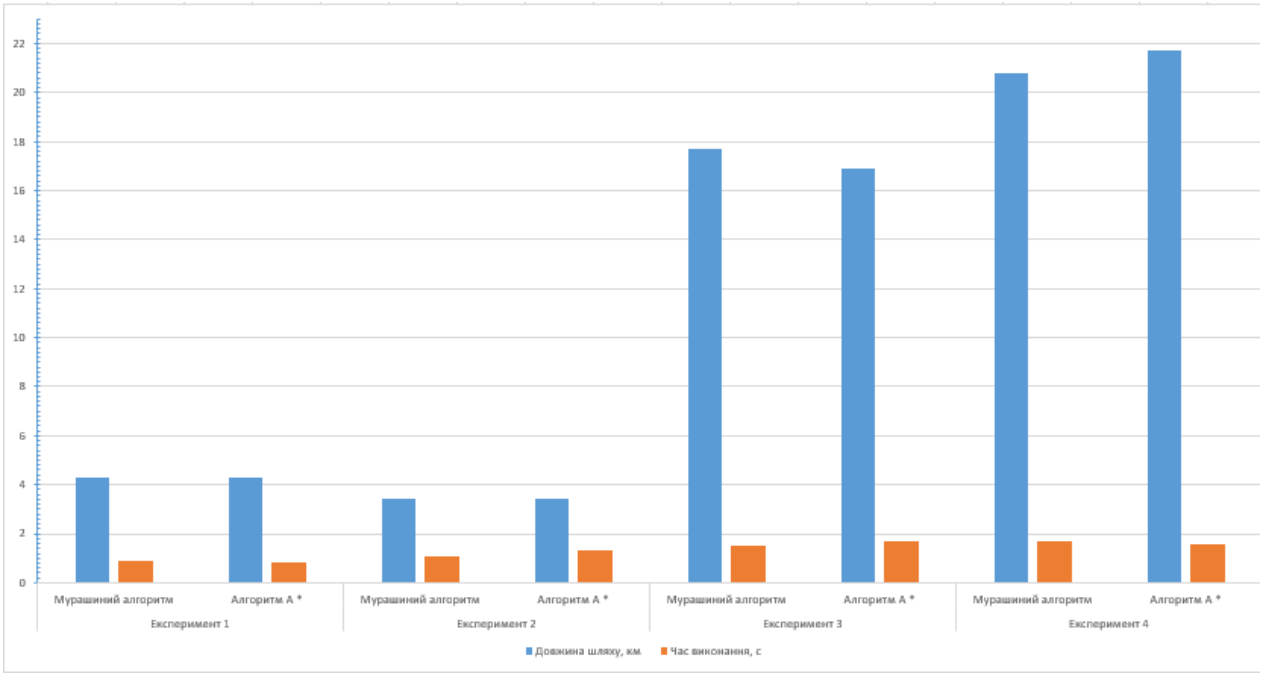

Рисунок 4.9 — Візуалізація результатів експериментів

Висновки до розділу

В рамках даного розділу була досліджена ефективність функціональності побудови маршрутів з використанням мурашиного алгоритму в розробленій системі для кур’єрських перевезень.

Мета даного дослідження була досягнута – система здатна будувати шляхи з меншою відстанню та за менший час ніж алгоритм A * при грамотному підборі параметрів.

В проведеному дослідженні алгоритм застосовувався в реальних умовах для побудови маршрутів між найнаселенішими місцями міста Києва, та порівнювалися результати отриманої довжини маршруту та швидкості побудови в порівнянні з алгоритмом A *, який використовується для генерації маршрутів в популярному сервісі Google Maps.

В залежності від відстані піж пунктами алгоритм повертав кращий результат протяжності маршруту або швидкості виконання в порівнянні з алгоритмом A *. Очевидно, що набір отриманих результатів не є досить великим, однак вже з нього є можливість зробити деякі висновки. Варто відзначити, що характеристики маршруту залежать від налаштувань керуючих параметрів (кількість ітерацій, кількість мурах, концентрація феромону тощо).

Результати експериментів дали змогу дійти до висновку, що мурашиний алгоритм, який використовується в системі побудови маршрутів для кур’єрських перевезень здатний ефективно працювати на коротких та довгих маршрутах при грамотному налаштуванні параметрів побудови.

5 РОЗРОБКА СТАРТАП-ПРОЕКТУ

В час стрімкого впровадження технологій у всі сфери діяльності, активно впроваджуються інтелектуальні системи побудови маршрутів для компаній, які працюють з різними типами перевезень. В даному розділі розглянемо розробку стартап проекту для системи побудови маршрутів.

5.1 Опис ідеї проекту

Таблиця 5.2 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
	1. Побудова маршрутів	Підвищення якості побудови маршрутів
	2. Зменшення впливу людського фактору	Захищеність інформаційної системи
	3. Підвищення ефективності складання маршрутів	Скорочення часу перевезення

Конкурентами є аналогічні методи та механізми побудови маршрутів кур’єрських перевезень. Основною відмінністю є те, що система побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів реалізується таким чином, щоб забезпечити (по можливості) необхідну швидкість передачі повідомлень різних типів з урахуванням їх цінності.

Довгостроковими перспективами є:

а) збільшення кількості клієнтів, що будуть використовувати запропоновані методи.

б) додавання новітніх механізмів побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів.

Потреби в стартовому фінансуванні: стартовий капітал = 5000 грн

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Zig-Zag	Aurama	«Мурашина логістика»			
1.	Вартість ПЗ	Низька	Висока	Висока	Висока			+
2.	Швидкість побудови маршруту	Середня	Висока	Низька	Середня			+
3.	Багатоплатформенність	Ні	Так	Ні	Так			+
4.	Сучасність дизайну	Так	Так	Так	Ні		+	
5.	Підтримка	+	+	-	+		+	

5.2 Технологічний аудит проекту

Таблиця 5.3 –Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
		Мурашині алгоритми		
		Можливість сегментації ділянок маршруту		
Для реалізації були обрані мови C# та JavaScript та алгоритм мурашки				

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Встановлення можливостей ринку, для їх використання в процесі ринкового впровадження проекту, та загроз, які створять додаткові труднощі при реалізації

проекту дасть можливість спланувати стратегію розвитку проекту із урахуванням характеристик ринкового середовища, потреб потенційних споживачів та переваг проектів-конкурентів.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку(найменування)	Характеристика
1	Кількість конкурентних продуктів на ринку, од	4
2	Загальний обсяг продаж, грн/ум.од	312000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу(вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середнє значення рентабельності в галузі(або по ринку), %	30%

На основі проведеного аналізу є можливість стверджувати про привабливість проекту системи побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів для входження на ринок за попереднім оцінюванням.

За результатами дослідження, представленого в таблиці 5.4 можна зробити висновок, що є можливість розповсюджувати систему на ринку як стартап-проект.

Далі розглянемо типи потенційних клієнтів та перелік вимог до системи в залежності від типу (таблиця 5.5).

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
	Побудова маршрутів для перевезень	Кур’єрські служби, логістичні компанії	Потенційні клієнти мають різний поточний та потенційний дохід при використанні даної системи, деякі компанії можуть бути	Швидкість та комфорт при роботі в системі, наявність довідкових матеріалів

			зацікавленні в підтримці вітчизняного виробника	

Дослідивши потенційних клієнтів потрібно визначити фактори які загрожують та сприяють реалізації стартап-проекту (табл. 5.6, 5.7)

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція	Наявність та поява систем з досконалішими функціями та характеристиками.	Поступове вдосконалення та пришвидшення роботи системи. Зосередження уваги на основних перевагах системи в порівнянні з конкурентами. Програми лояльності для постійних клієнтів.
2	Маловідомість системи	Маловідомість системи та компанії для потенційних клієнтів	Ефективні маркетингові ходи, реклама, активне розповсюдження системи серед логістичних та кур’єрських компаній

Продовження таблиці 5.6

3	Несанкціоноване втручання в роботу системи	Оскільки система працює в глобальній мережі, існує можливість втручання в роботу системи і нанесення збитків як зі сторони агресивних конкурентів, так і інших зацікавлених осіб	Акцентування уваги при розробці насамперед на безпеку. Використання захищених протоколів зв’язку та сертифікатів.

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Залучення іноземних та вітчизняних інвесторів	Залучення інвесторів та спонсорів.	Розширення діяльності компанії в інші міста та країни, тобто вихід на нові ринки
2	Поява оптимальніших методів побудови маршрутів	Розробляться кращі та швидші методи побудови маршрутів	Спрощене та продуктивніше користування системою для клієнтів
3	Партнерство	Партнерство з компаніями, які володіють системами для інтеграції з розробленою системою	Розширення функціональності системи. Інтеграція з іншими гравцями на ринку.

Надалі були досліджені характеристики конкуренції на ринку (таблиця 5.8).

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому виражена дана характеристика	Вплив на діяльність компанії (можливі дії для підвищення конкурентоспроможності)
1. Вказати тип конкуренції - монополістична	На ринках відсутні бар'єри для входу та виходу, також наявні продукти з кращими та гіршими характеристиками.	Неперервне покращення та оновлення якості та функціональності системи. Залучення інвесторів та спонсорів.
2. За рівнем конкурентної боротьби - міжнародний	Наявна значна кількість конкурентів на глобальному ринку	Вдосконалення системи для її впровадження на ринках інших країн, зменшення ціни.
3. За галузевою ознакою - внутрішньогалузева	Система використовується в різних сферах логістики	Організація системи таким чином, щоб вона оптимально працювала і для кур'єрських служб, компаній з міжнародних перевезень та інших
4. Конкуренція за видами товарів: - товарно-видова	Конкуренція між методами побудови маршрутів та особливостями їх роботи	Дослідження та покращення алгоритму для побудови маршрутів враховуючи при цьому недоліки методів, які використовують конкуренти
5. За характером конкурентних переваг - цінова	Застосування найсучасніших методів створення продукту, зменшення витрат, програми лояльності для клієнтів	Зменшення ціни в порівнянні з іноземними аналогами та спеціальні умови для стратегічних клієнтів. Надання переваги доступним та відкритим технологіям в порівнянні з продуктами конкурентів.
6. За інтенсивністю - не марочна	Бренд має місце, але його значення не є суттєвим	Участь в різних тематичних заходах (презентації, конференції, тури та інші), реклама

Надалі проаналізуємо конкуренцію у вибраній галузі використовуючи модель п’яти конкурентних сил Майкла Портера (табл. 5.9).

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Навести перелік безпосередніх конкурентів	Визначити бар’єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили клієнтів	Фактори загроз з боку товарів-замінників
	«Мурашина логістика»	Існуючі рішення	-	Функціональність системи та її асортимент	Впізнаваний продукт з хорошою підтримкою
Висновки:	На цільовому ринку спостерігається тенденція до скорочення кількості підприємств і посилення конкуренції на ринку. Вступ України до СОТ відкрив дорогу іноземним виробникам. Великі компанії з іноземним капіталом постійно	Бар'єри входу на ринок є порівняно незначними. Вартість організації бізнесу з виробництва сучасних механізмів побудови маршрутів кур’єрських перевезен	Існує чітка залежність від постачальника та якості продукції. Також ціна кінцевої продукції залежить від якості та асортименту	Споживачі мають широку географію і проживають переважно у містах. Придбання програмних додатків та алгоритмів реалізації механізмів підвищення ефективності побудови маршрутів	Посилилася конкуренція зі сторони товарів субститутів – інших видів, методів та алгоритмів побудови маршрутів для перевезень за рахунок збільшення асортименту останніх та появи нових для ринку категорій

	збільшують контрольовану ними частку ринку, поглинаючи конкурентів	ь безпеки сягає 100 тис. дол.		перевезень часто носить імпульсний характер	

У відповідності до наведеного дослідження, головними силами, які впливають на конкуренцію в галузі є постачальники та споживачі. Також в силу розвитку ринку все більшого значення набуває інтенсивність конкуренції між існуючими конкурентами та загроза зі сторони товарів-субститутів.

Таким чином в межах структурного підходу до аналізу конкуренції тип конкуренції – монополістична конкуренція. Далі проаналізуємо фактори конкурентоспроможності розроблюваної системи (табл. 5.10).

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування вибору
1	Частка ринку	Враховуючи той факт, що тип родового середовища в галузі – консолідований ринок, тобто існує група компаній, які контролюють разом понад 40% ринку, а також те, що інтенсивність суперництва між діючими конкурентами при низьких темпах зростання ринку є однією з головних сил, які діють на конкуренцію в галузі, одним з найважливіших факторів конкурентоспроможності виступає частка ринку, яку займає виробник. В таких умовах чим більше частка ринку, тим більшими ринковими можливостями володіє виробник.
2	Ціна	Чим вигіднішою є ціна для споживача, тим більша ймовірність виробу продукту
3	Асортимент	В умовах збільшення напруги між існуючими конкурентами завоювання споживачів відбувається за рахунок нових методів та алгоритмів.

Продовження таблиці 5.10

4	Доступ до каналів розподілу	Споживач далеко не завжди проявляє прихильність до певної категорії розробників і дуже схильний до експериментів. В цьому випадку завоювати лояльність споживача дуже складно і ще складніше її утримати. Тому для компаній виробників ключовими чинниками успіху стає сильна дистрибуція, якісний торговий маркетинг і налагоджена система логістики.
5	Торговий маркетинг	
6	Репутація виробника	Якщо компанія має бездоганну репутацію, особливо у сфері якості своєї продукції, то рівень довіри до неї зростає. Також репутація виробника важлива при виході на ринок з новими товарами, або при виході на нові сегменти, що полегшує позитивне сприйняття новинок.
7	Рівень лояльності до бренду	Чим вище рівень лояльності, тим більше компанія має прихильних, а значить постійних споживачів.
8	Маркетинговий бюджет	Від розміру маркетингового бюджету залежить здатність здійснювати маркетингову стратегію підприємства. Маркетингові заходи мають забезпечувати інші конкурентні переваги такі, як рівень диференціації, лояльності, репутація виробника, дистрибуція та просування в торгових точках.

На основі визначених факторів конкурентоспроможності (табл. 5.10) проаналізуємо слабкі та сильні сторони розробленого проекту (табл. 5.11).

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін системи

№ п/п	Фактор конкурентоспроможності	Бали (1-20)	Рейтинг конкурентів у порівнянні з власною системою						
			-3	-2	-1	0	1	2	3
1	Частка ринку	18				+			
2	Ціна	10					+		
3	Асортимент	18			+				
4	Доступ до каналів розподілу	15					+		
5	Торговий маркетинг	15						+	

Продовження таблиці 5.11

6	Репутація виробника	12				+			
7	Рівень лояльності до бренду	14					+		
8	Маркетинговий бюджет	10				+			

Завершальним етапом дослідження доцільності реалізації системи побудови маршрутів з використанням мурашиних алгоритмів як стартап-проекту буде формування матриці SWOT-аналізу (табл. 5.12), яка описує сильні, слабкі сторони та загрози і можливості. Дана матриця формуватиметься на основі списку маркетингових загроз та можливостей, які є наслідками впливу ринкових факторів.

Таблиця 5.12 — SWOT-аналіз стартап-проекту

Сильні сторони: Собівартість, наявність вертикальної інтеграції, ціна, спрямованість на споживача	Слабкі сторони: Відсутність чітко вираженої маркетингової стратегії, непослідовність в її реалізації, слабке самозабезпечення фінансовими ресурсами, залежність від сторонніх продуктів
Можливості: Зміцнення іміджу, збільшення обсягів реалізації, збільшення обсягів продаж за рахунок експансії в регіони, відсутність агресивної конкуренції на українському ринку	Загрози: Загроза працювати без прибутку за скорочення платоспроможного попиту, втрата споживачів внаслідок підвищення тиску зі сторони товарів-субститутів, підвищення цін, зміна потреб клієнта

З отриманого SWOT-аналізу можна побачити, що найбільш негативний вплив на діяльність системи на ринку чинить ринкове середовище. Це, перш за все, пов’язано із наслідками фінансово-економічної кризи в країні.

В свою чергу, така ситуація супроводжувалася зменшенням темпів приросту галузі, виходом з ринку менш сильних дрібних та регіональних виробників, приходом

на ринок транснаціональних компаній, що збільшило інтенсивність конкуренції між діючими учасниками ринку України. Було визначено, що найбільшою загрозою для проекту є загроза падіння прибутковості внаслідок скорочення попиту.

Сформуємо альтернативи ринкового впровадження проекту виділивши ймовірність отримання ресурсів та строки реалізації (таблиця 5.13).

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Використання засобів стимулювання збуту та мерчандайзингу для збільшення продаж	Дозволяє суттєво збільшити обсяги продаж	до 6 місяців
2.	Розширення асортиментної лінійки	Можливість залучення нових споживачів за рахунок новинки	до 1 року
3.	Безкоштовне надання обмеженої функціональності системи	Можливість залучення клієнтів за рахунок їх ознайомлення з роботою та перевагами системи	до 1 року

В результаті аналізу описаних критеріїв була обрана третя альтернатива.

5.4 Розроблення ринкової стратегії проекту

Для розробки ефективної ринкової стратегії потрібно передусім виділити стратегії охоплення ринку, тобто виділити групи потенційних клієнтів (табл. 5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Компанії, які займаються вантажними перевезеннями	Готові	Середній	Середня	Середня
2	Кур’єрські служби	Готові	Високий	Висока	Середня
Обрані цільові групи клієнтів: Кур’єрські служби та компанії перевізники					

В результаті дослідження потенційних груп клієнтів (сегментів) була обрана стратегія охоплення ринку - стратегія диференційованого маркетингу, в якій компанії співпрацюють з кількома сегментами ринку.

Виділимо базову стратегію розвитку стартап-проекту (таблиця. 5.15).

Таблиця 5.15 – Опис базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку стартап-проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	Стратегія диференціації	Дозволяє надати продукту необхідні для споживача властивості, які відрізнять товар від конкурентних товарів. Ця відмінність може формуватися з використанням об’єктивних або суб’єктивних,	Такий тип стратегії потребує зазвичай більших витрат. Однак правильна диференціація дасть можливість домогтись кращої рентабельності в умовах готовності ринку прийняти вищу ціну	Інструментом реалізації стратегії диференціації є ринкове позиціонування

		відчутних і невідчутних властивостей продукту (зазвичай у комплексі маркетингу), може мати вигляд уявної або реальної		

Також визначимо базову стратегію конкурентної поведінки для стартап-проекту (таблиця 5.16).

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є система першою подібною на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Ні	Залучати нових та забирати існуючих у конкурентів	Частково	Наслідування лідеру

Використовуючи результати проведених раніше досліджень визначимо стратегію позиціонування (таблиця 5.17), яка дозволить виділити ринкову позицію для ідентифікації проекту на ринку.

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Відповідність чинним нормам, побудова оптимального маршруту, сучасний та зручний інтерфейс	Наслідування лідеру	Дана стратегія потребує значні вклади фінансових ресурсів. При наданні продукту унікальних характеристик ринок зможе прийняти такий продукт за більшу вартість.	Унікальність; Доступна ціна; Реалізація нових методів; Економія часу

5.5 Розроблення маркетингової програми стартап-проекту

Для створення маркетингової програми проекту сформуємо передусім маркетингову концепцію товару, який буде надано клієнту (таблиця 5.18). Для цього використаємо результати вже розглянутого аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує система	Основні переваги перед продуктами конкурентами
1.	Побудова маршрутів	Підвищення якості складання маршрутів	Високий рівень якості маршрутів
2.	Зменшення впливу людського фактору	Захищеність інформаційної системи	Високий рівень оцінювання ефективності перевезень
3.	Підвищення ефективності складання маршрутів	Скорочення часу перевезення	Якість перевезень

Далі опишемо трирівневу маркетингову модель продукту, яка буде в собі містити: мету продукту, фізичну складову продукту та нюанси надання продукту (таблиця 5.19).

Перший рівень відповість на питання основного призначення та вигоди розроблюваної системи. Другий рівень – реалізацію продукту, посилаючись на поточне ринкове середовище (ціна, дизайн тощо). Третій – додаткові переваги для користувача, які формуються на основі товару в дійсній реалізації та товару за задумом (доставка, умови оплати, програми лояльності тощо).

Таблиця 5.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="3">Інформаційна система, яка дозволить виконувати побудову оптимальних маршрутів з використанням мурашиного алгоритму</td></tr>
<tr><td rowspan="5"></td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Хороша якість побудови маршрутів
2. Сучасний та інтуїтивно зрозумілий інтерфейс системи</td><td></td><td></td></tr>
<tr><td colspan="3">Пакування – без пакування</td></tr>
<tr><td colspan="3">Марка: назва організації-розробника + назва системи</td></tr>
<tr><td colspan="3">До продажу – рівень розробки</td></tr>
<tr><td>III. Товар із підкріпленням</td><td colspan="3">Після продажу – низка методів та алгоритмів</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання:
захист інтелектуальної власності, патентування.</td></tr>
</table>

Визначимо діапазони встановлення цін на продукт, враховуючи ціни на аналогічні продукти та доходи потенційних категорій клієнтів (таблиця 5.20).

Таблиця 5.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	300-1100$	300-1000$	2000$ і більше	250-750$

Оберемо систему збуту для обраного товару, за якої буде виконуватись розповсюдження системи (таблиця 5.21).

Таблиця 5.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Для даної системи цільовими клієнтами будуть компанії, яким потрібна оптимальна побудова маршрутів на дорогах	Організовувати широку мережу збуту товару, збільшувати попит та стимулювати продажі. Використовувати різні типи реклами для збільшення кількості потенційних клієнтів.	Один (продукт буде надаватись одразу клієнту)	Продаж системи через сайт виробника продукту

І кінцевим етапом буде обрання концепції маркетингових комунікацій (таблиця 5.22).

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки потенційних клієнтів	Канали комунікацій потенційних клієнтів	Ключові особливості позиціювання проекту	Ідея рекламної політики	Концепція рекламного звернення
1	Потреба клієнта отримати оптимальний маршрут для	Презентації, вебінари, контекстна реклама, конференції	Підвищення якості складання маршрутів	Описати переваги системи, цінову політику	Продемонструвати переваги перед продуктами конкурентами

	перевезення вантажу			та додаткові умови	

В результаті аналізу ми отримали ефективну ринкову програму яка сформована на основі цінності та потребах потенційних клієнтів та альтернативу ринкової економіки.

Висновки до розділу

У даному розділі проводився аналіз, який допоможе реалізувати систему побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів як стартап-проект.

Оскільки в галузі логістики та перевезень все частіше використовуються інформаційні технології та інтелектуальні системи, і відповідно кількість потенційних клієнтів зростає з кожним роком, то в таких умовах подальший розвиток проекту є доцільним.

Як наслідок, в розділі був розроблений стартап-проект для системи, маркетингова стратегія, ціноутворення, виділені основні конкуренти та був розроблений процес для ефективного виходу на ринки.

Для реалізації системи на ринку була обрана стратегія, яка передбачає надання обмеженої функціональності клієнту, який після ознайомлення та визначення переваг системи купує повну функціональність за певну оплату.

Також не слід забувати про конкуренцію, яка змушує обережно та розумно проводити процес просування продукту.

ВИСНОВКИ

В даній магістерській дисертації була спроектована та розроблена система побудови маршрутів кур’єрських перевезень з використанням мурашиних алгоритмів.

При дослідженні мурашиного алгоритму були розглянуті біологічні основи переміщення мурах в середовищі, типи розширень алгоритму, розібраний приклад ітерації руху та математична модель, визначений алгоритм переміщення. При цьому були проаналізовані відомі способи пошуку на графах при наявності великої кількості альтернативних шляхів.

Для реалізації системи була обрана стандартна клієнт-серверна архітектура та сучасні методи створення веб-застосунків від корпорації Microsoft, зокрема ASP.NET MVC, Entity Framework та Microsoft SQL Server. Безпосередня побудова маршрутів розроблена на клієнтській стороні з використанням сервісу «Google карти» та прикладним програмним інтерфейсом «Google Maps JavaScript API», який дозволив маніпулювати перебігом відображення маршруту.

Функціональність розробки була перевірена на практиці та задокументована з використанням тестових сценаріїв.

Окрім того, в розділі стартап проекту були досліджені ризики та сформовані шляхи для виходу і розповсюдження системи на ринку. В результаті аналізу можна підтвердити, що така система має потенціал до розвитку її як повноцінний стартап проект, оскільки в сфері перевезень відбувається неперервне впровадження новітніх технологій.

В результаті розробки система побудови маршрутів з використанням мурашиних алгоритмів реалізована відповідно до поставленого завдання на магістерську дисертацію.

СПИСОК ПОСИЛАНЬ

1. Ермолаев С.Ю. Муравьиные алгоритмы оптимизации // Инфокоммуникационные технологии. – 2017. – т. 6. – № 1 – С. 55-59.

2. Кажаров А.А., Курейчик В.М. Использование шаблонных решений в муравьиных алгоритмах. Известия Южного федерального университета. Технические науки. 2013. – № 7 (144). – С. 17-22.

3. Кажаров А.А., Курейчик В.М. Муравьиные алгоритмы для решения транспортных задач. Известия Российской академии наук. Теория и системы управления. 2016. – № 1. – С. 32-45.

4. Кажаров А.А., Курейчик В.М. О некоторых модификациях муравьиного алгоритма. Известия Южного федерального университета. Технические науки. 2018. – Т. 81. – № 4. – С. 7-12.

5. Кирсанов М. Н. Графы в Maple. М.: Физматлит, 2017. – 168 с. http://vuz.exponenta.ru/PDF/book/GrMaple.pdfhttp://eqworld.ipmnet.ru/ru/library/books/Kirsanov2017ru.pdf

6. Курейчик В.М., КажаровА.А. О некоторых модифи-кациях муравьиного алгоритма Известия ЮФУ. Технические науки. Тематический выпуск «Интеллектуальные САПР». – Таганрог: Изд-во ТТИ ЮФУ, 2018. – № 4(81). – 268 с.

7. Левитин А. Алгоритмы. Введение в разработку и анализ, 2016. – 453 с.

8. М. Тим Джонс. Программирование искусственного интеллекта в приложениях – ДМК Пресс, 2014.

9. Малыхина М.П. Базы данных: основы, проектирование, использование: учебное пособие. – 2-е изд. – СПб.: БХВ-Петербург, 2016. – 528 с.

10. Малыхина М.П., Частикова В.А., Власов К.А. Исследование эффективности работы модифицированного генетического алгоритма в задачах комбинаторики // Современные проблемы науки и образования. – 2013. – № 3; URL: www.science-education.ru/109-9254

11. Павленко А.И., Титов Ю.П. Метод муравьиной оптимизации в задачах распределения ресурсов – МАИ, 2018.

12. Рассел С. Дж., Норвиг, П. Искусственный интеллект: современный подход – М.: Вильямс, 2016.

13. С.Д. Штовба. Муравьиные алгоритмы. http://www.serhiy-shtovba.narod.ru/doc/Shtovba_Ant_Algorithms_ExponentaPro_2013_3.pdf

14. Субботин С.А., Олейник Ан.А., Олейник Ал.А. Интеллектуальные мультиагентные методы (swarm intelligence) – http://csit.narod.ru/subject/MA/MA_lect.pdf

15. Маланова Т.В., Серебрянска Н.С.. Сравнительный анализ алгоритмов муравья. – УДК 004.032.026:004.3

16. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн. Алгоритмы. Построение и анализ – М. «Вильямс», 2015.

17. Частикова В.А., Власов К.А. Свидетельство о государственной регистрации программы для ЭВМ № 2011616886. Программный комплекс для исследования и сравнительного анализа работы модифицированного генетического алгоритма: зарегистрировано в Реестре программ для ЭВМ 6 сентября 2011.

18. Чжо Мьо Хан, Диссертационная работа на тему «Планирование движения маршрута городского транспорта» – МАИ 2010.

19. Чураков М., Якушев А. Муравьиные алгоритмы [Электронный ресурс] – Режим доступа: http://rain.ifmo.ru/cat/data/theory/unsorted/ant-algo-2016/article.pdf

20. Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2014. – № 4 – 322 с.

21. Bonabeau, E. «Editor’s Introduction: Stigmergy.» special Issue of Artificial Life on Stigmergy. Volume 5, Issue 2 / Spring 2016, p.95-96.http://www.stigmergicsystems.com/stig_v1/stigrefs/article1.html

22. Blum C., 2015 «Ant colony optimization: Introduction and recent trends». Physics of Life Reviews, 2: 353-373

23. Colorni, M. Dorigo et V. Maniezzo, Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.

24. E. Bonabeau, M. Dorigo et G. Theraulaz, 1999. Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press.

25. F. Moyson, B. Manderick, The collective behaviour of Ants : an Example of Self-Organization in Massive Parallelism, Actes de AAAI Spring Symposium on Parallel Models of Intelligence, Stanford, Californie, 1988.

26. J.L. Denebourg, J.M. Pasteels et J.C. Verhaeghe, Probabilistic Behaviour in Ants : a Strategy of Errors?, Journal of Theoretical Biology, numero 105, 1983.

27. J.-L. Deneubourg, S. Aron, S. Goss et J.-M. Pasteels, The self-organizing exploratory pattern of the Argentine ant, Journal of Insect Behavior, volume 3, page 159, 1990

28. L. Bianchi, L. M. Gambardella et M. Dorigo, An ant colony optimization approach to the probabilistic traveling salesman problem, PPSN-VII, Seventh International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Springer Verlag, Berlin, Allemagne, 2002.

29. M. Dorigo & T. Stützle, 2004. Ant Colony Optimization, MIT Press.

30. M. Dorigo 1997. «Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem». IEEE Transactions on Evolutionary Computation, 1 (1): 53-66.

31. M. Dorigo et L.M. Gambardella, Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation, volume 1, numero 1, pages 53-66, 1997.

32. M. Dorigo, 2007. «Ant Colony Optimization». Scholarpedia.

33. M. Dorigo, G. Di Caro & L. M. Gambardella, 1999. «Ant Algorithms for Discrete Optimization». Artificial Life, 5 (2): 137-172.

34. M. Dorigo, M. Birattari & T. Stützle, 2006 Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique.

35. M. Dorigo, V. Maniezzo & A. Colorni, 1996. «Ant System: Optimization by a Colony of Cooperating Agents», IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26 (1): 29-41.

36. M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, 1992.

37. M. Ebling, M. Di Loreto, M. Presley, F. Wieland, et D. Jefferson,An Ant Foraging Model Implemented on the Time Warp Operating System, Proceedings of the SCS Multiconference on Distributed Simulation, 1989

38. P.-P. Grasse, La reconstruction du nid et les coordinations inter-individuelles chez Belicositermes natalensis et Cubitermes sp. La theorie de la Stigmergie : Essai d'interpretation du comportement des termites constructeurs, Insectes Sociaux, numero 6, p. 41-80, 1959.

39. S. Goss, S. Aron, J.-L. Deneubourg et J.-M. Pasteels, The self-organized exploratory pattern of the Argentine ant, Naturwissenschaften, volume 76, pages 579-581, 1989

40. S. Iredi, D. Merkle et M. Middendorf, Bi-Criterion Optimization with Multi Colony Ant Algorithms, Evolutionary Multi-Criterion Optimization, First International Conference (EMO'01), Zurich, Springer Verlag, pages 359-372, 2001.

41. T. Stützle et H.H. Hoos, MAX MIN Ant System, Future Generation Computer Systems, volume 16, pages 889-914, 2000

Додаток А. Вихідний код функції пошуку з використанням мурашиного алгоритму

```
function solveWithAntAlgorithm() {
    var option = 1;
    var q = 0.1;
    var p = 2.0;
    var pRate = 0.1;
    var rSharp = 0.9;
    var pheromoneArray = new Array();
    var nextPheromoneArray = new Array();
    var probabilityArrayabilityArray = new Array();
    var antAmount = 20;
    var iterationAmount = 20;
    for (var i = 0; i < locationAmount; ++i) {
        pheromoneArray[i] = new Array();
        nextPheromoneArray[i] = new Array();
    }
    for (var i = 0; i < locationAmount; ++i) {
        for (var j = 0; j < locationAmount; ++j) {
            pheromoneArray[i][j] = 1;
            nextPheromoneArray[i][j] = 0;
        }
    }

    var lastEdge = 0;
    var startEdge = 0;
    var amountOfSteps = locationAmount - 1;
    var amountOfCorrentDestinations = locationAmount;
    if (option == 1) {
        lastEdge = locationAmount - 1;
        amountOfSteps = locationAmount - 2;
        amountOfCorrentDestinations = locationAmount - 1;
    }
    for (var iteration = 0; iteration < iterationAmount; ++iteration) {
        for (var ant = 0; ant < numAnts; ++ant) {
            var current = startEdge;
            var currentDist = 0;
            for (var i = 0; i < locationAmount; ++i) {
                isVisited[i] = false;
            }
            currentWay[0] = current;
            for (var step = 0; step < amountOfSteps; ++step) {
                isVisited[current] = true;
                var cumprobabilityArray = 0.0;
                for (var next = 1; next < amountOfCorrentDestinations; ++next) {
                    if (!isVisited[next]) {
                        probabilityArray[next] =
Math.pow(nextPheromoneArray[current][next], q) *
                            Math.pow(dur[current][next], 0.0 - p);
                        cumprobabilityArray += probabilityArray[next];
                    }
                }
                var guess = Math.random() * cumprobabilityArray;
                var nextI = -1;
                for (var next = 1; next < amountOfCorrentDestinations; ++next) {
                    if (!isVisited[next]) {
                        nextI = next;
                        guess -= probabilityArray[next];
                        if (guess < 0) {
                            nextI = next;
                            break;
```

```
                    }
                }
            }
            currentDist += dur[current][nextI];
            currentWay[step + 1] = nextI;
            current = nextI;
        }
        currentWay[amountOfSteps + 1] = lastEdge;
        currentDist += dur[current][lastEdge];

        var lastStep = locationAmount;
        if (option == 1) {
            lastStep = locationAmount - 1;
        }
        var changed = true;
        var i = 0;
        while (changed) {
            changed = false;
            for (; i < lastStep - 2 && !changed; ++i) {
                var cost = dur[currentWay[i + 1]][currentWay[i + 2]];
                var revCost = dur[currentWay[i + 2]][currentWay[i + 1]];
                var iCost = dur[currentWay[i]][currentWay[i + 1]];
                var tmp, nowCost, newCost;
                for (var j = i + 2; j < lastStep && !changed; ++j) {
                    nowCost = cost + iCost + dur[currentWay[j]][currentWay[j +
1]];
                    newCost = revCost + dur[currentWay[i]][currentWay[j]] +
                        dur[currentWay[i + 1]][currentWay[j + 1]];
                    if (nowCost > newCost) {
                        currentDist += newCost - nowCost;
                        for (var k = 0; k < Math.floor((j - i) / 2); ++k) {
                            tmp = currentWay[i + 1 + k];
                            currentWay[i + 1 + k] = currentWay[j - k];
                            currentWay[j - k] = tmp;
                        }
                        changed = true;
                        --i;
                    }
                    cost += dur[currentWay[j]][currentWay[j + 1]];
                    revCost += dur[currentWay[j + 1]][currentWay[j]];
                }
            }
        }

        if (currentDist < bestWay) {
            bestPath = currentWay;
            bestWay = currentDist;
        }
        for (var i = 0; i <= amountOfSteps; ++i) {
            nextPheromoneArray[currentWay[i]][currentWay[i + 1]] += (bestWay -
rSharp * bestWay) / (numAnts * (currentDist - rSharp * bestWay));
        }
    }
    for (var i = 0; i < locationAmount; ++i) {
        for (var j = 0; j < locationAmount; ++j) {
            nextPheromoneArray[i][j] = nextPheromoneArray[i][j] * (1.0 - rho) +
rho * nextPheromoneArray[i][j];
            nextPheromoneArray[i][j] = 0.0;
        }
    }
}
```

Додаток Б. Вихідний код контролера автентифікації

```
namespace CourierRouteBuilder.WEB.Controllers
{
    [Authorize]
    public class AccountController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationUserManager _userManager;
        private ApplicationRoleManager _roleManager;

        public AccountController()
        {
        }

        public          AccountController(ApplicationUserManager          userManager,
ApplicationSignInManager signInManager )
        {
            UserManager = userManager;
            SignInManager = signInManager;
        }

        public ApplicationSignInManager SignInManager
        {
            get
            {
                return                          _signInManager                          ??
HttpContext.GetOwinContext().Get<ApplicationSignInManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }

        public ApplicationUserManager UserManager
        {
            get
            {
                return                           _userManager                           ??
HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();
```

```
            }
            private set
            {
                _userManager = value;
            }
        }

        public ApplicationRoleManager RoleManager
        {
            get
            {
                return                          _roleManager                          ??
HttpContext.GetOwinContext().Get<ApplicationRoleManager>();
            }
            private set
            {
                _roleManager = value;
            }
        }

        [AllowAnonymous]
        public ActionResult Login()
        {
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            var   result   =   await   SignInManager.PasswordSignInAsync(model.Email,
model.Password, model.RememberMe, shouldLockout: false);

            switch (result)
            {
                case SignInStatus.Succe
```

```
                    if  (UserManager.IsInRole(UserManager.FindByName(model.Email).Id,
"Operator"))
                    {
                        return RedirectToRoute(new { controller = "Work", action =
"Index" });
                    } else
                    {
                        return RedirectToRoute(new { controller = "CourierMain",
action = "Index" });
                    }
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                    return RedirectToAction("SendCode", new { ReturnUrl = returnUrl,
RememberMe = model.RememberMe });
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Invalid login attempt.");
                    return View(model);
            }
        }

        [AllowAnonymous]
        public ActionResult Register()
        {
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser { UserName = model.Email, Email =
model.Email };
                var result = UserManager.Create(user, model.Password);

                if (result.Succeeded)
                {
                    UserManager.AddToRole(UserManager.FindByName(model.Email).Id,
"Operator");
```

```
                    await     SignInManager.SignInAsync(user,     isPersistent:false,
rememberBrowser:false);

                    return RedirectToAction("Index", "Work");
                }
                AddErrors(result);
            }

            return View(model);
        }

        [AllowAnonymous]
        public async Task<ActionResult> ConfirmEmail(string userId, string code)
        {
            if (userId == null || code == null)
            {
                return View("Error");
            }
            var result = await UserManager.ConfirmEmailAsync(userId, code);
            return View(result.Succeeded ? "ConfirmEmail" : "Error");
        }

        [AllowAnonymous]
        public ActionResult ForgotPassword()
        {
            return View();
        }

        [AllowAnonymous]
        public ActionResult ResetPassword(string code)
        {
            return code == null ? View("Error") : View();
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)
        {
            if (!ModelState.IsValid)
            {
                return View(model)
```

```
            }
            var user = await UserManager.FindByNameAsync(model.Email);
            if (user == null)
            {
                return RedirectToAction("ResetPasswordConfirmation", "Account");
            }
            var result = await UserManager.ResetPasswordAsync(user.Id, model.Code,
model.Password);
            if (result.Succeeded)
            {
                return RedirectToAction("ResetPasswordConfirmation", "Account");
            }
            AddErrors(result);
            return View();
        }

        [AllowAnonymous]
        public ActionResult ResetPasswordConfirmation()
        {
            return View();
        }

        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public ActionResult ExternalLogin(string provider, string returnUrl)
        {
            return new ChallengeResult(provider, Url.Action("ExternalLoginCallback",
"Account", new { ReturnUrl = returnUrl }));
        }

        [HttpGet]
        public ActionResult LogOff()
        {

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);
            return RedirectToAction("Login", "Account");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
```

```
    {
        if (_userManager != null)
        {
            _userManager.Dispose();
            _userManager = null;
        }

        if (_signInManager != null)
        {
            _signInManager.Dispose();
            _signInManager = null;
        }
    }

    base.Dispose(disposing);
}

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager
{
    get
    {
        return HttpContext.GetOwinContext().Authentication;
    }
}

private void AddErrors(IdentityResult result)
{
    foreach (var error in result.Errors)
    {
        ModelState.AddModelError("", error);
    }
}

private ActionResult RedirectToLocal(string returnUrl)
{
    if (Url.IsLocalUrl(returnUrl))
    {
        return Redirect(returnUrl);
    }
    return RedirectToAction("Index", "Home");
}
```

```
        internal class ChallengeResult : HttpUnauthorizedResult
        {
            public ChallengeResult(string provider, string redirectUri)
                : this(provider, redirectUri, null)
            {
            }

            public ChallengeResult(string provider, string redirectUri, string userId)
            {
                LoginProvider = provider;
                RedirectUri = redirectUri;
                UserId = userId;
            }

            public string LoginProvider { get; set; }
            public string RedirectUri { get; set; }
            public string UserId { get; set; }

            public override void ExecuteResult(ControllerContext context)
            {
                var properties = new AuthenticationProperties { RedirectUri =
RedirectUri };
                if (UserId != null)
                {
                    properties.Dictionary[XsrfKey] = UserId;
                }

context.HttpContext.GetOwinContext().Authentication.Challenge(properties,
LoginProvider);
            }
        }
    }
}
```