НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.9:631.559:633

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Глазунова О.Г., д.п.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ 2023 р.	**__6__ ____листопада_____ 2023р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему «Аналіз і застосування алгоритмів вибору оптимального транспортного маршруту»

Спеціальність ___122_____Комп'ютерні науки
(код і назва)

Освітня програма __Комп'ютерний еколого-економічний моніторинг___
(назва)

Орієнтація освітньої програми _освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____д.т.н.,професор_____ _____ ___Семко В.В.___
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник магістерської кваліфікаційної роботи

_к.е.н., старший викладач____ _____ **_____Міловідов Ю. О.**
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ **_____Зелінський Д. В._____**
(підпис) (ПІБ студента)

КИЇВ-2023

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
___к.т.н.,доцент_____ _____ _Голуб Б.Л._____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"_29_" _____грудня_____2022 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Зелінському Данилу В'ячеславовичу_____
(прізвище, ім'я, по батькові)

Спеціальність _____122 Комп'ютерні науки
(код і назва)

Освітня програма ___Комп'ютерний еколого-економічний моніторинг
(назва)

Орієнтація освітньої програми_____освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи: Аналіз і застосування алгоритмів вибору оптимального транспортного маршруту
затверджена наказом ректора НУБіП України від "_29_"__грудня__2022р. №1917 –«С» ____

Термін подання завершеної роботи на кафедру _____2023 6 11_____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: мережеві ресурси OpenDataBot
Перелік питань, що підлягають дослідженню:
наліз предметної області.
еалізація системи.
изначення ефективності впровадження

Дата видачі завдання "29"грудня 2022 р.

Керівник магістерської кваліфікаційної роботи _____ Міловідов Ю.О.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ Зелінський Д.В.
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ВСТУП 5

1.ТЕОРІЇ ГРАФІВ ТА МЕТОДИ ЇХ РОЗВ'ЯЗАННЯ 7

1.1. Теорема оптимального шляху в графі 7

1.2 Умови завдання та основні принципи 9

1.3. Вірогідні ліміти 10

1.4 Використання оптимального шляху 10

1.5 Алгоритми пошуку маршруту 12

1.6 Алгоритм Дейкстри 13

1.7 Алгоритм Беллмана-Форда 15

1.8 Алгоритм A* 16

1.9 Алгоритм Флойда Уоршела 17

1.10 Алгоритм Лі 18

1.11 Алгоритм СН 19

Висновки за розділом 1 20

2.МУЛЬТИАГЕНТНИЙ ПІДХІД ДО РОЗВЯЗАННЯ СКЛАДНИХ ПРОБЛЕМ ТА ЗАПРОПОНОВАНА МОДЕЛЬ 22

2.1 Агентний підхід 22

2.1.1 Класифікація агентів 23

2.1.2 Координація поведінки агентів в мультиагентній системі 31

2.1.3 Приклади мультиагентних систем 32

2.2 Запропонована модель 35

Висновки за розділом 2 36

3.ОСОБЛИВОСТІ ВИКОРИСТАНИХ АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ. АРХІТЕКТУРА СИСТЕМИ 37

3.1 Contraction hierarchies 37
3.1.1 Передобробка 38
3.1.2 Witness search 40
3.1.3 Пошук маршруту в обробленому графі 41
3.1.4 Коректність 43
3.1.5 Послідовність обробки вузлів 46
3.2 Архітектура системи 48
3.2.1 Архітектура бази даних 48
3.2.2 Архітектура серверного застосунку 50
Висновки за розділом 3 56
4.РЕЗУЛЬТАТИ РОБОТИ 57
4.1 Обробка бази даних 57
4.2 Алгоритми пошуку 59
Висновки за розділом 4 81
5.РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ 83
5.1 Опис ідеї проекту (товару, послуги, технології) 83
5.2 Технологічний аудит ідеї проекту 84
5.3 Аналіз ринкових можливостей запуску стартап-проекту 84
5.4 Розроблення ринкової стратегії проекту 88
5.5 Розроблення маркетингової програми стартап-проекту 91
Висновки за розділом 5 93
ВИСНОВКИ ПО РОБОТІ 94
ПЕРЕЛІК ПОСИЛАНЬ 95

ВСТУП

Сучасні інноваційні технології автоматизації різних сфер людської діяльності відкрили унікальні можливості для прогнозування та імітації складних систем, вивчення їх характеристик і управління ними в умовах обмеженості часу, ресурсів або інформації. Формалізація даних є необхідною для математичного аналізу будь-якої системи, що дозволяє побудувати математичну платформу. Використання математичних моделей для аналізу ймовірно є одним з найбільш ефективних способів дослідження складних систем і вирішення важливих практичних завдань управління.

Графи є універсальною моделлю для представлення об'єктів та процесів у різних сферах, таких як фізика, медицина, теорія ймовірностей, виробництво обчислювальних машин, електроніка, механіка, фізіологія, радіозв'язок, авіація, картографія, графіка, антропологія та багато інших. Зазвичай, графи використовуються для вирішення задач таких як пошук найкоротшого шляху, ймовірності, топологічного планування, або кластеризації.

Один з піонерів у математичному підході до теорії графів був Леонард Ейлер, який, розглядаючи задачу про сім мостів, довів, що неможливо пройти всі сім міських шляхопроводів та повернутися до початкової точки, перетнувши кожен міст лише один раз. Через приблизно 100 років нові дослідження в галузі електромереж, кристалографії, хімії та суміжних галузей привнесли нові ідеї в теорію графів.

На сьогодні графи застосовуються не лише для отримання результату на зображенні різних процесів. Наприклад, при перегляді графа, який відображає мережу шляхів між містами держави, можна зосередитися на маршруті від пункту 1 до пункту 2.

Проте, коли маємо безліч маршрутів, хочемо вибрати оптимальний або найбезпечніший. Для розв'язання цієї проблеми потрібні розрахунки з використанням графів. В таких обставинах виявляється необхідність у використанні потужних цифрових технологій.

На сьогодні графи охоплюють різноманітні аспекти та активно розвиваються у багатьох напрямках. Особливий інтерес становить питання про знаходження найкоротшого шляху. Під час аналізу алгоритмів та методів маршрутизації важливо звернути увагу на ті, що відносяться до цієї проблеми. Математичне тлумачення графів відображається у формалізації об'єктів та зв'язків до складних випадкових множин.

1.ТЕОРІЇ ГРАФІВ ТА МЕТОДИ ЇХ РОЗВ'ЯЗАННЯ

1.1. Теорема оптимального шляху в графі

Завдання пошуку оптимального маршруту включає в себе вибір найкоротшого шляху (ланцюгу) між двома визначеними точками (вершинами) на графі, де сума ваг ребер мінімізується для прокладання маршруту. Цей тип маршруту часто відомий як геопросторовий. Важливість завдання пошуку оптимального маршруту визначається його всебічними практичними застосуваннями. Наприклад, у навігаційних системах, де шукається найкоротший маршрут між двома точками призначення на мапі.

У цьому контексті вершини відображають перехрестя, а ребра - ділянки доріг між ними. Оптимальний маршрут забезпечує мінімізацію загальної відстані між усіма пунктами, де дороги зведені до мінімуму, і таким чином визначається найкращий маршрут.

На рисунку 1.1 ілюстровано оптимальний шлях у неорієнтованому графі без вагових параметрів, а на рисунку 1.2 - оптимальний шлях у орієнтованому графі з ваговими параметрами.

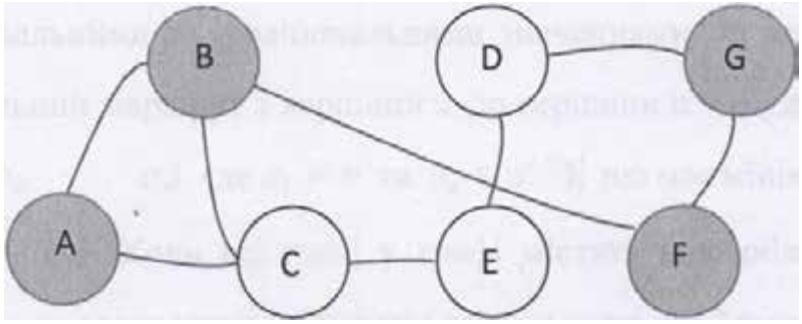

Рисунок 1.1 - Неорієнтований граф без ваг

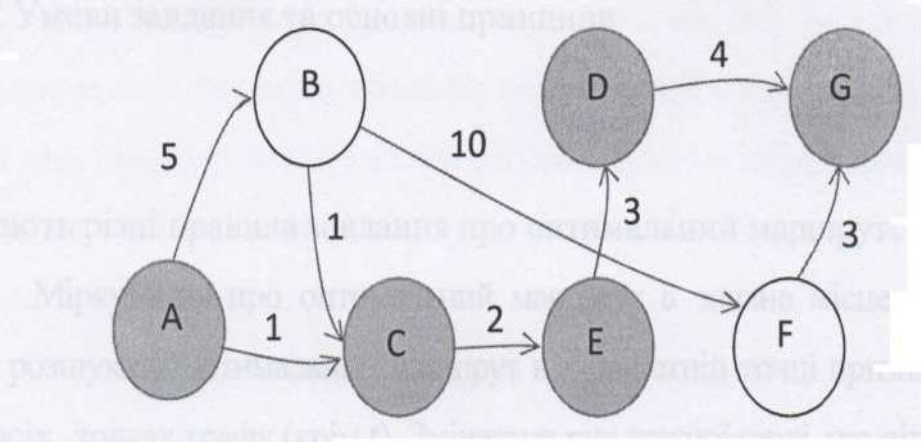

Рисунок 1.2 - Орієнтований граф з вагами

Розглянемо дилему пошуку оптимального маршруту на графі в його найпростішому представленні для неорієнтованого графа, який включає набір точок та ребер. Дві точки у графі вважаються сусідніми, якщо вони з'єднані спільним ребром[2].

Для вирішення завдання варто враховувати, що в неорієнтованому графі напрямок руху по ребрам не має значення, тому розглядаються всі можливі шляхи в обидва напрямки. У випадку орграфа (орієнтованого графа) напрямок руху вздовж ребра має визначення, що впливає на вибір оптимального маршруту. Для змішаного графа, який містить як орієнтовані, так і неорієнтовані ребра, необхідно враховувати обидва типи ребер для визначення найкоротшого шляху або оптимального маршруту.

. Шлях у неорграфі являє собою чередування вершин $P = (\vartheta_{1,}\vartheta_2, \cdots, \vartheta_\eta) \in V \times V \times \cdots \times V$, з чого зрозуміло, ЩО ϑ_i поруч з $\vartheta_i + 1$ для $1 < \text{i} < \eta$. Цей маршрут P іменують, як шлях протяжності $\mathcal{n}$ з точки $\vartheta_{1,}$ у ϑ_η (i - показує на номер точки маршруту, а також не містить аніякого ставлення до послідовності вершин у графах).

Припустимо $e_{i,j}$ — дуга, що об’єднує дві точки: $\vartheta_{1,}$ та ϑ_j .Ця зважена функція $\square$: $\text{E} \to R$, що віддзеркалює грані на їх ваги, компетентність яких випливає раціональними та ірраціональними значеннями, та неорграф G. У той час оптимальний маршрут з вершини ϑ до вершини ϑ' буде називатися шлях

$P = (\vartheta_1, \vartheta_2, \cdots, \vartheta_\eta)$ (де $\vartheta_{1,} = \vartheta$ та $\vartheta_\eta = \vartheta'$) що має мінімальну певну суму $\sum n - 1 \quad (e_{1,i+1})$. Коли всі грані у графі містять відособлену вагу, то завдання прямує до знаходження мінімальної кількості пройдених ребер [3].

1.2 Умови завдання та основні принципи

Дійсно, існує декілька варіантів постановок задачі щодо пошуку оптимального маршруту, залежно від конкретних вимог і цілей, які включають:

1. Розгляд оптимального маршруту до заданої кінцевої точки. Це означає знаходження найкоротшого маршруту від усіх інших точок графу (крім цільової точки t) до точки призначення t. Це завдання може бути переформульовано як завдання з однією початковою точкою, де оптимальний маршрут шукається з однієї вихідної точки до всіх інших.

2. Знаходження оптимального маршруту між двома заданими точками. Тут мета полягає в пошуку найкоротшого маршруту від точки u до точки v.

3. Розгляд оптимального маршруту між усіма парами точок. Це означає знаходження найкоротших маршрутів для кожної пари точок u та v. Ця задача також може бути розв'язана за допомогою методів, використаних для завдання з однією початковою точкою, але зазвичай вона вирішується швидше.

У більш загальних постановках задачі враховується не лише довжина ребра, але й інші параметри, такі як вартість, час, ресурси тощо, які враховуються при проходженні кожної точки. Ці постановки задач мають широке практичне застосування в різних галузях, таких як електроніка, економіка, картографія та інші.

1.3. Вірогідні ліміти

Завдання пошуку оптимального маршруту нерідко включає в себе додаткові обмеження, які суттєво ускладнюють задачу. Деякі з таких задач включають:

1. Оптимальний маршрут, що проходить через задану множину точок. В цьому випадку маршрут повинен пролягати через задану множину точок, а мінімальний маршрут повинен включати якомога менше невиділених вершин. Це може бути важливо в гіпотезі аналізу операцій.

2. Мінімізація оболонки вузла орієнтованого графа маршруту. В цьому випадку метою є знаходження мінімальної кількості маршрутів, що охоплюють усі вузли графа, тобто підмножина всіх s-t маршрутів, щоб усі вузли орграфа були включені хоча б у один такий маршрут.

3. Завдання про належні маршрути. Потрібно розшукати мінімальну за можливості множину $s-t$ маршрутів $P = p_{1,\cdots,p_m}$ таку, щоб для загальних $t_i \in R$ існував маршрут $p_j \in P$ що подавляє його. $R = t_{1,\cdots,t_{k-}}$ множина маршрутів у орграфі G;

4. Мінімальне зрівноваження ребер орієнтованого графа маршрутами. Ця задача передбачає пошук мінімальної кількості маршрутів, які охоплюють підмножину всіх шляхів, так щоб кожне ребро графа належало принаймні одному з цих маршрутів. Додатково враховується вимога того, щоб всі маршрути виходили з однієї точки.

1.4 Використання оптимального шляху

Питання вибору оптимального маршруту на графі може бути тлумачене по-різному та застосовуватися в різних галузях. Наступні приклади різноманітних використань цього питання будуть розглянуті. Застосування, що

не пов'язані одне з одним, досліджуються в галузі, що займається аналізом діяльності.

Методи вибору оптимального маршруту на графі використовуються для пошуку шляхів між фізичними об'єктами на картографічних додатках, таких як карти Google або Open Street Map. Можна уявити недетерміновану візуально відображену машину як граф, де вершини відображають стан, а грані визначають ймовірні переходи. Такі алгоритми пошуку найкоротшого шляху можуть бути застосовані для визначення оптимальної послідовності дій для досягнення головної мети. Наприклад, якщо вершини відображають стани Кубика Рубіка, а ребра відображають один крок, алгоритм може бути застосований для пошуку рішення з мінімальною кількістю кроків.

Завдання пошуку найкоротшого шляху на графі широко використовується для визначення найменшої відстані у мережі доріг. Мережу доріг можна представити у вигляді графа з позитивними вагами. Вершини відображають дорожні вузли, а ребра - дороги, що їх з'єднують. Ваги ребер можуть відображати довжину ділянки, час, необхідний для її подолання, або вартість подорожі по ній. Орієнтовані ребра можна використовувати для відображення односторонніх вулиць. У такому графі можна ввести характеристику, що показує, що деякі дороги важливіші за інші для довгих подорожей (наприклад, автомагістралі). Це поняття було формалізоване в ідеї про магістралі. Для впровадження підходу, де деякі дороги важливіші за інші, існує безліч алгоритмів. Вони вирішують задачу пошуку найкоротшого шляху набагато швидше, ніж аналогічні алгоритми для звичайних графів [4].

Схожі алгоритми можуть бути розбиті на два етапи:

Первинна обробка графа без врахування початкової та кінцевої вершини, яка може займати до кількох днів, особливо при роботі з реальними даними. Цей етап зазвичай виконується один раз, після чого отримані дані можуть бути використані повторно.

Здійснюється запит і пошук найкоротшого шляху, при цьому відомі початкова та кінцева вершина.

1.5 Алгоритми пошуку маршруту

До найбільш популярних алгоритмів пошуку маршруту в графі можна віднести такі:

1. Алгоритм Дейкстри шукає найкоротший шлях від однієї з вершин графа до всіх інших, працюючи лише для графів без ребер з негативною вагою.

2. Алгоритм Беллмана-Форда використовується для пошуку найкоротших шляхів в зваженому графі, де вага ребер може бути негативною.

3. Алгоритм пошуку A* знаходить маршрут з найменшою вартістю від однієї початкової вершини до іншої кінцевої, використовуючи алгоритм пошуку за першим найкращим збігом на графі.

4. Алгоритм Флойда-Уоршелла дозволяє знайти найкоротші шляхи між усіма вершинами в зваженому орієнтованому графі.

5. Алгоритм Джонсона забезпечує пошук найкоротших шляхів між усіма парами вершин у зваженому орієнтованому графі.

6. Алгоритм Лі (хвильовий алгоритм) базується на методі пошуку в ширину і використовується для знаходження найкоротшого шляху між вершинами графа, мінімізуючи кількість проміжних вершин (ребер).

7. Contraction hierarchies - це алгоритм, який використовується для пошуку коротших шляхів та "віртуального" видалення вершин, які можна пропустити при пошуку маршруту.

1.6 Алгоритм Дейкстри

Алгоритм голландського вченого Едсгера Дейкстри призначений для знаходження всіх найкоротших шляхів з однієї наперед заданої вершини графа до всіх інших. Цей метод може бути корисним у вирішенні різноманітних завдань, таких як визначення найкоротшого маршруту для подорожей між містами або оптимізація експорту товарів між країнами.

Однак, варто відзначити, що даному алгоритму притаманний певний недолік: він не пристосований для обробки графів, які мають ребра з негативною вагою. Таким чином, якщо відомо, що деяка система передбачає збиткові маршрути для підприємства, то для її обробки доцільно використати інший алгоритм [5].

Для програмної реалізації алгоритму необхідні два масиви: логічний visited, який відстежує відвідані вершини, та числовий distance, який зберігає найкоротші відстані.

Отже, є граф $G = (V, \mathrm{E})$. Кожна з вершин входять в множину V, спочатку відзначена не відвіданою, тобто елементам масиву visited присвоєно значення false.

Оскільки метою є знаходження найвигідніших шляхів, кожен елемент у векторі distance заповнюється числом, яке свідомо більше за будь-який потенційний шлях. Зазвичай це число називають "нескінченністю", але в програмній реалізації використовують, наприклад, максимальне значення конкретного типу даних. В якості початкової вершини вибирається вершина s, і їй присвоюється нульова відстань: distance[s] = 0, оскільки відсутнє ребро з s до s (метод не передбачає петель).Далі, знаходяться всі сусідні вершини (в які є ребро з s). Нехай такими будуть t і u. І по черзі досліджуються, а саме обчислюється вартість маршруту з s черзі в кожну з них.

Проте ймовірно, що до певної вершини з s існує кілька шляхів. Тому ціну шляху для цієї вершини в масиві distance необхідно переглядати, ігноруючи найбільше (неоптимальне) значення і встановлюючи найменше значення для відповідної вершини.

Після обробки суміжних з s вершин, вона відзначається як відвідана, і активною стає та вершина, до якої веде найкоротший шлях з s. Нехай шлях з s до u є коротшим, ніж шлях з s до t, тоді вершина u стає активною. Далі, вершина u відзначається як пройдена, активною стає вершина t, і весь процес повторюється для неї. Алгоритм Дейкстри продовжується до тих пір, поки всі доступні вершини з s не будуть оброблені.

На рисунку 1.3 представлена послідовність кроків пошуку найкоротших шляхів для всіх вершин від першої для певного графа.

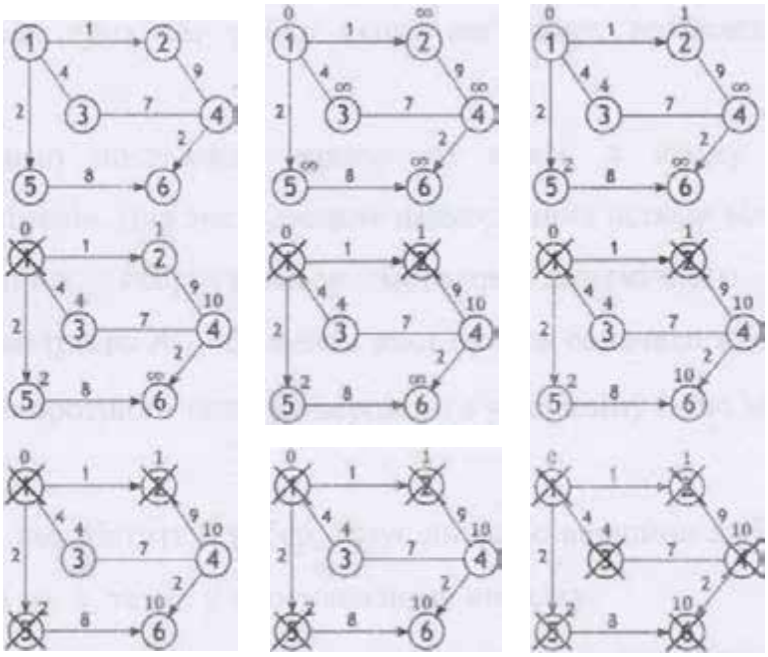

Рисунок 1.3 - Алгоритм

1.7 Алгоритм Беллмана-Форда

Алгоритм Беллмана-Форда - алгоритм пошуку найкоротшого шляху у зваженому графі. Запас $O(|V| \bullet \mathrm{I}E\mathrm{I})$ цей алгоритм призначений для знаходження найкоротших шляхів від однієї вершини графа до всіх інших. На відміну від алгоритму Дейкстри, алгоритм Беллмана-Форда дозволяє використовувати ребра з негативною вагою. Він був запропонований незалежно Річардом Беллманом і Лестером Фордом [6].

Припустимо, що у нас є орієнтований або неорієнтований граф G зі зваженими ребрами. Довжиною шляху є сума ваг усіх ребер, що входять до цього шляху. Метою є знайти найкоротші шляхи від виділеної вершини s до всіх інших вершин графа. Важливо зауважити, що в деяких випадках найкоротших шляхів може не існувати. Наприклад, у графі, який містить цикл з негативною сумарною вагою, існує нескінченно короткий шлях від однієї вершини цього циклу до іншої (кожен обхід циклу зменшує довжину шляху). Цей цикл з негативною сумою ваг ребер називається негативним циклом.

Проте, якщо в графі свідомо відсутні негативні цикли, можна застосувати алгоритм Беллмана-Форда для вирішення поставленої задачі. Для знаходження найкоротших шляхів від однієї вершини до всіх інших, використовується метод динамічного програмування.

Побудуємо матрицю A_{ij}, елементи якої будуть означати наступне: A_{ij} — це довжина найкоротшого шляху з вершини s у вершину I , що містить не більше j ребер.

Шлях, що містить 0 ребер, існує лише до вершини s. Таким чином, A_{i0} рівне 0 при $\mathrm{i} = s,$ та $+ \infty$ у протилежному випадку.

Тепер розглянемо усі шляхи з s до i, що містять рівно j ребер. Кожен такий шлях є шляхом з $j -$ 1 ребра, до якого додане останнє ребро. Якщо про довжини шляху $j -$ 1 усі дані вже підраховані, то визначити j — й стовпник матриці не складає труднощів.

1.8 Алгоритм A*

Алгоритм A* (вимовляється «А зірка» або «А стар», від англ. A star) є алгоритмом пошуку по першому найкращому збігу на графі, який знаходить маршрут з найменшою вартістю від однієї вершини (початкової) до іншої (цільової, кінцевої).

Порядок обходу вершин визначається евристичною функцією "відстань + вартість" (зазвичай позначається як f(x)). Ця функція представляє собою суму двох інших функцій: функції вартості досягнення розглянутої вершини (x) від початкової вершини (зазвичай позначається як g(x) і може бути як евристичною, так і ні) і евристичної оцінки відстані від розглянутої вершини до кінцевої (позначається як h(x)).

Функція h(x) має бути припустимою евристичною оцінкою, що означає, що вона не переоцінює відстані до цільової вершини. Наприклад, для завдання маршрутизації h(x) може представляти собою відстань до мети по прямій лінії, оскільки це фізично найкоротший можливий шлях між двома точками.

Алгоритм A* був вперше описаний в 1968 році Пітером Хартом, Нільсом Нільсоном і Бертрамом Рафаелем. Це по суті було розширення алгоритму Дейкстри, створеного в 1959 році. Він досягав більш високої продуктивності (за часом) за рахунок використання евристики. У їхній роботі він згадується як "алгоритм A". Однак, оскільки він обчислює найкращий маршрут для заданої евристики, його назвали A*.

Алгоритм A* поетапно переглядає всі шляхи, що ведуть від початкової вершини до кінцевої, поки не знайде найкоротший. Як і всі інформовані алгоритми пошуку, він спочатку розглядає ті шляхи, які "здаються" ведуть до мети. Його відрізняє те, що при виборі вершини враховується весь пройдений до неї шлях (складова g(x) - це вартість шляху від початкової вершини, а не від попередньої, як в жадібних алгоритмах).

У початковій стадії алгоритм переглядає вузли, що прилягають до початкового вузла. Обирається той, у якого значення f(x) мінімальне, після чого

цей вузол розгортається. На кожному кроці алгоритм працює з безліччю шляхів від початкової точки до всіх ще нерозгорнутих (листових) вершин графа («безліч приватних рішень»), які розміщуються в черзі з пріоритетом. Пріоритет шляху визначається значенням $f(x) = g(x) + h(x)$. Алгоритм продовжує свою роботу до тих пір, поки значення f(x) для цільової вершини не стане меншим, ніж будь-яке значення в черзі (або поки все дерево не буде проглянуто). З множини рішень обирається рішення з найменшою вартістю. Чим менше евристична функція h(x), тим більший пріоритет (тому для реалізації черги можна використовувати сортовані дерева).

1.9 Алгоритм Флойда Уоршела

Алгоритм Флойда-Уоршела є динамічним алгоритмом, який використовується для знаходження найкоротших відстаней між усіма вершинами в зваженому орієнтованому графі. Цей метод був спроектований Робертом Флойдом і Стівеном Уоршелом у 1962 році, хоча вже у 1959 році Бернард Рой розробив практично аналогічний алгоритм, але його внесок залишився непоміченим. Нехай вершини графа $G = (V, \mathrm{E}), |V| = n$ пронумеровані від 1 до n та введено позначення d_{ij}^{k} для довжини найкоротшого шляху від i до j, який окрім самих вершин i,j проходить лише через вершини $1 \ldots k$. Очевидно, що d_{ij}^{k} — довжина (вага) ребра (i,j) якщо таке існує (в протилежному випадку його довжина може бути позначена як ∞).

Існує два варіанти значення d_{ij}^{k}, k ∈ (1, ... , n)

негативною вагою, можна обчислити нову множину ребер з невід'ємними вагами, що дозволяє скористатися попереднім методом. Нова множина, що складається з ваг ребер $\widehat{\omega}$, повинна задовольняти наступні вимоги:

- для усіх ребер (u, v) нова вага $\widehat{\omega}$ (u, v) > 0;
- для усіх пар вершин u, v $\in$ V шлях P є найкоротшим шляхом з вершиниu до вершини v з використанням вагової функції ω тоді і тільки тоді, коли P — також найкоротший шлях з вершини u до вершини v з ваговою функцією $\widehat{\omega}$ [10].

1.10 Алгоритм Лі

Хвильовий алгоритм, також відомий як алгоритм Лі, є методом пошуку найкоротшого шляху на планарному графі, що базується на методах пошуку в ширину. Його застосування широко поширене в галузі комп'ютерного трасування, зокрема для планування з'єднань на друкованих платах та в різних комп'ютерних стратегічних іграх.

Алгоритм використовується для пошуку шляху в лабіринті на дискретному робочому полі (ДРП), яке представляє обмежену замкнуту лінію фігуру, розбиту на прямокутні або квадратні осередки. Робота алгоритму включає три етапи: ініціалізацію, поширення хвилі та відновлення шляху. Під час ініціалізації будується образ множини осередків оброблюваного поля, де кожному осередку приписуються атрибути прохідності чи непрохідності, а також запам'ятовуються стартовий та фінішний осередки. Алгоритм призначений для пошуку найкоротшого шляху від стартового осередку до кінцевого осередку, або в разі відсутності шляху видає повідомлення про непрохідність.

Після породження кроку до сусіднього осередку від стартового осередку перевіряється його прохідність, а також те, чи не був вже відвіданий цей осередок раніше в ході побудови шляху.

Сусідні осередки можна класифікувати у двох аспектах: за межами Мура та фон Неймана. При класифікації за межами Мура сусідніми осередками вважаються всі 8 осередків, включаючи діагональні, тоді як при класифікації за межами фон Неймана враховуються лише 4 осередки по вертикалі та горизонталі.

Якщо в осередку виконуються умови прохідності та він не належить до осередків, що вже були відвідані під час побудови шляху, в атрибут цього осередку записується число, що відображає кількість кроків від стартового осередку. На першому кроці від стартового осередку це буде 1. Кожен осередок, що має позначку кількості кроків від стартового осередку, стає новим стартовим, з якого генеруються чергові кроки до сусідніх осередків. Таким чином, ітеративний процес перебору дозволяє знайти шлях від початкового осередку до кінцевого, або виявити неможливість здійснення наступного кроку з будь-якого породженого в ході побудови шляху осередку. У зворотному напрямку від фінішного осередку до стартового відбувається процес відновлення найкоротшого шляху. [13].

1.11 Алгоритм CH

Алгоритм з переобробки графа, відомий як алгоритм скорочення (або алгоритм згортання), є ефективним інструментом для оптимізації графа з метою прискорення пошуку маршрутів. Після застосування цього алгоритму отримується спрощений граф, який може бути подано в іншій формі для подальшого пошуку маршрутів.

Принцип дії цього алгоритму полягає у виборі однієї вершини на кожному кроці, після чого для всіх сусідніх вершин додаються ребра скорочення, якщо це можливо. Основна ідея полягає в тому, щоб зменшити загальну складність графа, ігноруючи деякі проміжні вершини, через які проходять найкоротші маршрути.

На прикладі рисунка 1.4 можна побачити, як відбувається додавання ребра скорочення між вершинами A та E, оскільки найкоротший маршрут між цими вершинами проходить через вершину C. Аналогічно додається ребро скорочення між вершинами A та B. Це призводить до спрощення графа і оптимізації майбутнього пошуку маршрутів, ігноруючи вершину C при подальших обчисленнях.

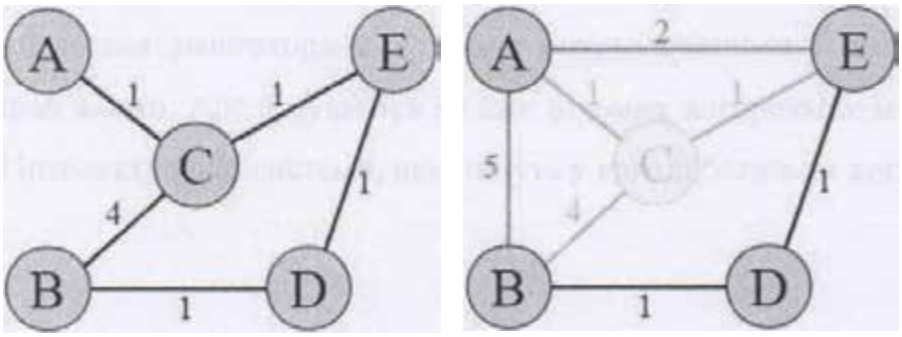

Рисунок 1.4- Додавання ребер

Так, вибір порядку обробки вершин дійсно може впливати на тривалість роботи алгоритму. Хоча результат не зміниться, ефективність обробки буде варіюватися в залежності від способу вибору порядку обробки.

Передобробка, яка передує застосуванню алгоритму скорочення, може займати певний час. Але великий виграш у швидкості пошуку маршрутів, який вона забезпечує для графа реальних доріг, що може включати мільйони вершин і ребер, виправдовує це.

Важливо враховувати цей аспект під час розробки алгоритмів, оскільки оптимізація шляху обробки може суттєво вплинути на продуктивність системи в цілому.

Висновки за розділом 1

Так, задачі теорії графів, пов'язані з пошуком шляху, є фундаментальними і широко застосовуються в різних галузях. Алгоритми, які були описані, такі як алгоритм Дейкстри, алгоритм Беллмана-Форда, алгоритм A*, алгоритм Флойда-

Уоршела, алгоритм Джонсона і алгоритм Лі, є ключовими для розробки ефективних навігаційних систем та інших інтелектуальних систем, що вимагають оптимальних шляхів.

Незважаючи на те, що сучасні алгоритми пошуку шляхів у графах є добре оптимізованими, можливість використання вже відомих алгоритмів як основи для розробки нових інтелектуальних систем з вдосконаленими функціями дозволяє досягати нових вершин в цій області. Інновації у цій сфері можуть покращити якість навігації та інших важливих систем для користувачів.

2.МУЛЬТИАГЕНТНИЙ ПІДХІД ДО РОЗВЯЗАННЯ СКЛАДНИХ ПРОБЛЕМ ТА ЗАПРОПОНОВАНА МОДЕЛЬ

2.1 Агентний підхід

Інтелектуальні мультиагентні системи (МАС) представляють собою перспективний напрямок штучного інтелекту, що базується на результатах досліджень у галузі розподілених комп'ютерних систем, мережевих технологій і паралельних обчислень. Вони забезпечують автономність окремих частин програми, які взаємодіють у розподіленій системі, виконуючи безліч взаємопов'язаних процесів.

Такі системи знайшли широке застосування в багатьох сферах, включаючи управління інформаційними потоками і мережами, управління повітряним рухом, пошук інформації в мережі Інтернет, електронну комерцію, навчання, колективне прийняття багатокритеріальних управлінських рішень та інші.

Агент у МАС є автономним штучним об'єктом, який зазвичай представлений комп'ютерною програмою. Він володіє активною мотивованою поведінкою і може взаємодіяти з іншими об'єктами у динамічних віртуальних середовищах. Кожен агент може приймати повідомлення, інтерпретувати їх зміст і формувати нові повідомлення, які передаються в загальну базу або направляються іншим агентам.

Мультиагентні системи є потужним інструментом для моделювання складних систем, таких як системи управління транспортним рухом, системи управління мережами, ринкові системи та інші. Вони дозволяють ефективно вирішувати завдання, які вимагають координації і взаємодії різних агентів для досягнення спільної мети.

Застосування інтелектуальних мультиагентних систем в різних галузях дозволяє ефективно вирішувати складні завдання, які вимагають координації

багатьох агентів, забезпечуючи оптимальні результати та ефективну роботу системи в цілому.

Інтелектуальні агенти в МАС володіють рядом важливих властивостей, які забезпечують їх ефективну роботу в різноманітних умовах. Давайте розглянемо ці властивості докладніше:

- Автономність: ця властивість дозволяє агентам функціонувати без постійного контролю та втручання з боку зовнішнього нагляду.
- Активність: агенти можуть організовувати та здійснювати дії, які спрямовані на досягнення своїх цілей.
- Товариськість: здатність до взаємодії та спілкування з іншими агентами, що дозволяє їм працювати разом для досягнення спільної мети.
- Реактивність: здатність сприймати зміни в середовищі та відповідати на них відповідним чином.
- Цілеспрямованість: наявність власних мотивацій та цілей, які спонукають агента до певних дій та вирішення завдань.
- Базові знання: розуміння агентом себе, інших агентів та навколишнього середовища, що є основою для прийняття рішень та взаємодії з оточенням.
- Переконання: змінні частини базових знань, які можуть змінюватися з часом, відображаючи зміни в стані середовища або у свідомості агента.
- Бажання: прагнення агента до певних станів або результатів, що мотивує його дії та вирішення завдань.
- Зобов'язання: завдання, які агент приймає на себе, відповідаючи на запити або доручення інших агентів або системи в цілому.

2.1.1 Класифікація агентів

Так, класифікація агентних програм може враховувати ступінь розвитку уявлень про навколишній світ та спосіб прийняття рішень. Найпростіші агенти, такі як прості рефлексивні агенти, діють на основі поточного сприйняття стану

середовища і не враховують історію сприйняття. Вони мають обмежений рівень інтелектуальності, оскільки їхні дії базуються виключно на поточних станах оточення.

У випадку агентів з урахуванням внутрішнього стану, вони відстежують зміни в середовищі та володіють багатогранними внутрішніми станами, які залежать від історії сприйняття. Вони поєднують поточне сприйняття з колишнім внутрішнім станом для прийняття рішення та здійснення дій, що сприяють зміні їхнього внутрішнього стану. Це дозволяє їм адаптуватися до змін у середовищі та розвивати більш складну поведінку, ніж у простих рефлексивних агентів.

Класифікація агентних програм за цими ознаками допомагає зрозуміти різні рівні інтелектуальності та способи взаємодії агентів з оточенням.

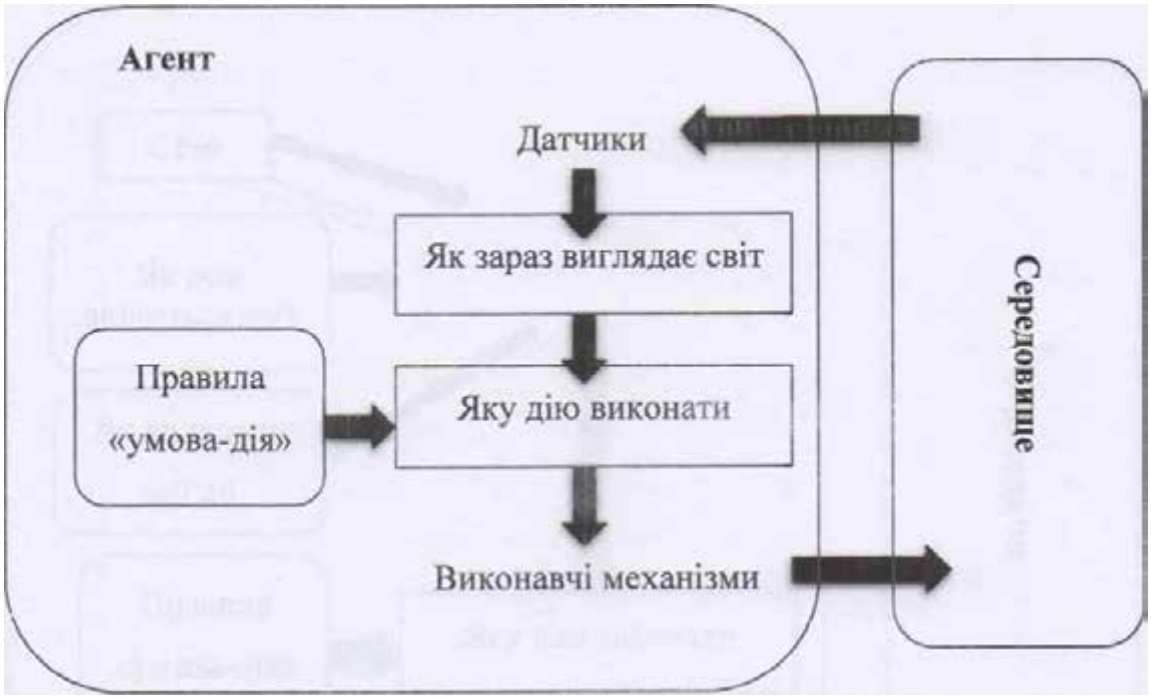

Рисунок 2.1 - Структура простого рефлексивного агенту

Знань про поточний стан середовища не завжди достатньо для прийняття рішення. Тоді агенту потрібно не тільки опис поточного стану, а й інформація про цілі, яка описує бажані ситуації. Структура агенту, що діє на основі цілей

зображена на рис. 2.2. Він стежить за станом середовища, а також за кількістю цілей, яких він намагається досягти, і вибирає дію, спрямовану на досягнення цих цілей.

Часто бувають ситуації, коли для прийняття рішення недостатньо інформації тільки про цілі. По-перше, якщо є конфліктні цілі, такі, що можуть бути досягнуті тільки деякі з них (наприклад, швидкість, або безпека). По- друге, якщо є кілька цілей, до яких може прагнути агент, але кожна з них може бути досягнута з деякою ймовірністю успіху. В цьому випадку в програм агента вводиться функція корисності яка ставить у відповідність станам агента дійсне число, що означає корисність даного стану. Агент вибирає дію, яка веде до найкращої очікуваної корисності.

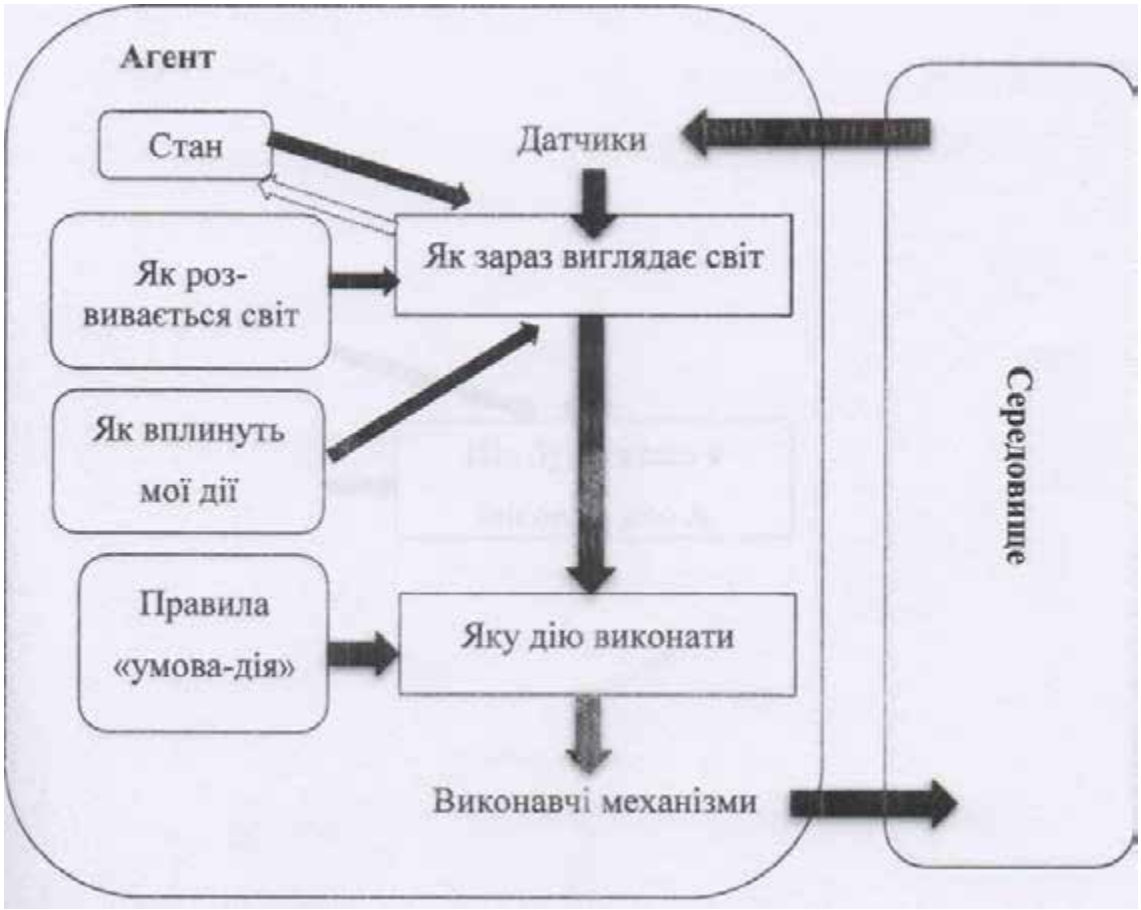

Рисунок 2.2 – Агент, що діє з врахуванням внутрішнього стану

В особливий клас виділяють агентів, що навчаються. Навчання має важливу перевагу: воно дозволяє агенту функціонувати в спочатку невідомих йому варіантах середовища і ставати більш компетентним у порівнянні з тим, що могли б дозволити тільки його початкові знання.

Структура агента, що навчається, включає чотири концептуальні компоненти. Продуктивний компонент визначає, як агент має взаємодіяти з середовищем, вибираючи дії на основі поточного стану. Навчальний компонент використовує інформацію зворотного зв'язку від критика, яка надходить від оцінки дій агента, для модифікації продуктивного компонента з метою покращення його дій у майбутньому.

Навчання дозволяє агентам збільшувати свою ефективність та адаптуватися до змін в середовищі, забезпечуючи їм здатність працювати в широкому спектрі ситуацій та досягати кращих результатів у вирішенні поставлених завдань.

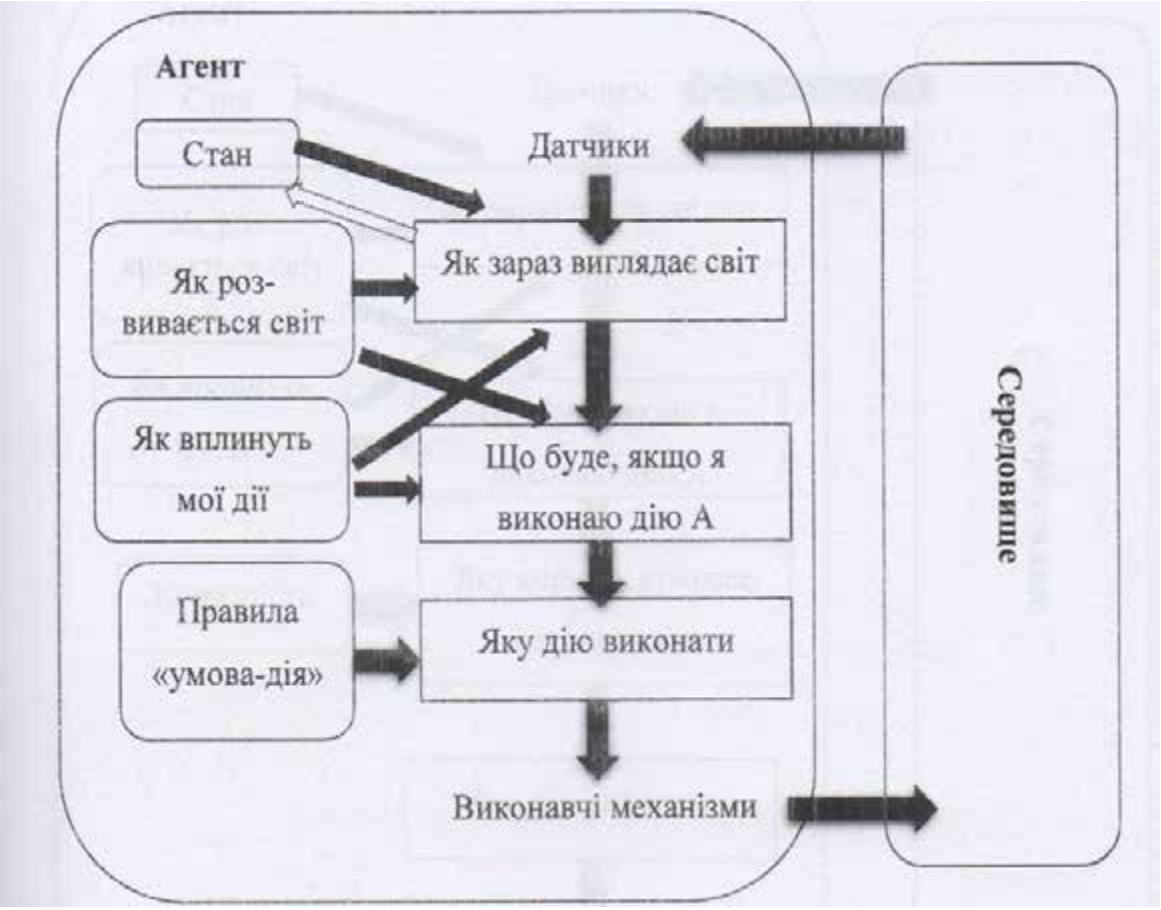

Рисунок 2.3 - Агент, що діє на основі цілей

Критик повідомляє навчальному компоненту, наскільки добре діє агент з урахуванням постійного стандарту продуктивності, оскільки самі результати сприйняття не дають ніяких вказівок на те, чи успішно діє агент. Цей стандарт виступає як зовнішній по відношенню до агента, оскільки агент не повинен мати можливості модифікувати його. Наприклад, в шаховій грі програма може отримати дані сприйняття, що вказують на те, що вона ставить мат своєму противнику, але для визначення того, що це вважається хорошим результатом, необхідно мати стандарт продуктивності.

Критик надає повернену інформацію навчальному компоненту, який використовує ці дані для покращення стратегії агента. Навчання базується на використанні зворотного зв'язку для вироблення кращих дій агента в

майбутньому, забезпечуючи йому можливість удосконалення його рішень та досягнення кращих результатів у майбутніх завданнях.

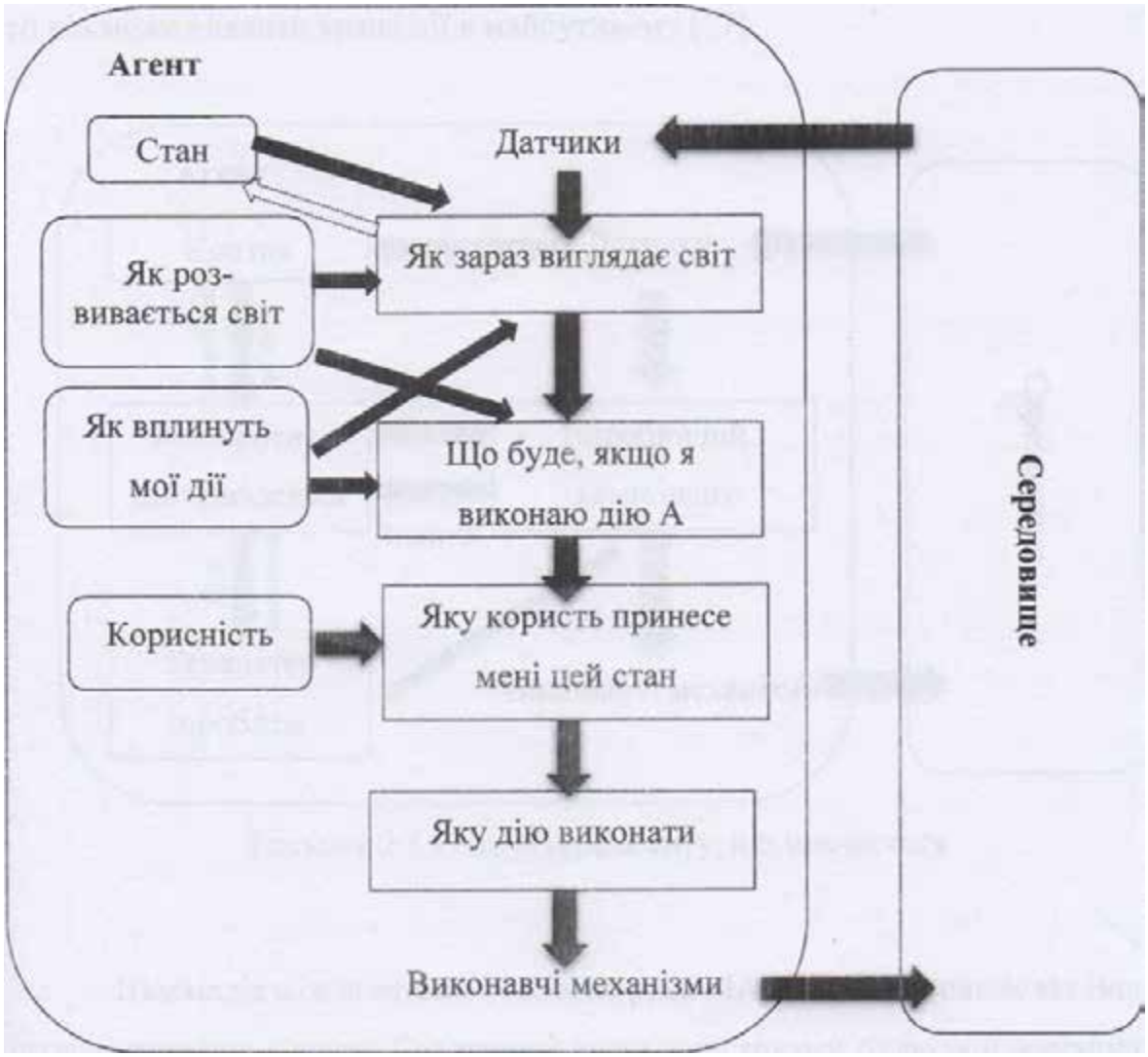

Рисунок 2.4 - Структура агенту заснованого на моделі і корисності

Генератор проблем в агенті, що навчається, відіграє важливу роль у забезпеченні розвитку агента шляхом пропозиції нових дій, які можуть допомогти збагатити його досвід. Це відкриває можливість для експериментації та вивчення нових стратегій, які можуть бути корисними у майбутньому. Агент, який готовий до випробувань і спроможний пристосовуватися до нових умов, може здобувати перевагу у довгостроковій перспективі, допомагаючи йому уникати застою та покращувати свої результати[17].

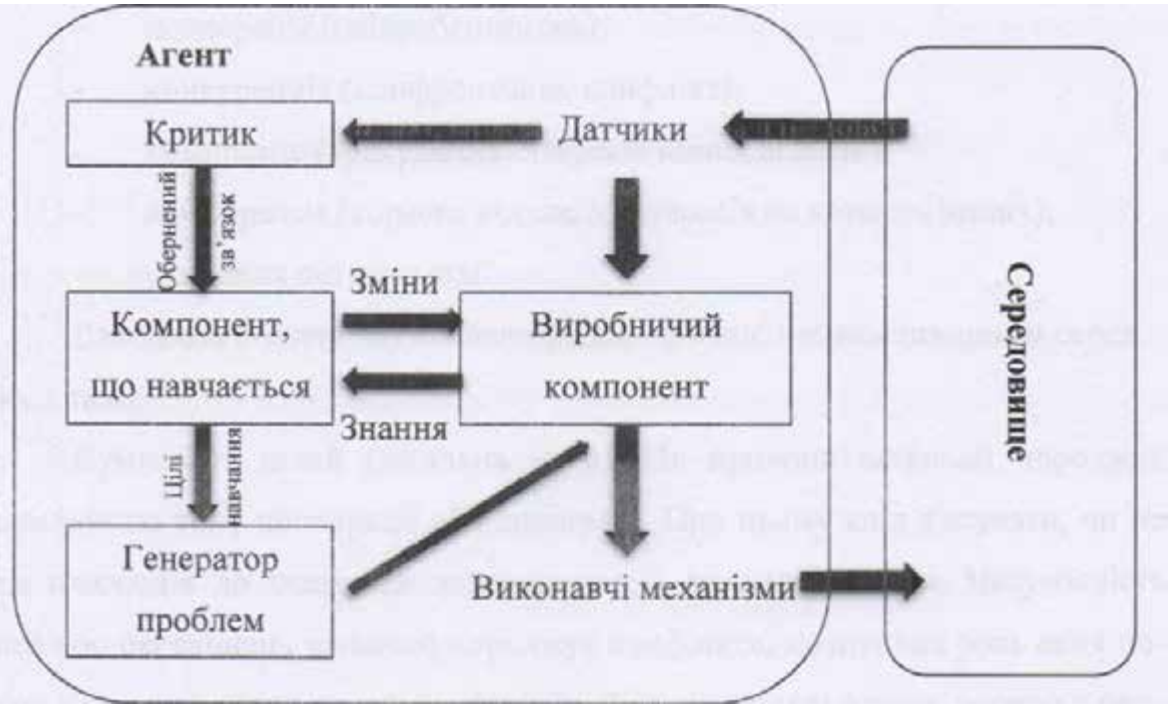

Рисунок 2.5 - Структура агенту, що навчається

Взаємодія між агентами в мультиагентних системах (МАС) є ключовою рисою, що відрізняє їх від інших інтелектуальних систем. В контексті МАС, розглядаючи спрямованість, вибірковість, інтенсивність і динамічність взаємодії, можна визначити різні форми взаємодії, такі як кооперація, конкуренція, компроміс, конформізм та ухилення від взаємодії.

Низка причин може викликати взаємодію між агентами. Наприклад, спільна мета агентів може стимулювати співробітництво або конфронтацію. У таких випадках важливо враховувати можливі наслідки для життєздатності окремих агентів. Несумісність цілей або переконань, навпаки, може призвести до конфліктів, які в свою чергу можуть спонукати до розвитку та розширення уявлень.

Додатково, взаємодія може бути обумовлена загальними ресурсами, які використовуються агентами для досягнення їх цілей. Обмеженість таких ресурсів часто породжує конфлікти, що можуть бути вирішені шляхом переговорів та пошуку компромісів, що враховують інтереси всіх сторін.

Розподіл ринкових часток, витрат та прибутків у спільних підприємствах є прикладами взаємодії, яка обумовлена загальними ресурсами.

Ці механізми взаємодії допомагають агентам співпрацювати або конкурувати залежно від ситуації, що виникає у конкретних умовах їхньої діяльності[17].

Взаємодія між агентами в MAC є складним процесом, що може приймати різні форми в залежності від ситуації та мети. Агенти можуть співпрацювати для вирішення спільних завдань, але при цьому вони також можуть мати власні індивідуальні цілі.

Зобов'язання є важливим інструментом, який дозволяє упорядковувати взаємодію агентів, передбачати їхню поведінку і планувати власні дії. Вони можуть бути спрямовані на інших агентів, на групу агентів, або навіть на самого себе. Формальне представлення цілей, зобов'язань, бажань і намірів є ключовим елементом ментальної моделі інтелектуального агента, яка керує його мотивованою поведінкою в автономному режимі.

Залучення відсутнього досвіду, обмін знаннями та координація дій агентів є важливими елементами взаємодії між ними. Водночас, непродуктивна співпраця може виникнути, коли агенти не ефективно обмінюються досвідом, або коли їхні дії взаємно заважають один одному.

Досягнення балансу між різними формами взаємодії та ефективним використанням знань і ресурсів кожного агента є важливою задачею у розвитку інтелектуальних мультиагентних систем[17].

2.1.2 Координація поведінки агентів в мультиагентній системі

Моделювання колективної роботи агентів дійсно вимагає уваги до ряду проблем, які виникають у процесі їх взаємодії. Розпізнавання необхідності кооперації, вибір партнерів, врахування їхніх інтересів, організація переговорів, декомпозиція завдань, розподіл обов'язків та інші аспекти мають вирішальне значення для успішної координації дій агентів.

У моделях координації поведінки агентів використовуються різні ідеї, включаючи відмову від строго пошуку оптимального рішення на користь прийнятного компромісу, застосування самоорганізації для формування колективної поведінки, рандомізацію для вирішення конфліктів та рефлексивне управління.

Теоретико-ігрові моделі, моделі колективної поведінки автоматів, моделі планування колективної поведінки, моделі на основі BDI-архітектури (Belief - Desire - Intention) та моделі координації поведінки на основі конкуренції є найбільш відомими підходами для вирішення цих проблем у контексті колективної роботи агентів.

Теорія ігор є важливим інструментом для моделювання вибору рішень у ситуаціях конфлікту та невизначеності. Поняття рівноваги гри дозволяє визначити оптимальні рішення для учасників гри, що стало корисним у теорії багатоагентних систем (МАС). Механізм пошуку положень рівноваги може служити засобом самоорганізації колективної поведінки агентів.

Моделі колективної поведінки автоматів, які ґрунтуються на ідеях рандомізації, самоорганізації та повної розподіленості, добре підходять для створення протоколів переговорів у задачах з великою кількістю простих взаємодій з невідомими характеристиками.

Моделі планування колективної поведінки можуть бути централізованими, частково централізованими або розподіленими. У випадку розподіленого планування агенти самостійно приймають рішення про вибір своїх дій у процесі

координації часткових планів. Це вимагає розгляду раціональної децентралізації, зміни цілей при конфліктах та вирішення обчислювальної складності.

Моделі на основі BDI-архітектури зосереджуються на описі понять, таких як переконання (belief), бажання (desire) і наміри (intention). Вони використовують аксіоматичні методи теорії ігор і логічної парадигми штучного інтелекту, щоб узгодити результати логічного висновку з баз знань окремих агентів. Однак це може призводити до складнощів в обчисленнях та проблем, пов'язаних з вибором між виконанням власних завдань та зобов'язань по відношенню до партнерів.

Моделі на основі конкуренції використовують поняття аукціону як механізму координації поведінки агентів. Аукціони можуть бути відкриті або закриті, а також англійські або голландські. Учасники можуть приймати рішення на основі різноманітних моделей міркувань, які використовують різні типи знань і способи їх обробки. Повинні бути закладені кошти для вирішення можливих конфліктів, а рандомізація може використовуватися як спосіб вирішення конфліктів.

Ці моделі сприяють узгодженню дій між агентами, використовуючи різні підходи, що базуються на теорії ігор, логіці, конкуренції та аукціонах, залежно від контексту та вимог системи.

2.1.3 Приклади мультиагентних систем

Розглянемо практичні приклади організації взаємодії в мультиагентних системах з використанням різних механізмів узгодження поведінки.

Електронний магазин. Розглянемо тип завдання електронної комерції за участю агентів-продавців і агентів-покупців. Торгівля працює в електронному магазині, який являє собою програму, розміщену на сервері.

Його основна мета — організація взаємодії агентів, інтереси яких збігаються. Агенти діють від імені своїх особистих користувачів. При цьому агенти з продажу прагнуть продати свій товар за якомога вищою ціною, а агенти

з купівлі хочуть купити товар за якомога нижчою ціною. Обидва типи агентів діють автономно і не мають повної співпраці. Електронний магазин реєструє появу та зникнення агентів і організовує контакти між ними, роблячи їх «видимими» один для одного.

Поведінка торгового агента характеризується наступними параметрами:

- бажана дата, до якої товар повинен бути проданий;
- бажана ціна, за якою користувач бажає продати товар;
- найнижча допустима ціна, нижче якої товар не реалізується;
- функція зниження ціни в часі (лінійна, квадратична тощо);
- опис товару, що продається.

Агент-покупець має «симетричні» параметри:

- остання дата придбання товару;
- бажана ціна закупівлі;
- найвища прийнятна ціна;
- функція зростання ціни в часі;
- опис товару, що купується.

Торги проводяться за схемою закритого аукціону першої ціни. Поведінка агентів описує просту модель, у якій знання та вимірювання не відбуваються. Торговий агент, отримавши від електронного магазину інформацію про накопичених покупців його товару, опитує їх усіх і приймає рішення про можливість укладання угоди. Договір укладається з першим агентом-покупцем, який готовий надати запитувану ціну за товар. Продавець не може вступати в повторний контакт з будь-яким покупцем, поки не опитає всіх найбільших покупців. З кожним контактом агент з продажу веде переговори, пропонуючи початкову ціну або знижуючи її. Подібним чином діє і агент покупця, який шукає продавців необхідного товару і пропонує їм свою ціну покупки, яку він може знизити в процесі переговорів. Будь-яка угода вважається укладеною тільки в разі її схвалення користувачем агента. Дана схема переговорів являє собою найпростіший випадок взаємодії автономних агентів, що діють реактивно. Проте підсумкова поведінка системи цілком адекватна реальності.

Створення віртуальних підприємств - це актуальний напрям у сучасному бізнесі, що сприяє збільшенню ефективності завдяки поєднанню ресурсів та швидшому введенню продукції на ринок. Віртуальне підприємство є спільною організацією незалежних юридичних осіб, які спільно виробляють продукцію або послуги в рамках спільного бізнес-процесу.

Для ефективного управління віртуальним підприємством необхідно розподілити завдання між реальними підприємствами-партнерами, які виконують окремі частини загального технологічного ланцюжка. Це вимагає використання методів системного аналізу та мультиагентних технологій. Для ефективного розподілу робіт між підпроцесами та учасниками використовуються методи призначення, а також механізми аукціону.

На рисунку 2.6 зображено схему аукціону для створення віртуального підприємства, де виділені бізнес-процеси A, B, C, D, E, і чотири підприємства P1, P2, P5, P4, що конкурують за їх виконання. Кожне з підприємств представлено інтелектуальним агентом, з одним з них (Px) виступає в ролі ініціатора аукціону.

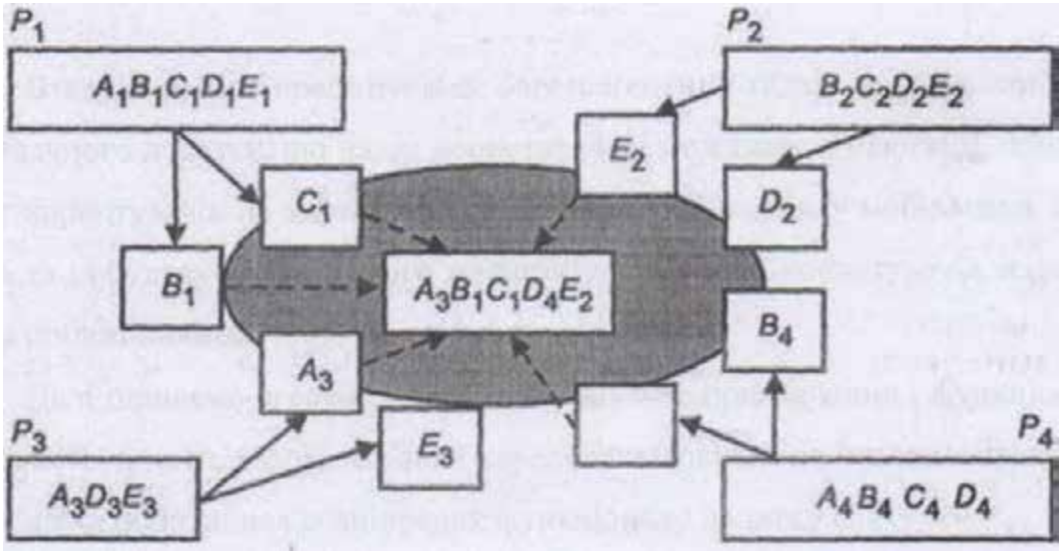

Рисунок 2.6 - Схема створення віртуального підприємства

Перед початком аукціону менеджер або аукціонер вирішує встановити базу даних та базу знань про учасників аукціону. Він подає на продаж окремі бізнес-процеси, описуючи їх початкову ціну та вимоги, що відповідають заданим критеріям. Кожен учасник аукціону може висунути свої пропозиції, вказавши параметри, які він може забезпечити, а також свою ціну.

Аукціонер оцінює та обробляє всі пропозиції, використовуючи модель розуміння, яка враховує інформацію, що він має про учасників. Після цього він приймає рішення щодо вибору призначень або відкидає їх, і, при необхідності, висуває нові пропозиції.

Завдання створення віртуального підприємства є складним завданням, яке можна віднести до категорії структурного синтезу складних систем, спрямованих на вирішення певних задач згідно з вимогами, що ставляться[16].

2.2 Запропонована модель

У цьому проекті пропонується багатоагентний підхід до створення багатозадачного додатку, який надає користувачеві можливість навігації та пошуку інших користувачів на карті за допомогою сигналу GPS мобільного пристрою, а також можливість побудови колективного маршруту для групи користувачів згідно з їхніми побажаннями.

Давайте розглянемо опис агентів, які були реалізовані, їх функції та додаткові агенти, які передбачається реалізувати для розширення функціоналу додатку.

Агент обробки даних OSM: Цей агент використовує картографічну інформацію, що надається сервісом OpenStreetMap. Його завдання полягає в завантаженні необхідної частини карти на сервер і парсингу цих даних в базу даних, щоб забезпечити зручну обробку даних під час пошуку маршрутів.

Агент пошуку маршруту: Цей агент взаємодіє з базою даних, яку створив попередній агент, і вибирає алгоритм, який найкраще підходить для конкретної задачі пошуку шляху.

Агент передобробки графу: Цей агент реалізує алгоритм Contraction Hierarchies (CH), який оптимізує граф доріг шляхом зменшення кількості ребер, що дозволяє прискорити пошук маршрутів.

Додаткові агенти: Для розширення функціоналу можуть бути реалізовані інші агенти, наприклад, агент моніторингу трафіку на дорогах, який надаватиме інформацію про затори та допоможе внести зміни в знайдений маршрут.

Висновки за розділом 2

В цьому розділі було розглянуто загальну концепцію агентного підходу до розробки програмних продуктів. Була розглянута класифікація агентів і наведені приклади багатоагентних систем. В результаті була запропонована модель багатоагентної системи для створення та обробки графу доріг, роутингу, в тому числі колективного роутингу - побудови маршрутів для групи людей. Зазначено, що ця модель передбачає можливість збільшення кількості агентів для покращення продуктивності та ефективності системи.

3.ОСОБЛИВОСТІ ВИКОРИСТАНИХ АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ. АРХІТЕКТУРА СИСТЕМИ

Розділ присвячений детальній розробці та реалізації алгоритмів пошуку маршрутів у конкретній багатоагентній системі маршрутизації. Особлива увага приділяється двостороннім модифікаціям алгоритмів Дейкстри та A*. Розглянуто етапи алгоритму Contraction hierarchies, а також висвітлено особливості побудови архітектури додатка для його реалізації.

Зазначено, що двосторонні алгоритми відрізняються від односторонніх тим, що запускаються дві процедури пошуку з початкової і кінцевої точок. Кроки обох процедур виконуються почергово, і алгоритми Дейкстри та A* зупиняються тоді, коли в закриті списки обох процедур потрапляє одна і та ж вершина. На завершальному етапі шлях прямої процедури об’єднується з оберненим шляхом зворотної процедури[17].

3.1 Contraction hierarchies

Алгоритм Contraction hierarchies (CH) базується на поступовому вилученні вершин з вихідного графу в заданому порядку. При видаленні вершини найкоротший шлях між парами вершин може бути втрачений, якщо він пролягає через видалену вершину. Для збереження відстаней у відповідності до вихідного графу, алгоритм додає нові "скорочені" ребра, які з'єднують пари вершин. Це дозволяє зберегти відстань між будь-якими двома вершинами такою ж, як в початковому графі. Таким чином, граф, що утворюється, містить той самий набір вершин, що й вихідний граф, усі його ребра, а також додаткові шляхи-скорочення.

Алгоритм дозволяє оптимізувати пошук шляхів у великих графах шляхом передпідготовки графа і зменшення його розміру за рахунок додавання скорочених ребер. Одним із основних завдань CH є забезпечення швидкості пошуку найкоротших шляхів у графі з великою кількістю вершин та ребер. [24].

Розглянемо алгоритм детальніше.

3.1.1 Передобробка

Розглянемо, що відбувається при обробці вершини. На рисунку 3.1 зображено приклад елементарного графа рис 3.1.

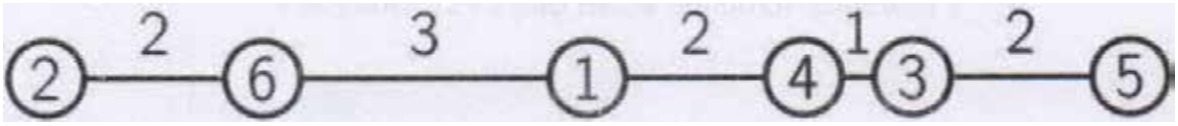

Рисунок 3.1 - Приклад графу

Числа у вершинах - це порядок, у якому вони будуть оброблятися. Вибір порядку обробки буде розглянуто пізніше. Цифри над ребрами означають довжину ребра.

При обробці вершини номер 1 ми вважаємо, що вершина зникає з графа, а разом з нею і всі ребра, що входять і виходять з нього. Тобто ми втрачаємо зв'язок між вершинами 6 і 4, який є єдиним, а отже найкоротшим. Ми повинні відновити цей зв'язок, додавши між ними ребро скорочення довжиною 5. Результат обробки вершини 1 показано на рисунку 3.2.

Під час обробки вершини 2 жодні з’єднання не будуть розірвані, а отже, не потрібно додавати ребра.

Розглянемо обробку вершини 3. Як і у випадку з вершиною 1, при обробці третьої вершини буде видалено єдиний зв'язок між вершинами 4 і 5. Його теж треба відновити. Ми бачимо результат на малюнку 3.3

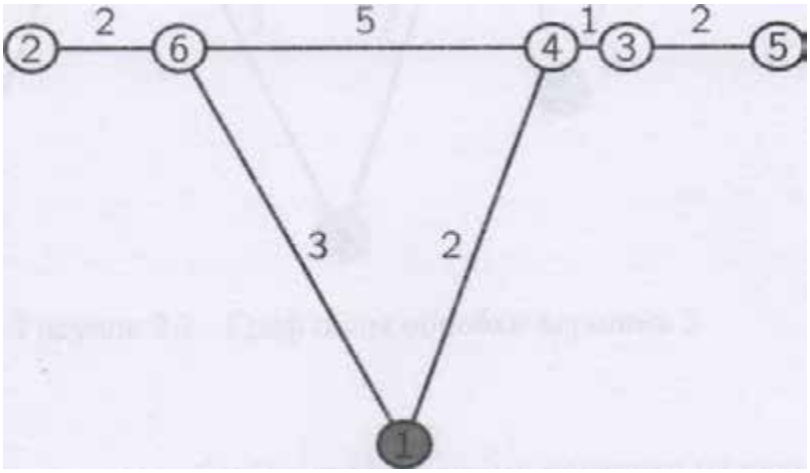

Рисунок 3.2 - Граф після обробки вершини 1

Відмітимо, що при обробці нас цікавлять лише шляхи, що з'єднують вершини, які ще не оброблялися. Так, при обробці вершини номер 4 ми не звертаємо уваги на те, що між 1 та 3 зв'язок зникне. Розглядаємо ми лише вершин 6 та 5 і додаємо між ними ребро скорочення довжини 5+3=8.

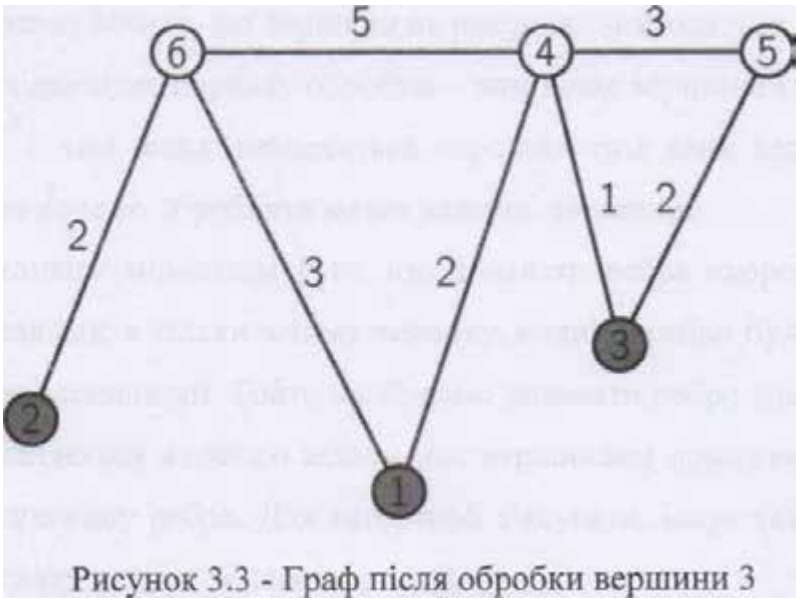

Рисунок 3.3 - Граф після обробки вершини 3

Повний результат обробки графу можемо побачити на рисунку 3.4.

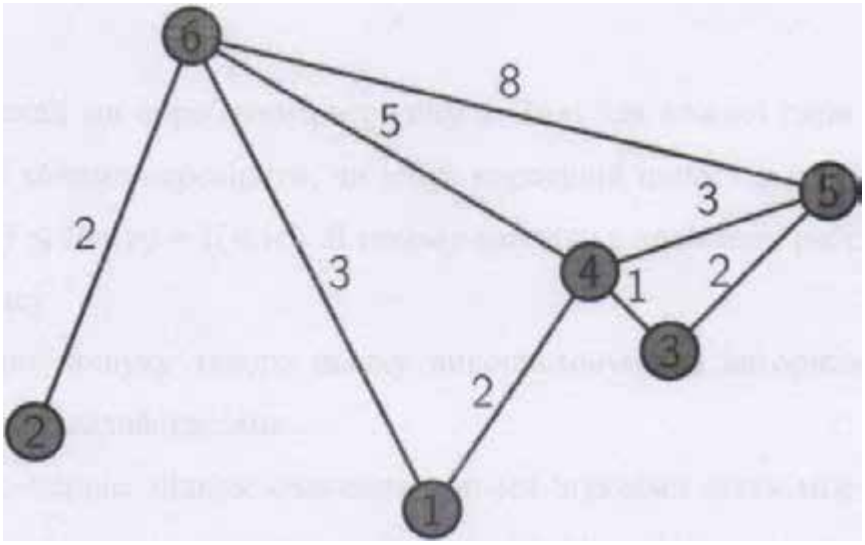

Рисунок 3.4 - Граф після повної обробки

Ми бачимо, що після обробки вершини 4 між 6 і 5 з'являється тільки одне ребро довжиною 8, але насправді це ребро приховує ланцюжок вершин 6 - 1 -4-3-5.

Ми бачимо, що піки на малюнках знаходяться на різній висоті. Ця висота відповідає порядку обробки - чим вищий пік, тим пізніше він був оброблений. І чим вищий пік, тим він важливіший. Тобто ми хочемо спочатку обробити менш важливі вершини.

Важливим моментом є те, що не завжди потрібно додавати ребра скорочення, а тільки в тому випадку, коли це ребро є найкоротшим шляхом між вершинами. Тобто ми додамо ребро лише тоді, коли в графі немає шляху між вершинами з довжиною, меншою або рівною довжині ребра. Щоб з'ясувати, чи є такий спосіб, виконується процедура пошуку свідка [19].

3.1.2 Witness search

Нехай ми обробляємо вершину v. Тоді для кожної пари ребер (u, v) і (y, w) ми хочемо перевірити, чи існує коротший шлях від u до w, такий, що $l(u, \dots, w) \leq l$ (u, v) +l(v, w). В такому випадку в додаванні ребра скорочення немає сенсу.

При пошуку такого шляху використовується алгоритм Дейкстри з невеликими змінами.

По-перше, цілком очевидно, що ми шукаємо шлях між y і W, який не буде проходити через вершину v. Тому на всіх ітераціях алгоритму ми будемо пропускати цю вершину, ніби її не існує.

По-друге, запуск цієї процедури буде дуже частою операцією, тому важлива її оптимізація.

Нас цікавить, чи існує шлях довжиною, меншою за певне відоме значення. Коли ми будемо додавати у відкритий список тільки ті вершини, відстань до яких буде менше довжини ребра скорочення, для якого розпочато пошук. Зрештою, якщо шуканого шляху не існує, то у відкритому списку не залишиться жодного елемента, і ми ніколи не досягнемо краю w. Тоді можна з упевненістю вважати, що шлях скорочення є найкоротшим у графі між розглянутими вершинами і його необхідно додати.

Іншим варіантом оптимізації є обмеження кількості країв у шляху пошуку. Наприклад, ми можемо встановити, що якщо число ребер досягло п'яти, а ми не досягли вершини w, незважаючи на те, що у відкритому списку все ще є елементи, ми зупинимо процедуру і припустимо, що коротшого немає шлях.

Варто також зазначити, що хоча ця процедура буде виконуватися досить часто, пошук буде проводитися між досить близькими вершинами, і тому він триватиме недовго [20].

3.1.3 Пошук маршруту в обробленому графі

При пошуку маршруту в обробленому графі необхідно використовувати двосторонній алгоритм пошуку. У нашому випадку це буде двосторонній алгоритм Дейкстри.

Як уже було сказано, під час обробки ми зберігаємо порядок обробки вершин графа. Таким чином ми отримуємо ієрархії вершин відповідно до їх

важливості. При пошуку маршруту ми будемо йти тільки «вгору», тобто від вершини з меншим порядковим номером до вершини з більш високим.

Найкоротший знайдений шлях спочатку підніметься на якусь вершину, а потім почне спускатися до кінцевої. Обґрунтування чому завжди буде наведено нижче.

Прямий пошук алгоритму починається з початкової вершини, зворотний – з кінцевої. На рисунку 3.5 показано шлях, який буде знайдено між вершинами 2 і 3 у графі, розглянутому в попередньому підрозділі.

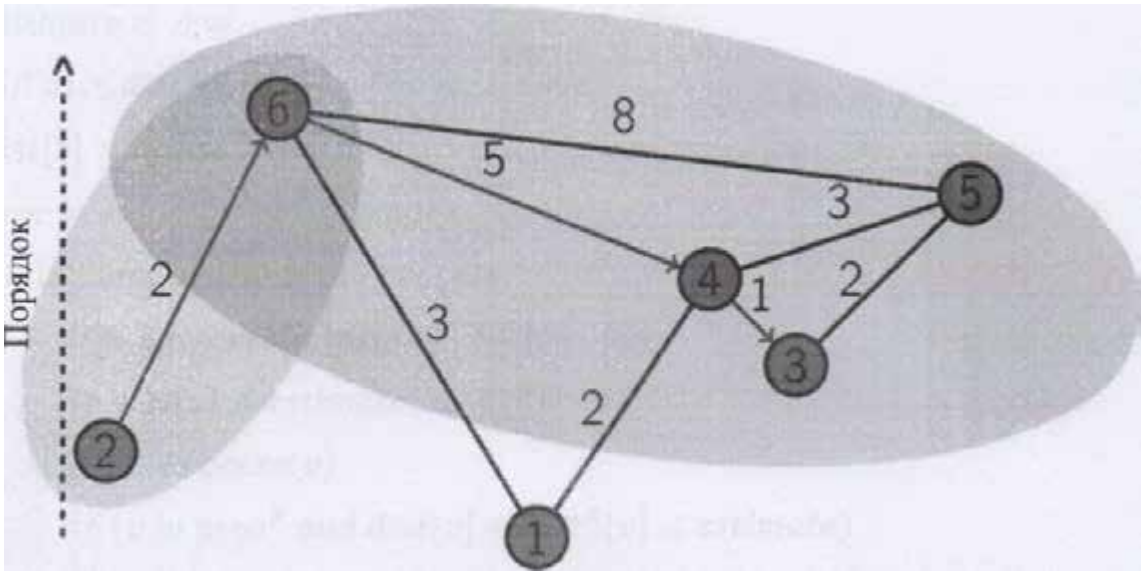

Рисунок 3.5 - Шлях від вершини 2 до вершини 3

Розглянуті особливості та обмеження двобічного алгоритму Дейкстри в контексті вашого випадку є важливими, оскільки вони дозволяють ефективно виконувати пошук маршруту в обробленому графі. Враховуючи тільки ребра, що йдуть вгору, і продовжуючи роботу алгоритму досягнення оптимального шляху між шуканими точками можна забезпечити ефективність та швидкість пошуку.

Додатково, обмеження на кількість вершин, що розглядаються, та врахування тільки тих вершин, відстань до яких менша за вже знайдену відстань між шуканими точками, сприяють зменшенню обчислювального навантаження та прискоренню процесу пошуку.

Наведемо псевдокод алгоритму: $estimate = +\infty$

$Fill\ dist, dist^R with\ +\ \infty\ for\ each\ node$

$dist[s]\ =\ 0, dist^R\ [t]\ =\ 0$

$proc\ =\ empty, proc^R\ =\ empty\ while\ there\ are\ nodes\ to\ process$

$v\ =\ ExtractMin(dist)$

$if\ dist[v]\ <\ estimate$:

$Process(y)$

$if\ (y\ in\ proc^R\ \ and\ dist[v]\ + dist^R[v] <\ estimate)$

$estimate\ =\ dist[v]\ +\ dist^R\ [v]$

$Repeat\ symmetrically\ for\ v^R\ return\ estimate$

3.1.4 Коректність

Спочатку дамо необхідні визначення.

Розширений граф G^+= (V, E^+) графу G = (V, E) це граф, що містить ті ж вершини V, що й граф G, ребра E та ребра скорочення додані на стадії передобробки.

Відстанню d (s, t) називатимемо довжину найкоротшого шляху між вершинами s, t ∈ V графу G — (V, E).

Порядком вершини r(v) назвемо порядковий номер під яким ця вершина була оброблена на стадії переобробки.

Шлях P: $v_1\ \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$розширеного графу G^+називається

зростаючим, якщо $\forall\ i < j \in N\ r\ (v_i) < r\ (v_j)$, і *спадним*, якщо $\forall\ i < j \in N\ r\ (v_i) > r\ (v_j)$.

Лема. $\forall\ s, t\ \in V$відстань $d^+\ (s, t)$в графі G^+ дорівнює відстані $d(s, t)$ в графі G.

Доведення. По-перше. Ребра тільки додаються до графа $G\ =\ (V,$ E). Тому $d^+\ (s, t) \leq d\ (s, t)$

По-друге, для будь-якого доданого ребра скорочення (u, w), існує шлях и$u \to v \to w$, довжини $l(u, w)$ = I $(u, v) + l(v, w)$, отже, $d^+(s, t)$ не може бути меншою від $d(s, t)$.

Таким чином ми довели, що $d^+(s, t) = d(s, t)$.

Обґрунтування двобічного пошуку Дейкстри.

Лема. $\forall s, t \in V$, якщо існує найкоротший маршрут P_{st}в розширеному графі G^+= (V, E^+) то існує вершина v така, що $P_{st} = P_{sv} + P_{sv}$ і P_{sv}- зростаючий, а P_{vt}- спадний.

Доведення. Нехай існує найкоротший шлях між s, t P_{st}. P_{st}: $s \to u_1 \to u_2 \to \cdots \to u_k \to \cdots \to t$.

Припустимо, що існує така вершина u_k , що $r(u_{k-1}) > r(u_k) < r(u_{k-1})$ - назвемо її локальним мінімумом. Можемо побачити на рисунку 3.6.

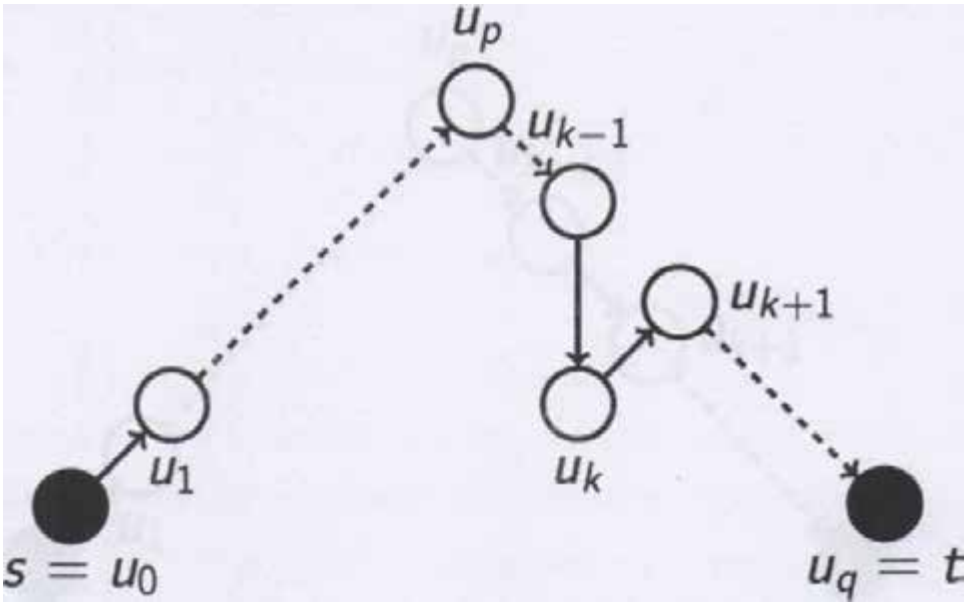

Рисунок 3.6. - u_k, $r(u_{k-1}) > r(u_k) < r(u_{k-1})$

Тоді u_k було оброблено раніше ніж u_{k-1} і u_{k+1} і було б знайдено ребро скорочення(u_{k-1}, u_{k+1}).

В такому випадку існує два варіанти.

- Ребро скорочення(u_{k-1}, u_{k+1}) не додається, оскільки існує коротший шлях між (u_{k-1}, u_{k+1}). Тоді наш шляхP_{st}: $s \to u_k \to \cdots t$ не є найкоротшим. Ми дійшли до протиріччя рис. 3.7;

- Коротшого шляху між (u_{k-1}, u_{k+1}) не існує. Тоді додається ребро скорочення. В цьому випадку шлях P_{st} буде проходити через додане ребро скорочення і локальний мінімум u_k зникне зі знайденого маршруту рис. 3.8.

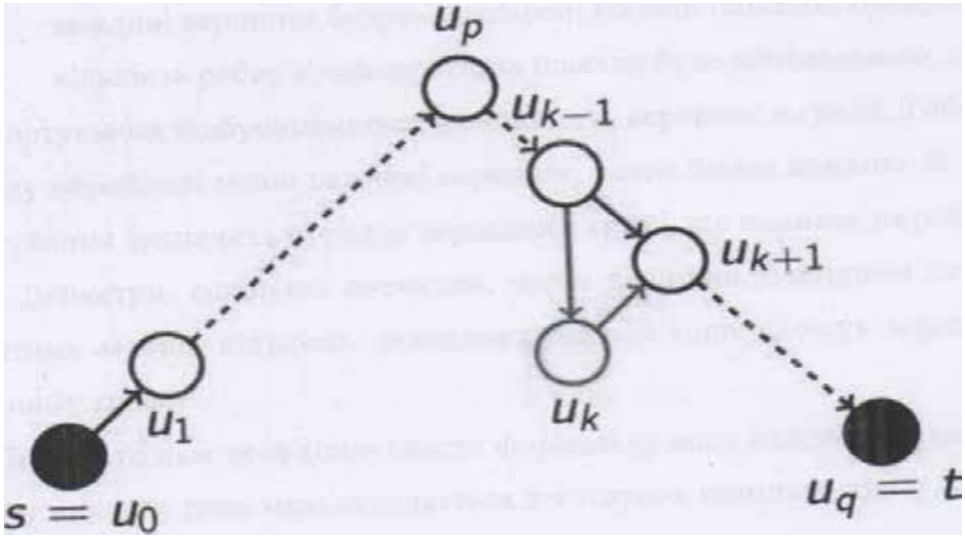

Рисунок 3.7 - Між (u_{k-1}, u_{k+1}) знайдено коротший шлях

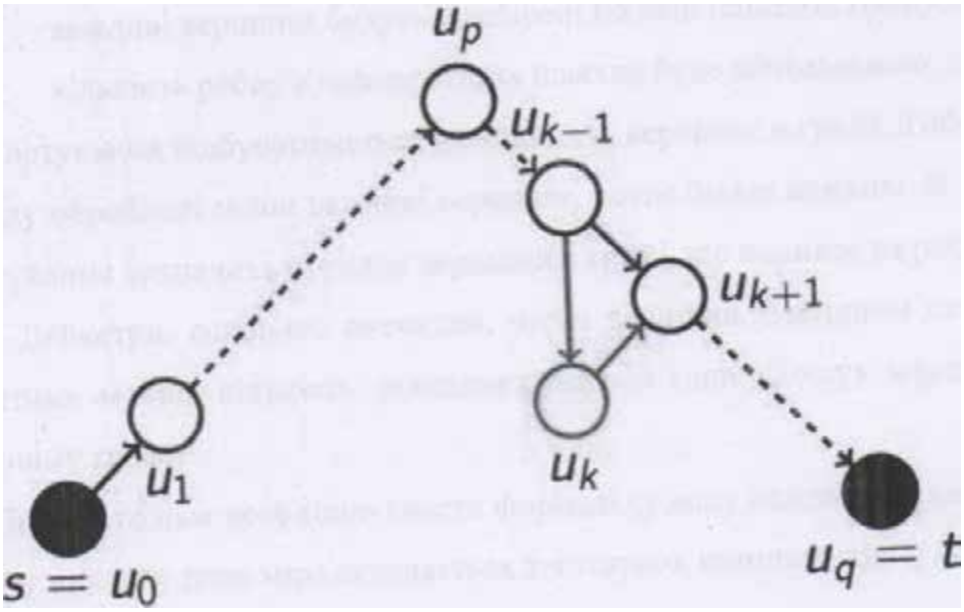

Рисунок 3.8 - Використання знайденого ребра скорочення

Отже, ми довели, що якщо існує найкоротший маршрут то він спочатку зростатиме до якоїсь вершини, а потім спадатиме до кінцевої [22].

3.1.5 Послідовність обробки вузлів

Не залежно від послідовності обробки вершин, алгоритм переобробки графа працюватиме належним чином, що забезпечить правильний результат. Проте, варто відзначити, що порядок обробки може вплинути на вигляд графа і швидкість роботи алгоритму пошуку. Тому важливо виконати сортування вершин перед обробкою, щоб врахувати вагомість кожної з них.

Наша мета полягає в розробці сортування, яке враховує такі критерії:

Мінімізація кількості доданих ребер.

Розподіл важливих вершин по всьому графу.

Мінімізація кількості ребер у найкоротших шляхах.

Сортування відбуватиметься відповідно до вагомості кожної вершини в графі. Таким чином, спочатку будуть оброблені менш важливі вершини, а потім більш важливі. Це визначить порядок вершин в графі, що впливатиме на роботу алгоритму Дейкстри, оскільки менша кількість знайдених шляхів буде проходити через вершини з меншою вагомістю.

Для досягнення цієї мети ми вводимо формальну міру важливості кожної вершини, що складається з чотирьох компонентів, зокрема різниці ребер, кількості оброблених сусідів, покриття ребрами скорочення та рівня вершини. Мета цих критеріїв полягає у забезпеченні мінімізації кількості ребер у розширеному графі.

Розраховується за формулою.

$$ed(v) = s(v) — in(v) — out(v),$$

де $ed(v)$ - різниця ребер для вершини V,

$s(v)$ - кількість доданих ребер скорочення після обробки вершини V,

$in(v)$ - кількість ребер, що входять в вершину V,

$out(v)$ - кількість ребер, що виходять з вершини V.

Пріоритетним є обробка вершин з низьким значенням різниці ребер, оскільки такі вершини вносять мінімальний вплив на граф, і кількість найкоротших шляхів, які проходять через них, обмежена або навіть відсутня.

Один із проблемних аспектів полягає у розрахунку компоненти c(v), яка відповідає за кількість доданих ребер скорочення після обробки вершини. Цей розрахунок вимагає виконання процедури пошуку свідків, що призводить до істотного збільшення часу роботи алгоритму, особливо для великих графів з більше ніж 100 000 вершинами та ребрами. Наприклад, для графа Києва з близько 160 000 вершин і 175 000 ребер процедура сортування у багатопотоковому режимі займає понад 1,5 години, а в однопотоковому - близько чотирьох годин.

Щодо кількості оброблених сусідів, головною метою є рівномірне поширення важливих вершин по всьому графу. Цей критерій враховує кількість вже оброблених ребер, до або від яких можна потрапити до вершини v. Чим менше значення критерію, тим раніше слід обробити вершину, і навпаки.

Щодо критерію покриття ребрами скорочення, його метою є відображення важливості вершин, від яких залежить багато інших вершин. Він позначається як sc(v) для вершини v і розраховується як кількість вершин w, для яких після обробки v буде додано ребро скорочення до чи від w. Чим більше значення критерію, тим більш важливою є вершина v.Так як і з попередніми критеріями, чим менше значення cv(v) тим раніше необхідно обробляти вершину.

Рівень вершини.

Рівень вершини v означає верхню межу кількості ребер в найкоротшому шляху від будь-якої вершини s в розширеному графі G^+= (V, E^+).

Спочатку рівень усіх вершин прирівнюється до 0. При обробці вершини v до сусідньої вершини и ми можемо потрапити або пройшовши вершину v, або не проходячи її. Тому рівень вершини и визначається як максимум між рівнем и і рівнем вершини $v + 1$. В результаті після обробки вершини перераховується рівень сусідніх вершин наступним чином:

$$L(u) = max(L(u), L(v) + 1)$$

Міра важливості.

Після обрахунку всіх чотирьох критеріїв можемо формально визначити міру важливості вершини v:

$$L(v) = ed(v) + cn(v) + sc(v) + L(v)$$

Зазначена міра є евристичною, і ми можемо експериментувати з коефіцієнтами для кожної характеристики, щоб проаналізувати, як змінюється час переобробки і час пошуку маршрутів. Проте в поточній реалізації було вибрано рівність вагомості всіх критеріїв [23].

3.2 Архітектура системи

Підготовлений програмний продукт побудований на платформі .NET з використанням мови програмування C# та технології WPF. Для зберігання даних використовується СУБД Microsoft SQL Server. Реалізовані алгоритми можуть бути застосовані як веб-сервер, а графічний інтерфейс було розроблено для візуалізації роботи та для тестування.

Давайте розглянемо архітектуру бази даних та серверного застосунку детальніше..

3.2.1 Архітектура бази даних

Схема бази даних зображена на рисунку 3.10.

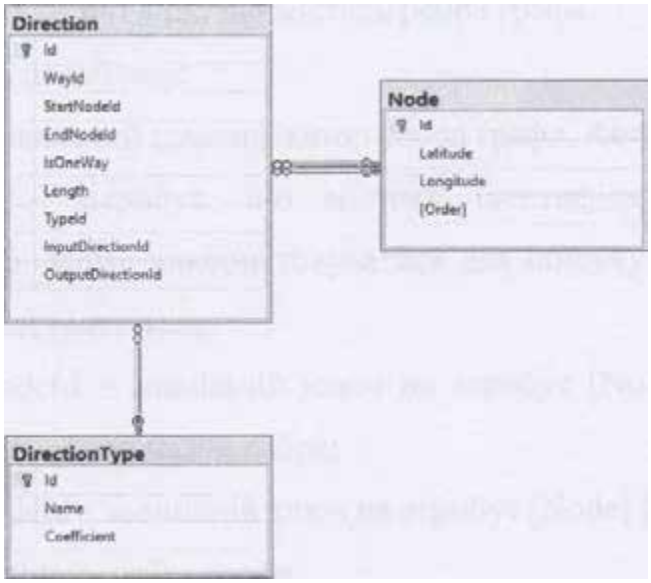

Рисунок 3.10 - Схема бази даних

Дана база даних містить дві таблиці - "Node" та "Direction", які відповідають вершинам та ребрам графа відповідно.

Таблиця "Node" містить такі атрибути:

- Id: Унікальний ідентифікатор вершини, що збігається з ідентифікатором вершини в даних OpenStreetMap.

- Latitude: Географічна широта точки на карті, пов'язана з даними вершини.

- Longitude: Географічна довгота точки на карті, пов'язана з даними вершини.

- Order: Атрибут, який використовується в процесі передобробки методу "Contraction hierarchies" для визначення порядку обробки вершини.

Таблиця "Direction" містить наступні атрибути:

- Id: Унікальний ідентифікатор ребра графа з автоінкрементним значенням.

- Wayld: Ідентифікатор дороги з даних OpenStreetMap, який може використовуватися для пошуку адрес на карті.

- StartNodeld: Зовнішній ключ, який посилається на атрибут "Node.Id" та вказує на початкову вершину ребра.

- EndNodeld: Зовнішній ключ, який посилається на атрибут "Node.Id" та вказує на кінцеву вершину ребра.

- IsOneWay: Булевий атрибут, який вказує на односторонній або двосторонній рух вулицею.

- Length: Довжина ребра в кілометрах.

- Typeld: Зовнішній ключ, який посилається на атрибут "DirectionType.Id".

- InputDirectionld, OutputDirectionld: Атрибути, які вказують на ідентифікатори таблиці "Direction" і заповнюються під час обробки "Contraction hierarchies". Вони потрібні для відновлення початкового маршруту.

DirectionType — таблиця, що містить типи ребер графа. Містить 13 значень - таблиця 3.1.

I	Name	Co
1	motor	12
2	Trunk	11
3	primar	10
4	secon	9
5	tertiar	8
6	unclas	7
7	reside	6
8	Servic	5
9	motor	4
1	trunkji	3
1	primar	2
1	secon	1
1	Shortc	100

Таблиця 3.1 - Значення DirectionType

Id - унікальний ідентифікатор;

Name - назва типу;

Coefficient - «важливість» ребра. Чим вище значення - тим важливішій вулиці відповідає дане ребро. Очевидно, при навігації краще обирати великі дороги з типом магістраль - motorway, а не допоміжні - service. Дані значення використовуються в алгоритмі A* при виборі напрямку руху хвилі: враховується відстань до кінцевої вершини і тип дороги.

3.2.2 Архітектура серверного застосунку

На рисунку 3.9 зображено загальну схему розробленої системи.

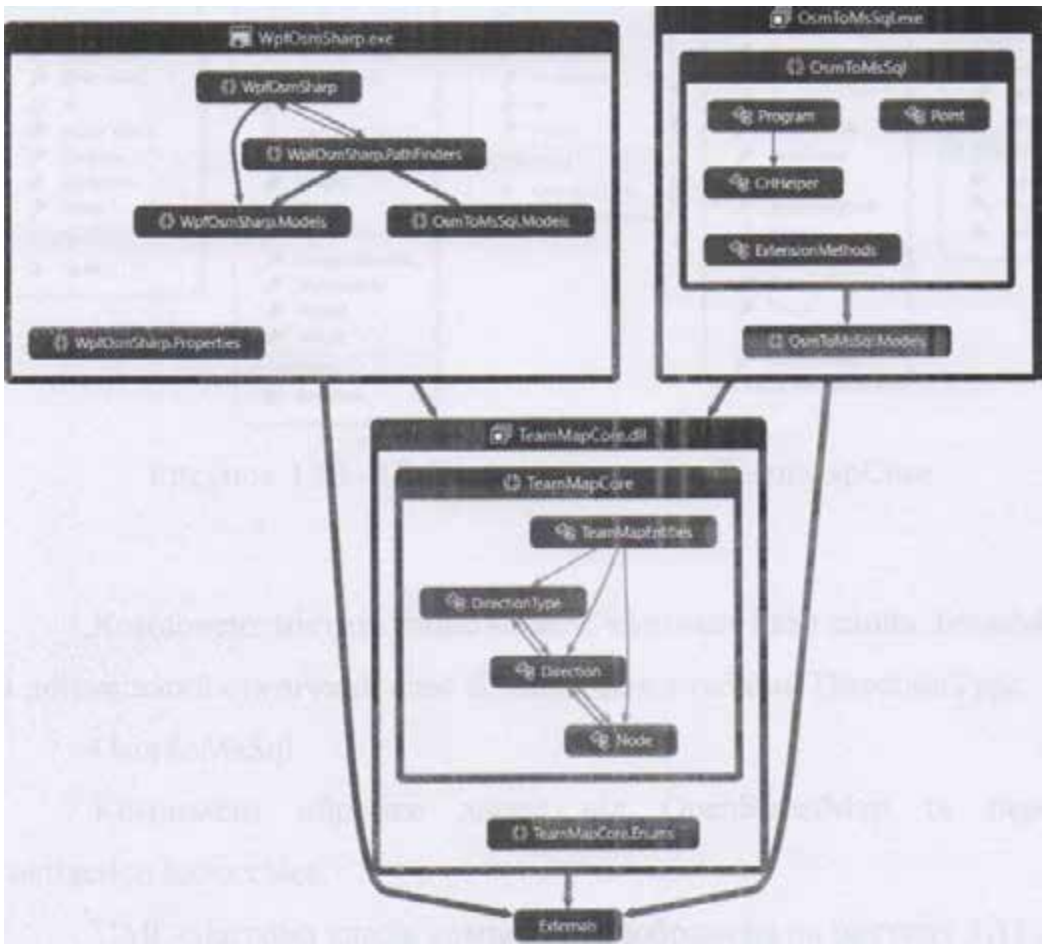

Рисунок 3.9 - Архітектурна схема системи

Розроблене рішення складається з трьох ключових компонентів. WpfOsmSharp відповідає за агент роутингу, побудований на основі фреймворку Windows Presentation Foundation [25]. OsmToMsSql є агентом, що відповідає за генерацію бази даних і обробку графа доріг. Нарешті, TeamMapCore є компонентом, який забезпечує зв'язок з базою даних, містить контекст бази даних і надає інтерфейс для отримання, зміни та додавання даних в базу даних.

Розглянемо детальніше кожен компонент.

TeamMapCore

UML-діаграма класів компонента зображена на рисунку 3.10.

.

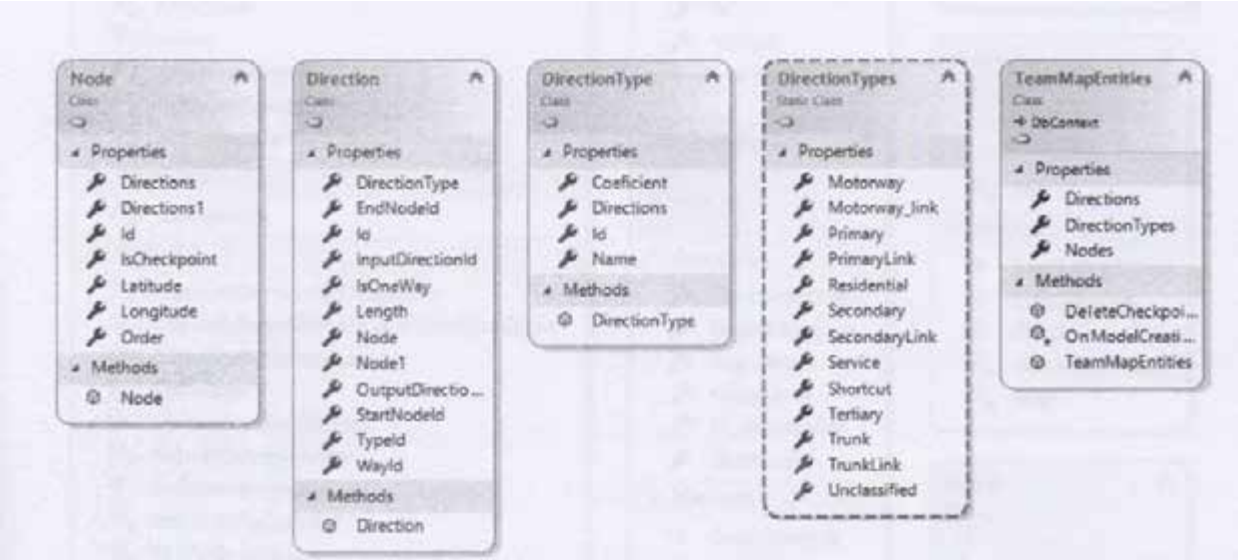

Рисунок 3.10 - UML-діаграма класів TeamMapCore

Компонент містить набір моделей та контекст бази даних під назвою TeamMapEntities, а також допоміжний статичний клас, який містить значення таблиці DirectionType.

OsmToMsSql є відповідальним за обробку даних з OpenStreetMap та виконання передобробки методом Contraction hierarchies.

UML-діаграма класів цього компонента надана на рисунку 3.11.

Ключовими класами в цьому компоненті є Program і CHHelper.

Клас Program відповідає за генерацію бази даних із XML-файлу OpenStreetMap.

Структура картографічного XML-файлу OpenStreetMap та її обробка.

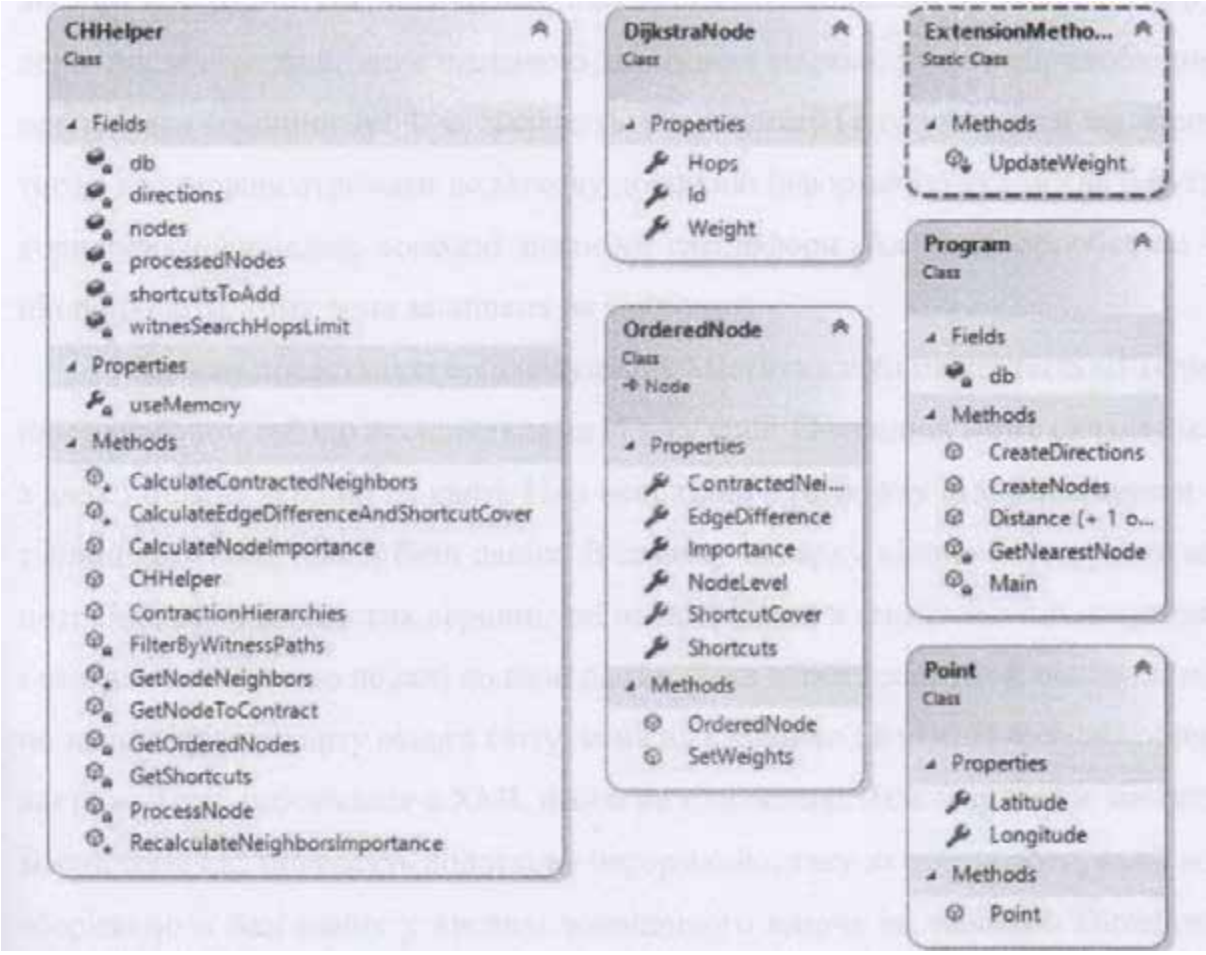

Рисунок 3.11 - UML-діаграма класів OsmToMsSql

Ключовим тегом є тег osm, який містить усю інформацію, що міститься в документі. Національні теги для нас - це теги node та way. В додаток до них у документі також містяться теги relation, але вони наразі не використовуються.

Тег node включає кілька атрибутів, зокрема id, latitude, longitude. Саме ці атрибути використовуються нами для створення вершин графа. Помимо атрибутів, тег node може містити масив тегів tag, які містять додаткову інформацію про точку на карті, наприклад, тип дороги, на якій вона знаходиться, або інформацію про будівлю, до якої вона відноситься, якщо вона не є частиною дорожньої мережі. Однією з проблем при обробці даних є їх перенасиченість, тому найважливіше - відфільтрувати потрібну нам інформацію. Наприклад, ми додаємо до графа тільки ті вершини, що належать до дорожньої мережі, та пропускаємо ті, які не відносяться до неї. Теги tag допомагають вирішити цю проблему. Крім цього, вони можуть надавати додаткову дорожню інформацію,

наприклад, про дорожні знаки або світлофори. Однак обробка цих даних не є нашою основною метою, тому це залишається на майбутнє.

Тег way представляє собою вулицю і включає масив тегів nd з атрибутом ref, який посилається на id тегу node. Очевидно, що вулиця може складатися з двох або більше вершин на карті. Нам необхідно створити записи в таблиці Direction бази даних з тегів way. Основною проблемою тут є фільтрація тільки тих вершин, які зустрічаються в завантаженому документі, і які ще не були додані до бази даних. Це може виникнути через те, що ми не завантажуємо карту всього світу, і тому інформація в XML-файлі не є повною. Також тег way може містити масив тегів tag, які містять додаткову інформацію, наприклад, про тип дороги, який ми зберігаємо в базі даних у вигляді зовнішнього ключа на таблицю DirectionType, а також про назву вулиці тощо.

Теги relation містять масив тегів member з атрибутом type, який може мати значення way або node. Таким чином, тег relation об'єднує в собі шляхи та вершини. Він може описувати різноманітні об'єкти, наприклад, будівлі. У нашій роботі дані тегів relation наразі не використовуються.

Однією з головних складнощів при обробці картографічного файлу є його значний обсяг даних. Наприклад, файл, який містить карту Києва, складається з більше ніж 3,5 мільйонів рядків. Клас CHHelper виконує обробку бази даних у методі Contraction hierarchies. Саме обробка цих великих обсягів даних створює численні труднощі.

По-перше, звернення до бази даних займає значну кількість часу. Через це важливо працювати з даними у пам'яті. Максимальна кількість даних, яку можна обробити в пам'яті, значно перевищує обмеження, яке виникає при роботі з базою даних. Однак при збільшенні площі карти, яку необхідно обробити, максимальна кількість може бути досягнута, що призводить до нових викликів, пов'язаних з алгоритмом обробки і подальшим пошуком. Це стає об'єктом подальших досліджень.

По-друге, багато обчислень в алгоритмі є незалежними одне від одного. Наприклад, для розрахунку початкового порядку вершин ми повинні обрахувати

4 характеристики для всіх вершин. Особливо обчислювально витратним є розрахунок характеристики "різниця ребер", оскільки це вимагає знаходження всіх ребер скорочення, які будуть додані при обробці, а це передбачає багаторазовий запуск "дорогої" процедури Witness search для пошуку можливих коротших маршрутів. Хоча порядок обробки може змінюватись під час процесу, саме початкове сортування вершин може виконуватись паралельно, оскільки вони незалежні між собою. Варто користуватись цією можливістю, оскільки це може зменшити час сортування у 4 рази.

Процедуру обробки не можна виконувати паралельно, оскільки це може призвести до некоректного порядку вершин, що негативно вплине на навігацію по графу. "Найдорожчими" операціями у процедурі обробки є вибір вершини для обробки, який виконується за допомогою методу GetNodeToContract в UML-діаграмі, та фільтрація знайдених ребер скорочення у методі FilterByWitnessPath.

Обидві ці процедури можна оптимізувати, запускаючи, в першому випадку, перерахунок порядку одразу для п вершин в черзі, де п відповідає кількості ядер процесора машини, на якій виконується обробка. У другому випадку можна паралельно запускати п процедур Witness search. Процедуру Witness search можна також покращити, запускаючи алгоритм Дейкстри не для всіх знайдених ребер скорочення окремо, а об'єднуючи їх в групи за вершиною початку і зупиняючи алгоритм при досягненні найбільш віддаленої вершини або при досягненні іншої умови зупинки. Це може зменшити кількість запусків процедури в 4-25 разів.

Складність алгоритму обробки росте зі збільшенням степені вершини графа. Тобто, якщо лише 2 ребра інцидентні вершині, то може бути додано лише 2 ребра скорочення (в обох напрямках у випадку двонаправлених ребер), і тільки для них необхідно запускати Witness search. Така ситуація цілком можлива і навіть поширена на початку обробки графа. Тому на початку обробка проходить досить швидко. Але при додаванні ребер скорочення степені вершин графа зростають, і кількість запусків Witness search також зростає. Саме тому важливе значення має максимальна кількість вершин від початкової і кінцевої вершин у

процедурі Witness search. Цей показник є дуже важливим, коли кількість ребер інцидентних вершині може досягати 1000. Звісно, зменшення цього значення може погіршити якість результуючого графа, але значно прискорити алгоритм обробки. Тому в процесі обробки можна змінювати його значення від 5 на початку до 2 в кінці.

Для розуміння важливості оптимального використання ресурсів комп'ютера наведемо отримані результати часу обробки. У випадку повної карти Києва маємо близько 160 000 вершин. Якщо використовувати послідовну обробку при сортуванні вершин, сам алгоритм сортування працює приблизно 12 годин. При використанні паралельної обробки час роботи зменшується до 3 годин.

Висновки за розділом 3

У розділі детально розглянуто алгоритм Contraction hierarchies. Складові його реалізації - сортування вершин для передобробки, witness search, механізм обробки вершини, модифікований двонаправлений алгоритм Дейкстри - були описані і доведено коректність алгоритму. Також у розділі була розглянута архітектура розробленого програмного продукту, включаючи струк

туру бази даних та трьох складових компонентів серверного додатку. У додаток були представлені UML-діаграми класів розробленого програмного продукту та діаграма залежностей.

4.РЕЗУЛЬТАТИ РОБОТИ

4.1 Обробка бази даних

Процес створення бази даних з XML-файлу OpenStreetMap вимагає незначного часу. Нижче наведені результати створення графу для карти Києва та прилеглих територій - див. Рисунок 4.1.

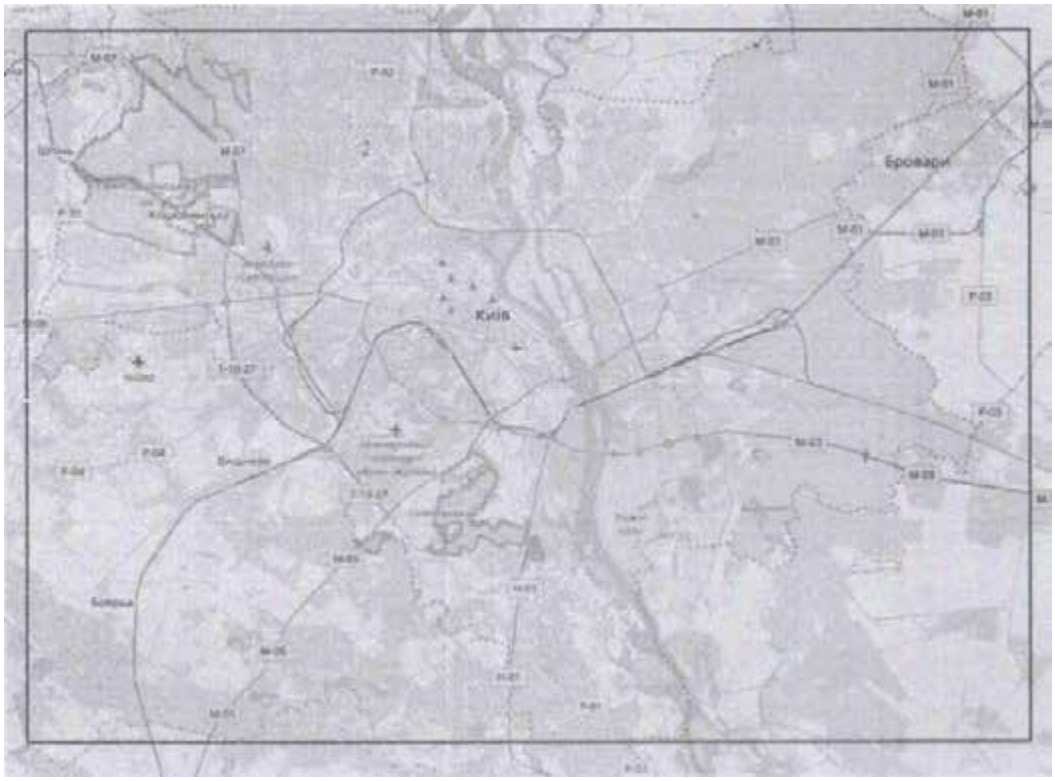

Рисунок 4.1 - Оброблена область Києва та приміських територій

Заповнення таблиці Node всіма значеннями з файлу (1 077 306 значень) займає 7 хвилин 43 секунди. Фільтрація ребер, заповнення таблиці Direction та видалення зайвих вершин займає 25 хвилин 20 секунд. Таким чином, процедура генерації графу доріг усього Києва зайняла 33 хвилини 3 секунди. Отримано 161 059 вершин та 175 167 ребер. Можна відмітити, що тривалість роботи лінійно залежить від розміру файлу, тобто від розмірів території. Це є задовільним результатом, оскільки дана операція виконується лише один раз.В межах даної роботи не було необхідності обробляти повну карту Києва, для тестування та

порівняння роботи алгоритмів цілком достатньо і її частини, тому було оброблено лише частину Києва - рисунок 4.2.

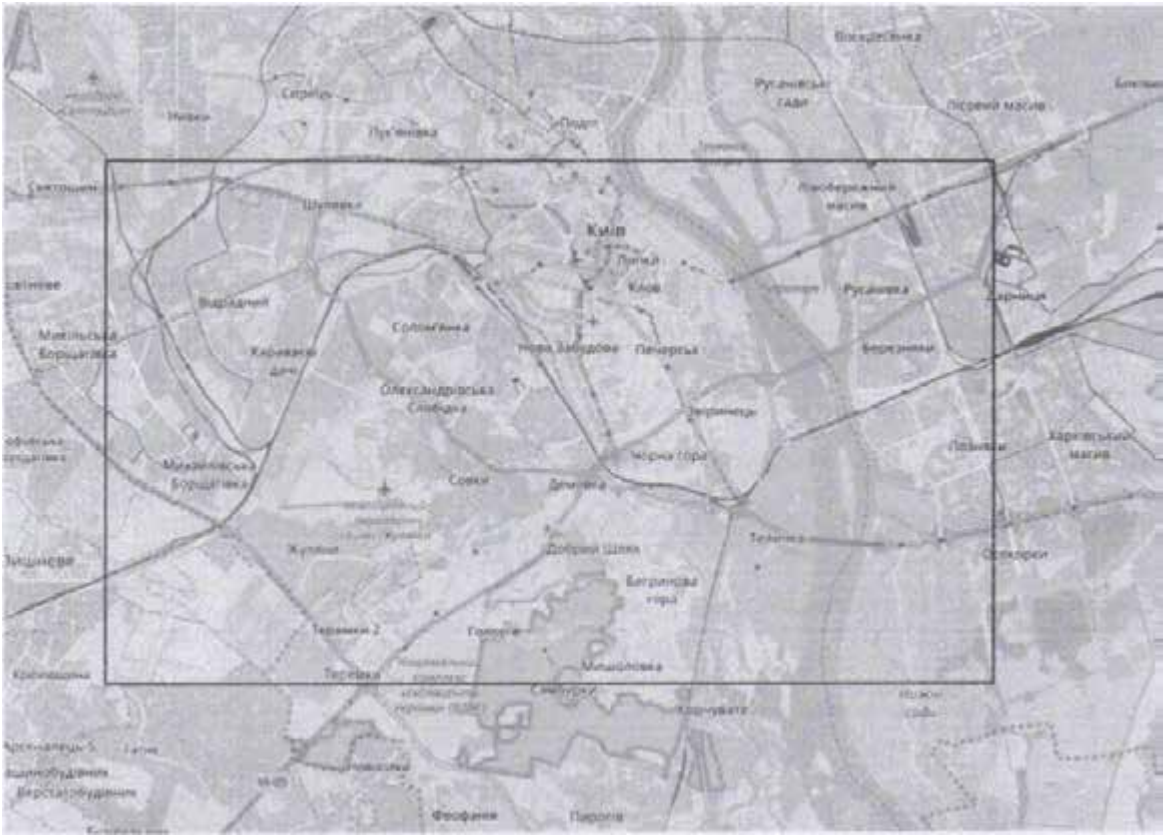

Рисунок 4.2 - Область, оброблена алгоритмом Contraction hierarchies

У відміченій області міститься 64 526 вершин та 69 536 ребер. Процес обробки цієї області зайняв близько 35 годин. Під час обробки було створено 190 793 ребра скорочення, з яких 67 077 були виявлені як дублікати. Дублікатами називаються ребра скорочення, які з'єднують однакові вершини. Їхнє утворення стає можливим, якщо процедуру Witness search припиняють при досягненні ланцюжків довжиною p. В нашому випадку значення p спочатку дорівнювало 5, поступово зменшувалося до 2, а для останніх 100 вершин становило 1. Це означає, що фільтрувалися лише ті ребра, які вже існують і мають більшу довжину, ніж існуючі. Така фільтрація виявилася досить ефективною, зменшивши кількість можливих ребер скорочення для окремих вершин із близько 4000 до близько 100.

Звісно, ми отримаємо більш якісний граф при великому значення n або взагалі при відсутності такої умови зупинки процедури Witness search (□ =

+∞), але це значно впливає на час обробки. Тому, в дослідницьких цілях достатньо згенерувати граф посередньої якості, що і було зроблено.

Дублікати були видалені з бази даних зі збереженням цілісності зв’язків. Таким чином в результуючій базі даних міститься 64 526 вершин та 193 252 ребер.

Відмітимо, що тривалість обробки залежить від розміру карти нелінійно і тривалість роботи алгоритму на карті всього Києва може сягати кількох тижнів.

4.2 Алгоритми пошуку

Для порівняння роботи алгоритмів пошуку маршрутів ми будемо запускати процедури пошуку на різних маршрутах. Важливо, щоб маршрути в обраній вибірці включали як короткі, так і довгі відстані, оскільки різні алгоритми можуть показувати різні результати на маршрутах різної довжини. При цьому можна зробити припущення, що алгоритм Contraction Hierarchies буде найефективнішим, і його перевага буде найбільш очевидною саме на довгих маршрутах, тоді як алгоритм A* може показати себе добре на коротких і прямих маршрутах.

Розглянемо по три короткі і довгі маршрути для аналізу результатів.

Маршрут №1. Прямий маршрут по проспекту Перемоги - від цирку до Бессарабською площі. Довжина 2,19 км.

Contraction hierarchies - рис. 4.3:

Рисунок 4.3 - Маршрут №1 - Contraction hierarchies

Час пошуку - 1,23 с.

Дистанція - 2,19 км.

A*:

Знайдений маршрут ідентичний попередньому.

Час пошуку - 0,25 с.

Дистанція 2,19 км.

A* двонаправлений:

Знайдений маршрут ідентичний попередньому.

Час пошуку - 0,31 с.

Дистанція 2,19 км.

Результат Google maps - рис.4.4.

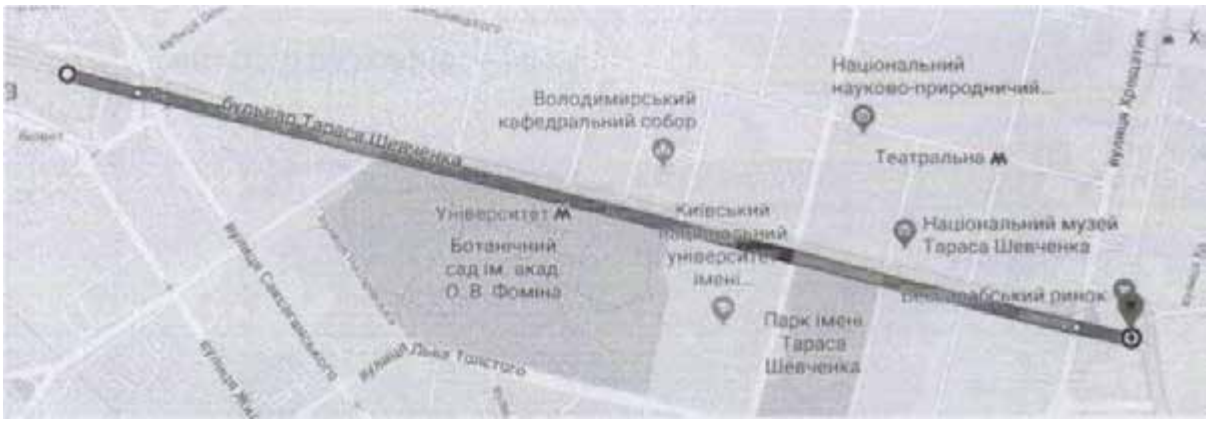

Рисунок 4.4 - Маршрут №1 - Google maps

Дистанція - 2,2 км.

Перший маршрут є найпростішим, і всі алгоритми показали однаковий результат пошуку. Найшвидшим у цьому випадку виявився алгоритм A*, оскільки йому потрібно було перебрати меншу кількість вершин, ніж алгоритму Contraction Hierarchies, а також не довелося витрачати час на переключення між потоками виконання, на відміну від двонаправленого алгоритму A*. Усі маршрути аналогічні маршруту, що показує Google Maps.

Маршрут №2. Маршрут від цирку до станції метро Олімпійська.

Довжина 3,33 км.

Contraction hierarchies - рис.4.5:

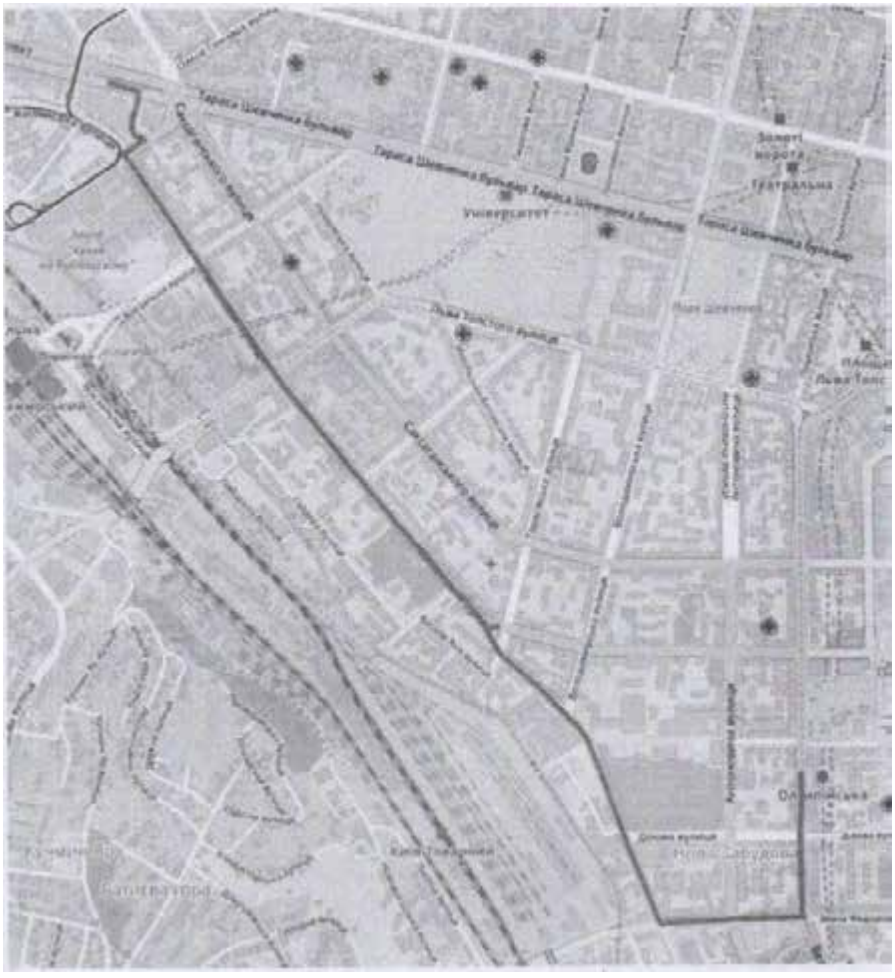

Рисунок 4.5 - Маршрут №2 - Contraction hierarchies

Час пошуку - 1,14 с.

Дистанція - 3,33 км.

A* - рис. 4.6:

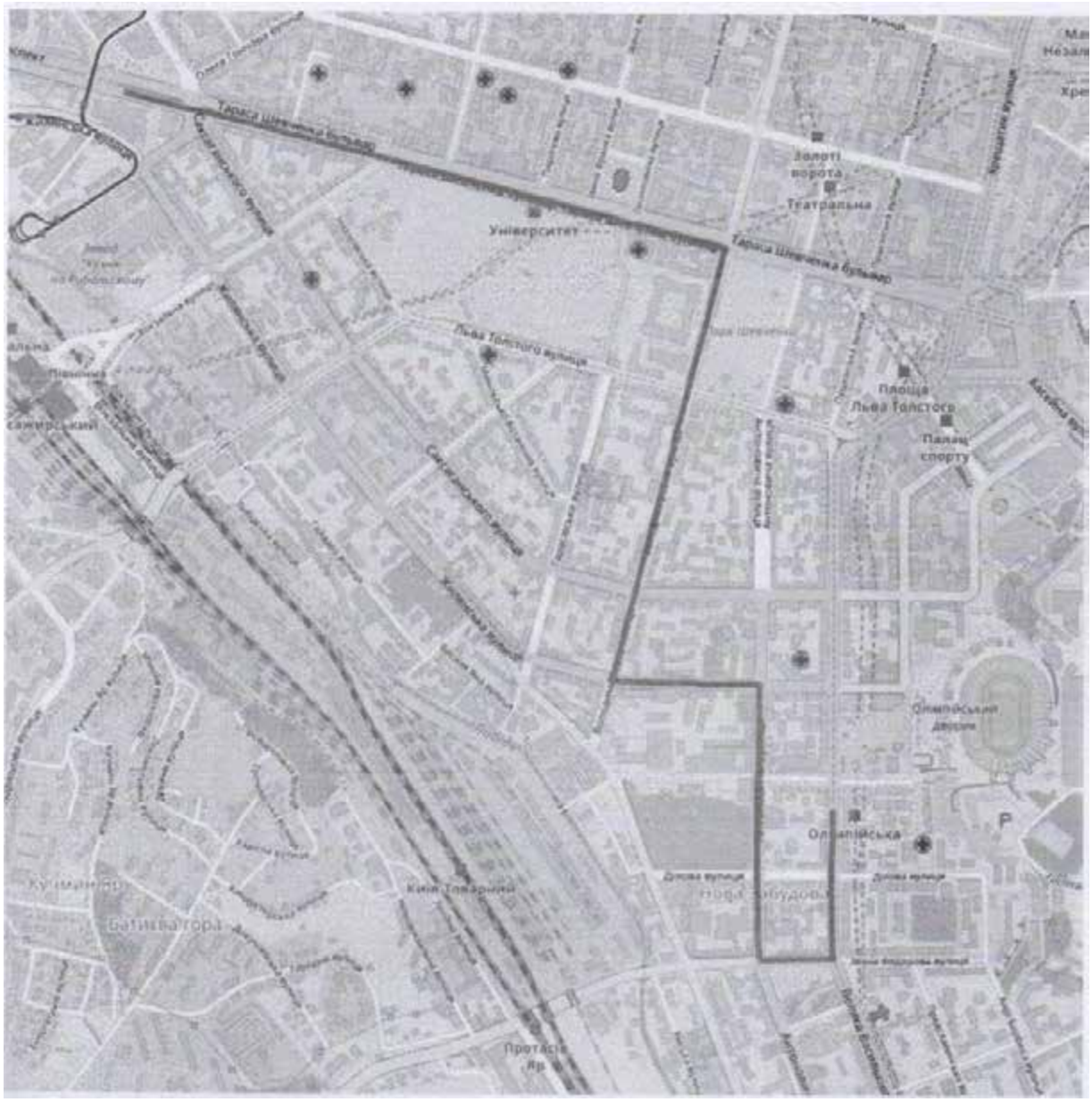

Рисунок 4.6 - Маршрут №2 - A*

Час пошуку - 0,98 с.

Дистанція -4,12 км.

A* двонаправлений - рис. 4.7:

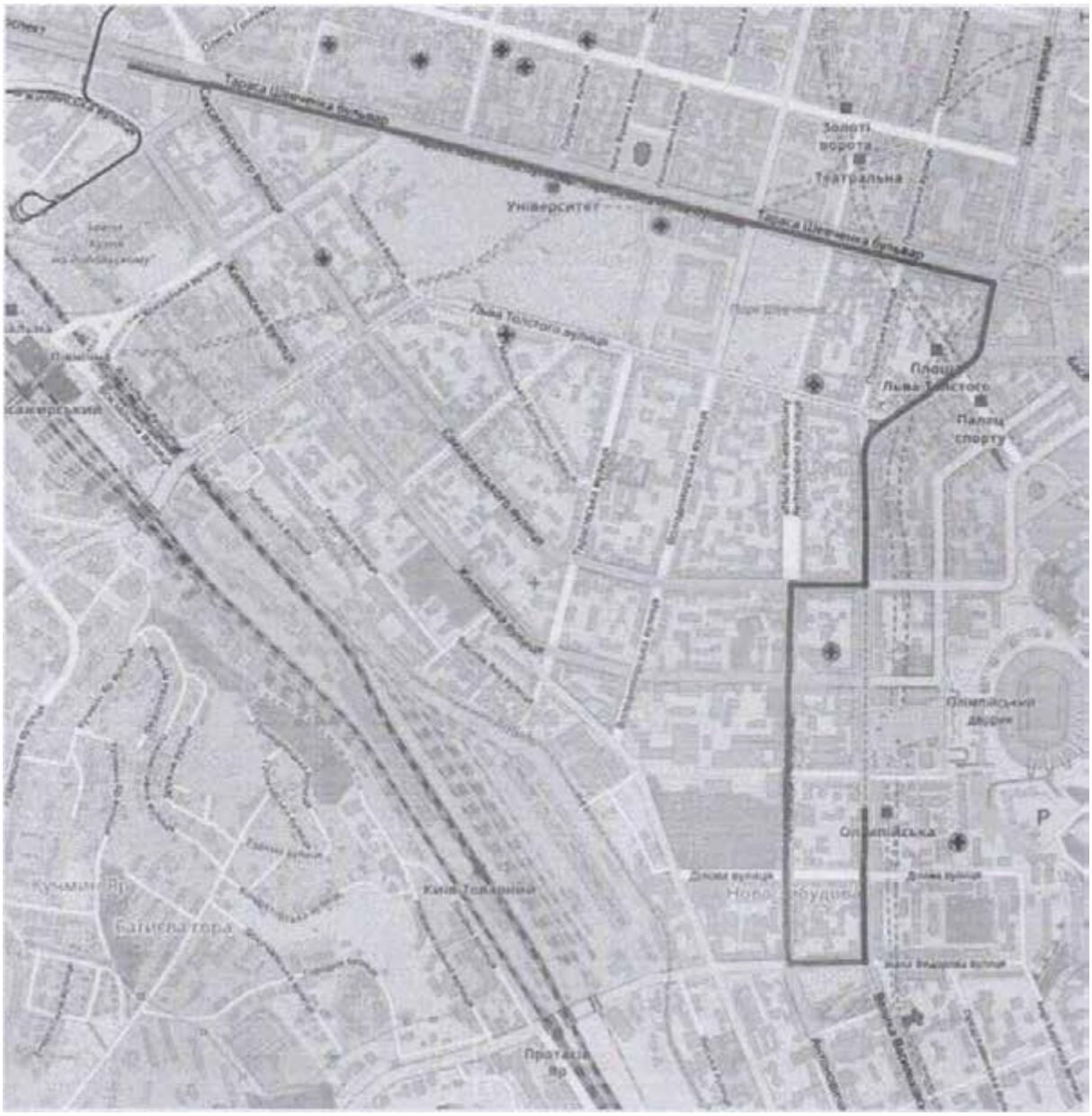

Рисунок 4.7 - Маршрут №2 - A* двонаправлений

Час пошуку -1,04 с.

Дистанція 4,48 км.

Результат Google maps - рис. 4.8.

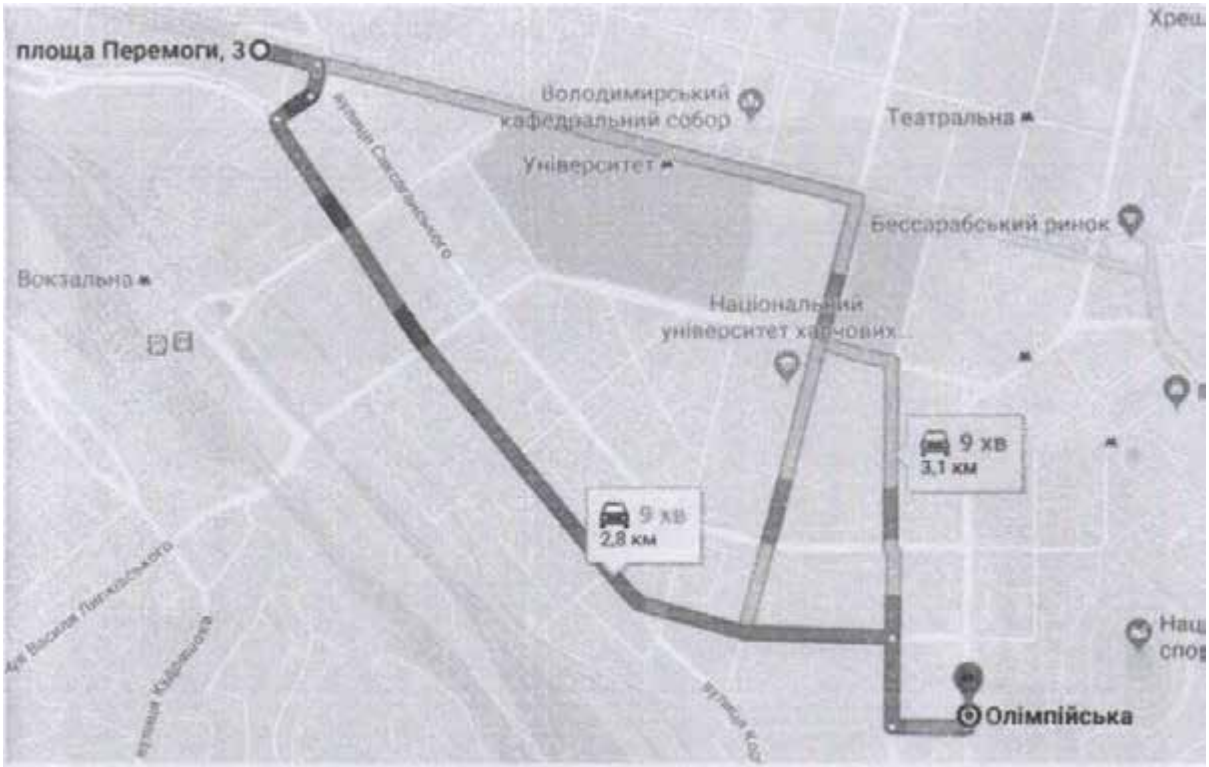

Рисунок 4.8 - Маршрут №2 – Google maps

Довжина - 2,8 км.

У даному випадку всі три маршрути виявилися різними. Це пояснюється тим, що в алгоритмах A* та A* двонаправлений враховується тип дороги, але враховується це по-різному. Алгоритм Contraction Hierarchies завжди шукає найкоротший шлях, незважаючи на тип дороги. Найкраще за часом відповіді знову проявив себе алгоритм A*, проте довжина його маршруту значно більша в порівнянні з результатом від алгоритму Contraction Hierarchies. Деякі дороги у цьому випадку виявилися еквівалентними, а час виконання у Contraction Hierarchies незначно перевищує час в інших алгоритмів. Тому можна припустити, що у другому випробуванні найкращим виявився алгоритм Contraction Hierarchies. Найгірше себе проявив A* двонаправлений, показавши середній час та значно більшу відстань.

Результат, який показує Google Maps, в даному випадку дещо кращий за результат від алгоритму Contraction Hierarchies. Як альтернативу, Google Maps пропонує маршрут, схожий на той, який був побудований за допомогою алгоритму A*.

Маршрут №3. Маршрут від перетину вулиць Лютеранської та Шовковичної до станції метро Кловська, Довжина - 2,23 км.

Contraction hierarchies - рис.4.9:

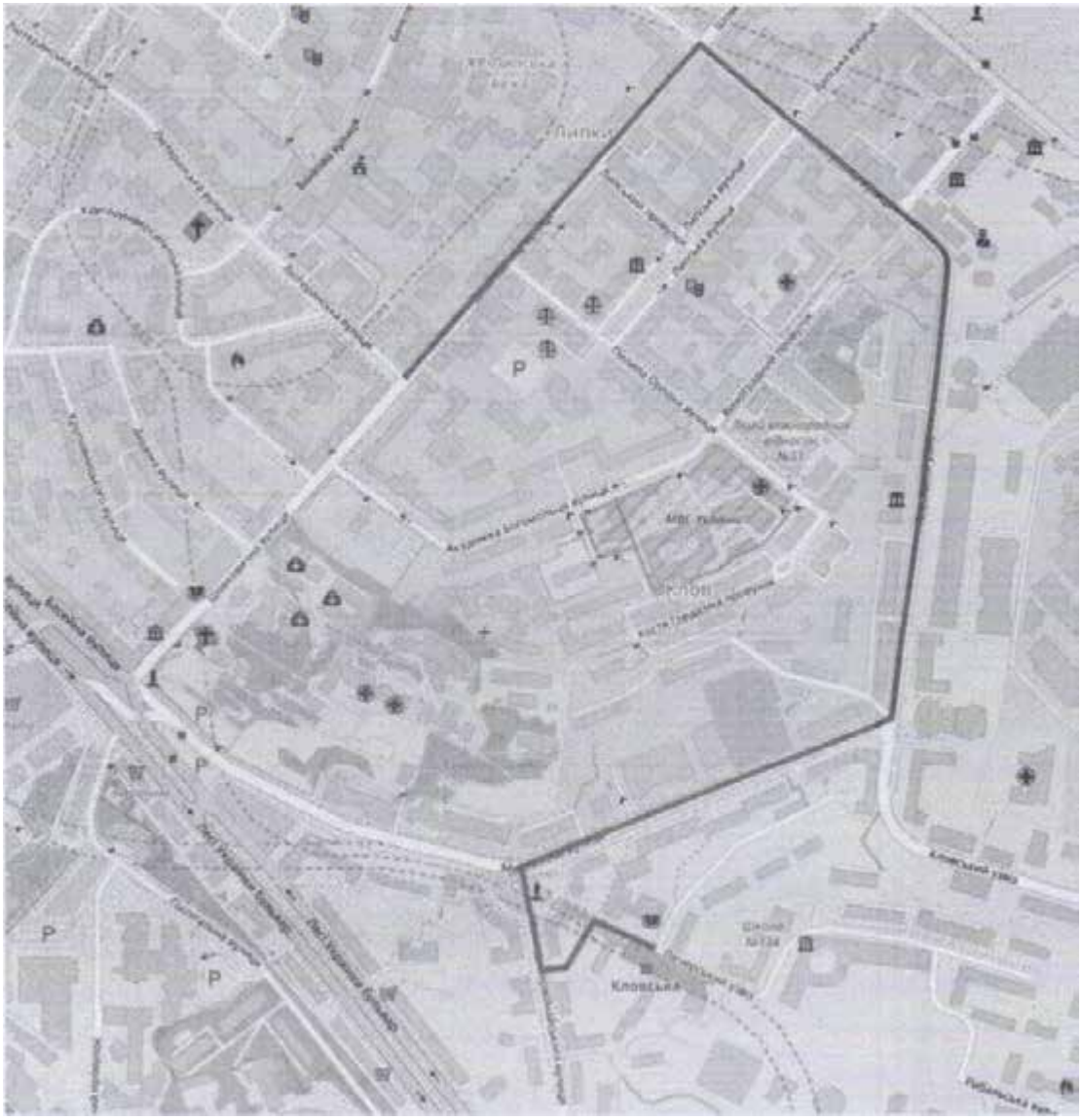

Рисунок 4.9 - Маршрут №3 - Contraction hierarchies

Час пошуку - 0,96 с.

Дистанція - 2,23 км.

А* - рис.4.10:

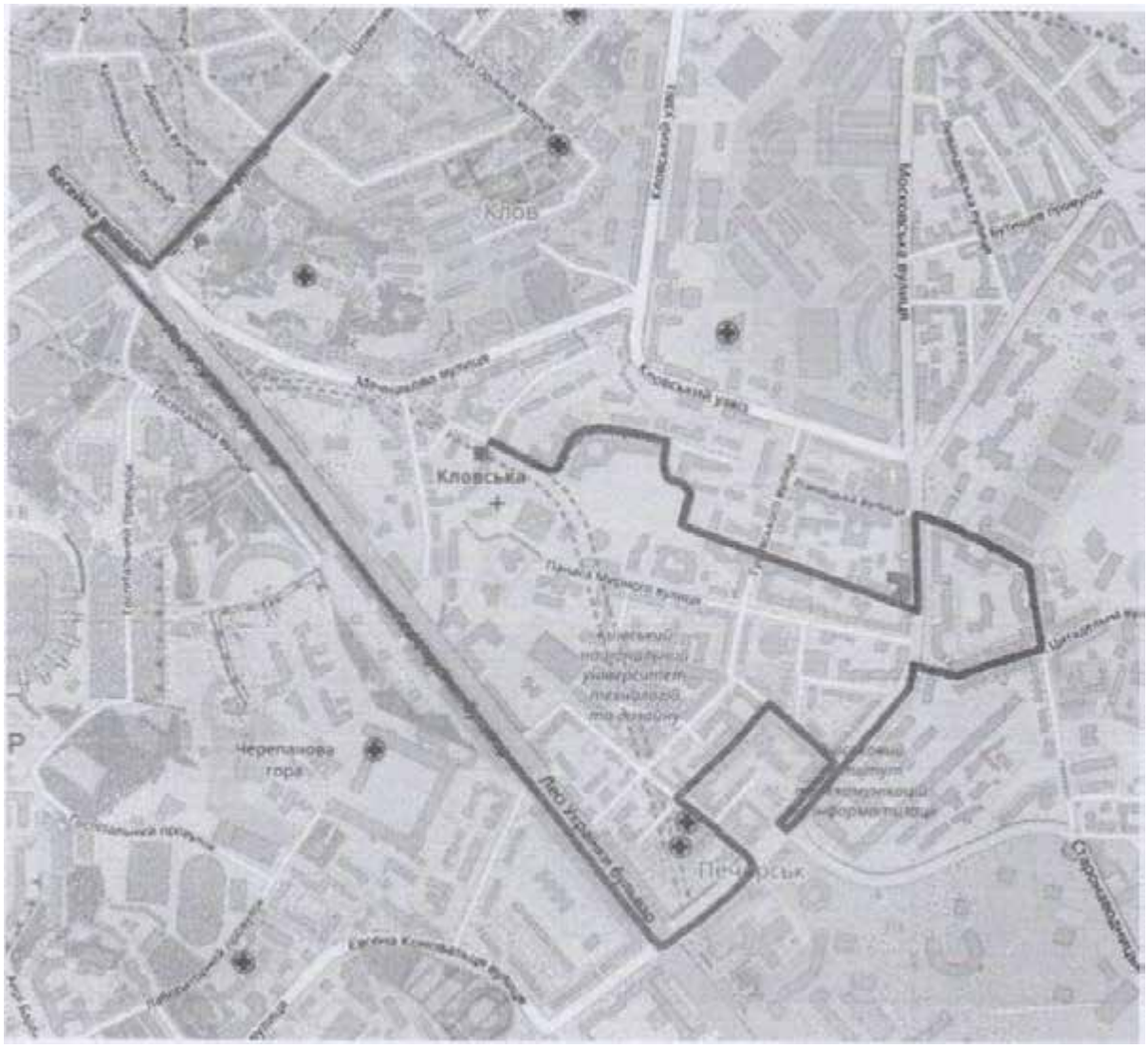

Рисунок 4.10 - Маршрут №3 - А*

Час пошуку -6,18 с.

Дистанція - 5,51 км.

A* двонаправлений - рис. 4.1:

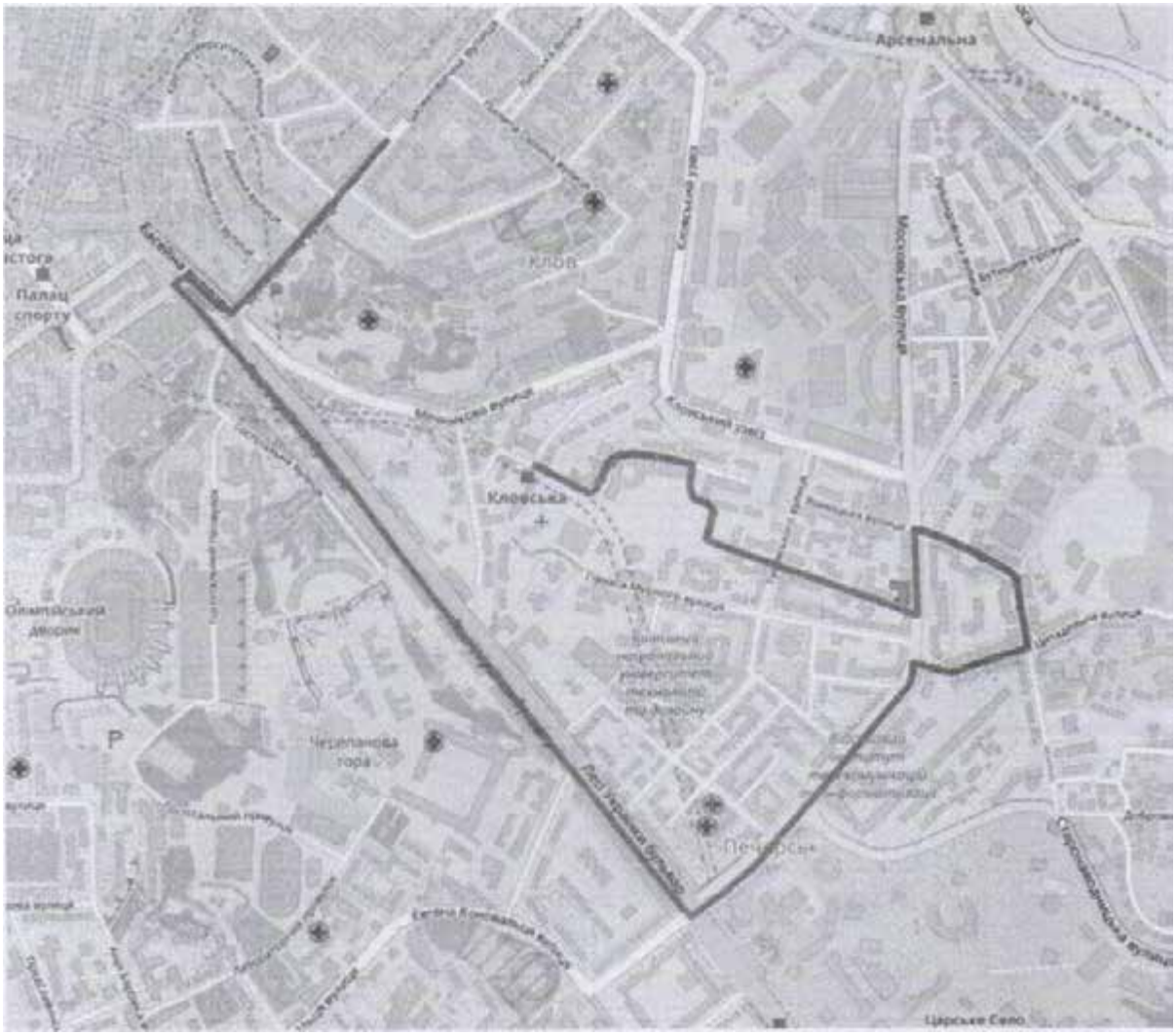

Рисунок 4.11 - Маршрут №3 - A* двонаправлений

Час пошуку - 16,52 с.

Дистанція 4,84 км.

Результат Google maps - рис.4.12.

РИСУНОК 4.12 - МАРШРУТ №3 - GOOGLE MAPS

Дистанція - 1,9 км.

У цьому випробуванні алгоритми A* показали дуже низьку ефективність як у відношенні часу, дистанції, так і якості маршруту при візуальній оцінці. Це може бути пояснено тим, що ці алгоритми намагаються прокласти маршрут по головним дорогам, у відміну від Contraction Hierarchies, який не враховує тип дороги, а просто будує маршрут на основі створеної ієрархії вершин.

Порівняно з Google Maps усі алгоритми показали гірші результати. Хоча відставання алгоритму Contraction Hierarchies було незначним.Маршрут

№4. від станції метро Шулявська до станції метро Деміївська.

Довжина - 8,78 км.

Contraction hierarchies - рис. 4.13:

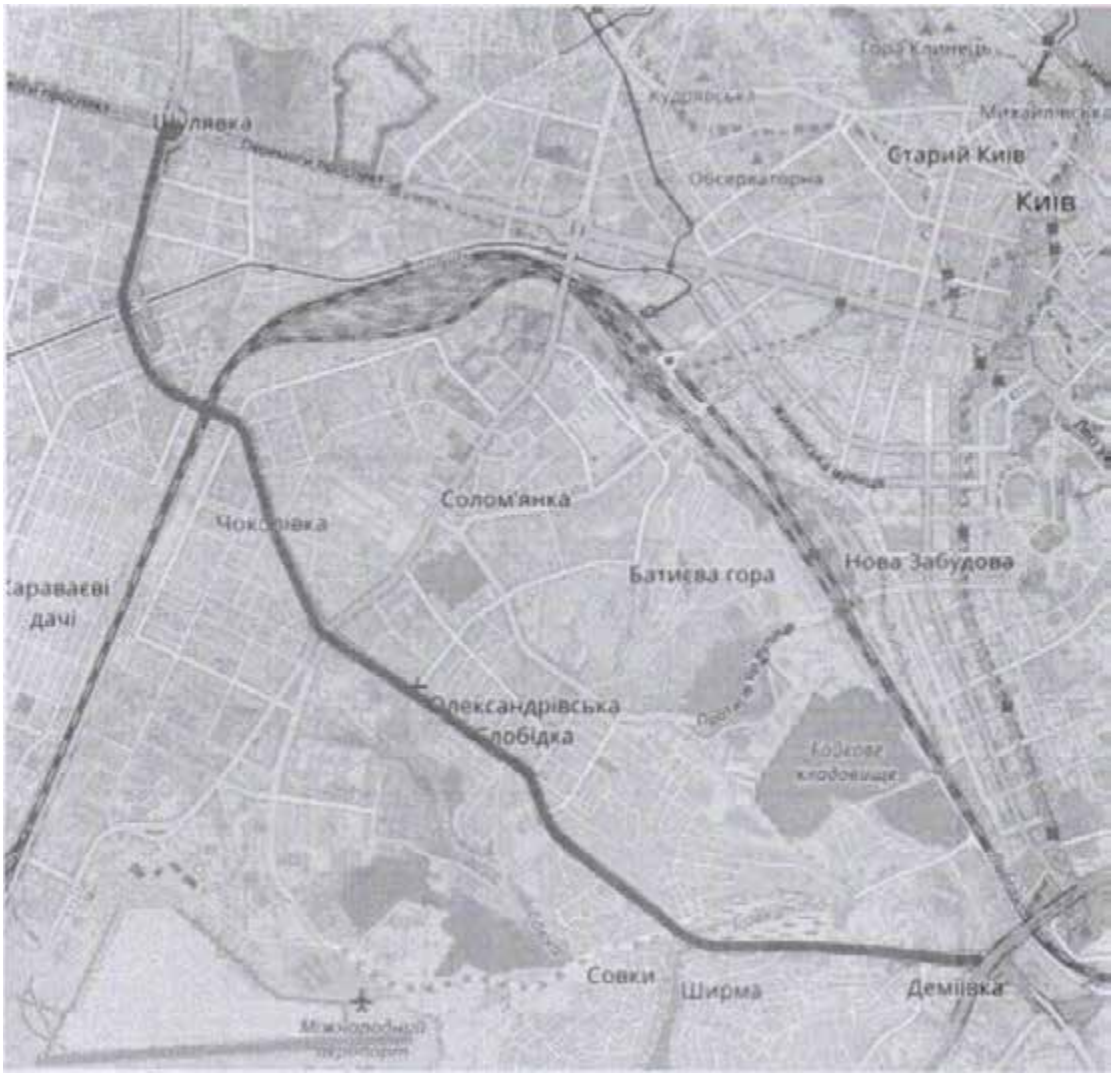

Рисунок 4.13 - Маршрут №4 - Contraction hierarchies

Час пошуку- 1,48 с.

Дистанція - 8,78 км.

A* - рис. 4.14:

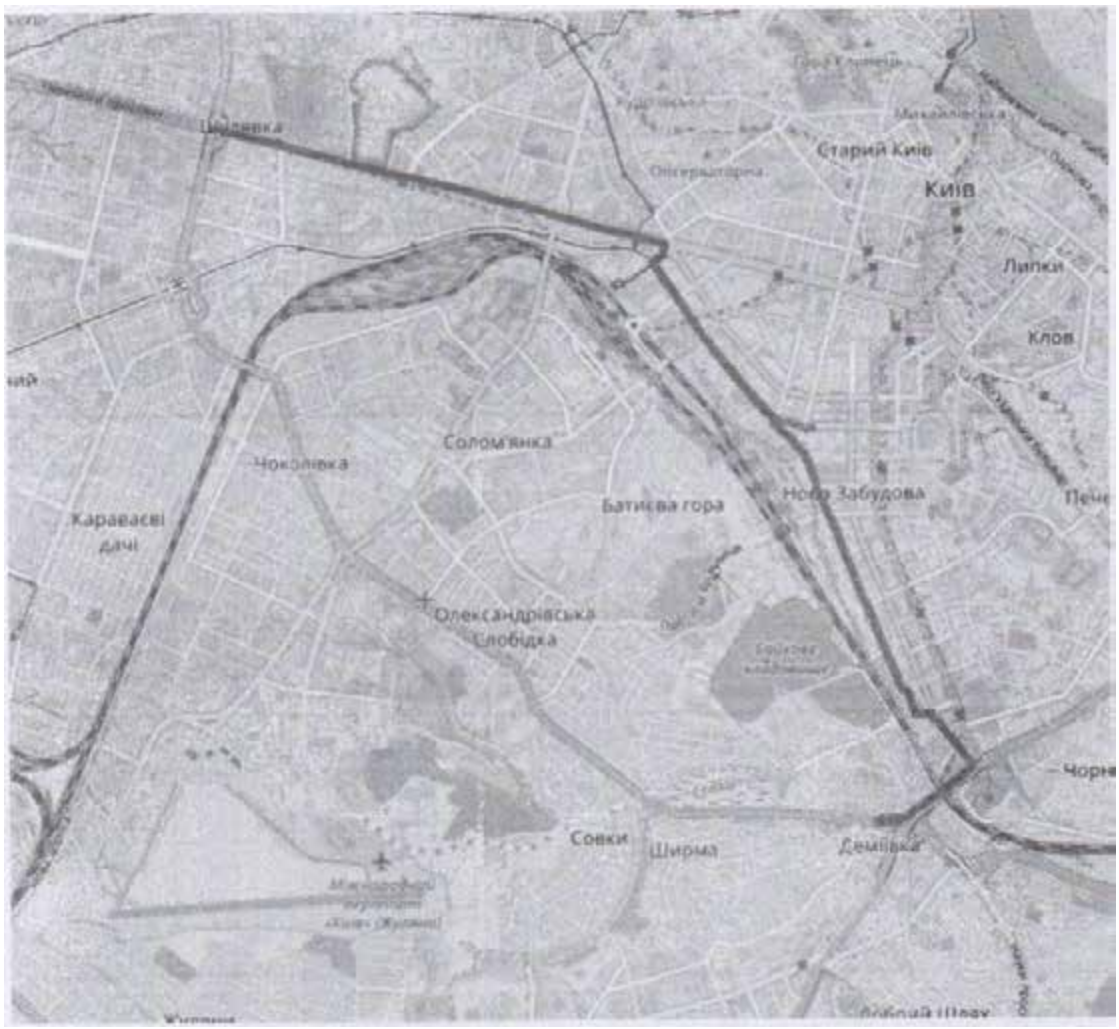

Рисунок 4.14 - Маршрут №4 - A*

Час пошуку - 2,37 с.

Дистанція - 9,4 км.

A* двонаправлений - рис. 4.15:

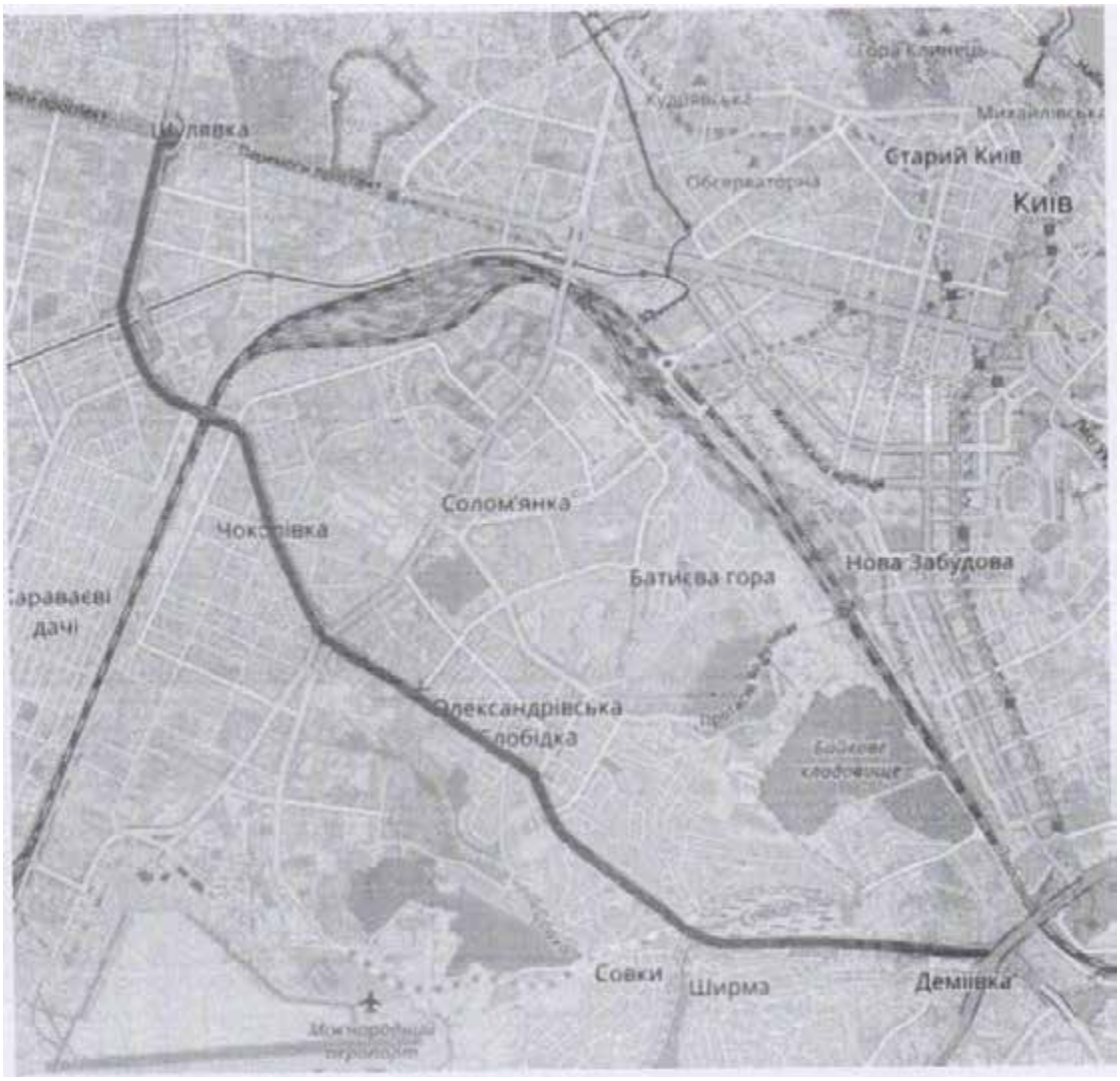

Рисунок 4.12 - Маршрут №4 - A* двонаправлений

Час пошуку - 1,17 с.

Дистанція 8,78 км.

Результат Google maps - рис. 4.16.

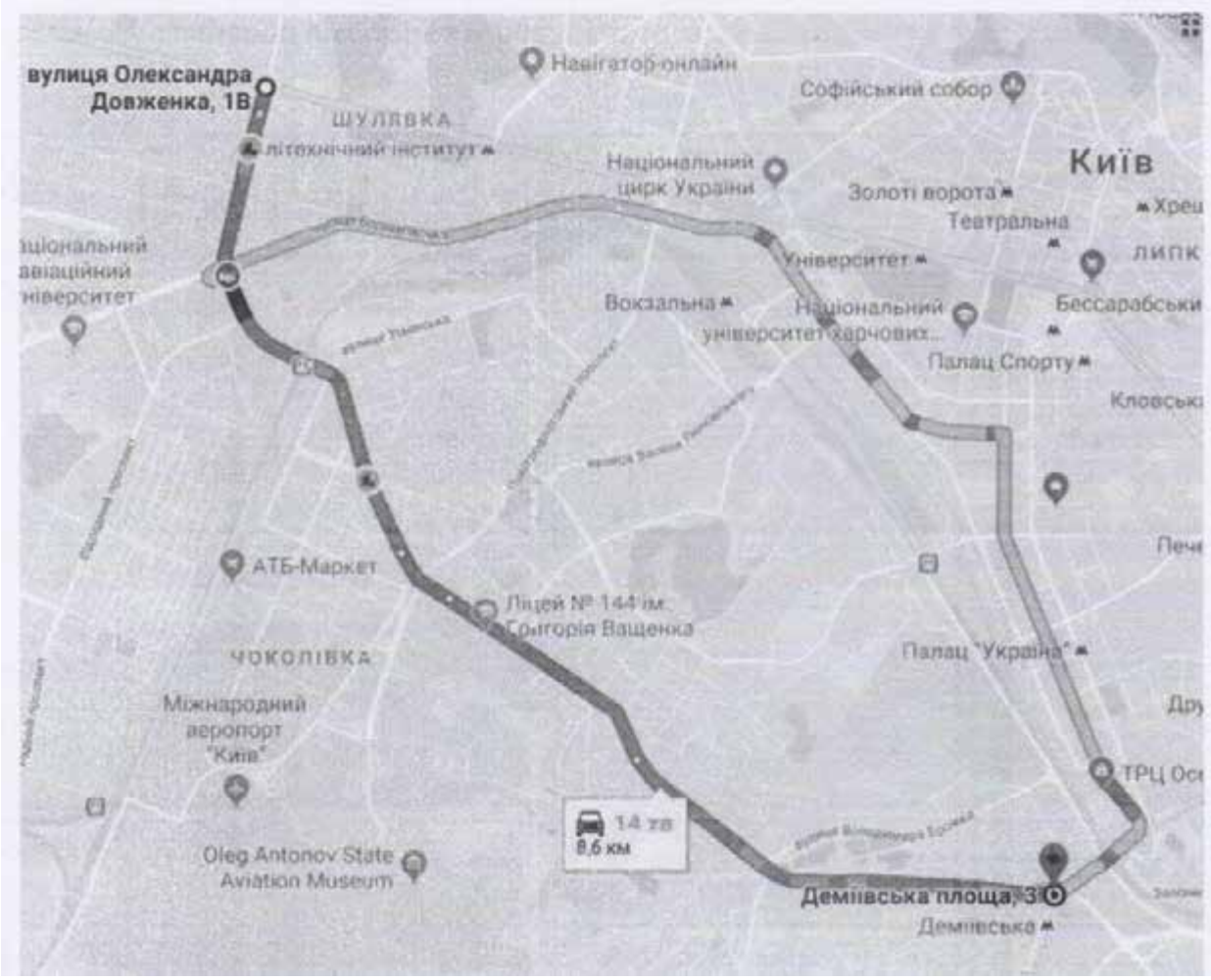

Рисунок 4.16 - Маршрут №4 - Google maps

Довжина - 8.6 км.

Алгоритм A* двосторонній виявився найефективнішим, знайшовши оптимальний маршрут за найменший час. Contraction Hierarchies також знайшов цей маршрут, але трохи повільніше. Алгоритм A* проклав більш довгий маршрут за більший час.

Результати Contraction Hierarchies та A* практично співпадають з Google Maps. Різницю можна пояснити помилковим вибором початку і кінця маршруту.

Маршрут №5. Маршрут від Майдану Незалежності до Виставкового центру. Довжина - 9,96 км.

Contraction hierarchies - рис 4.17:

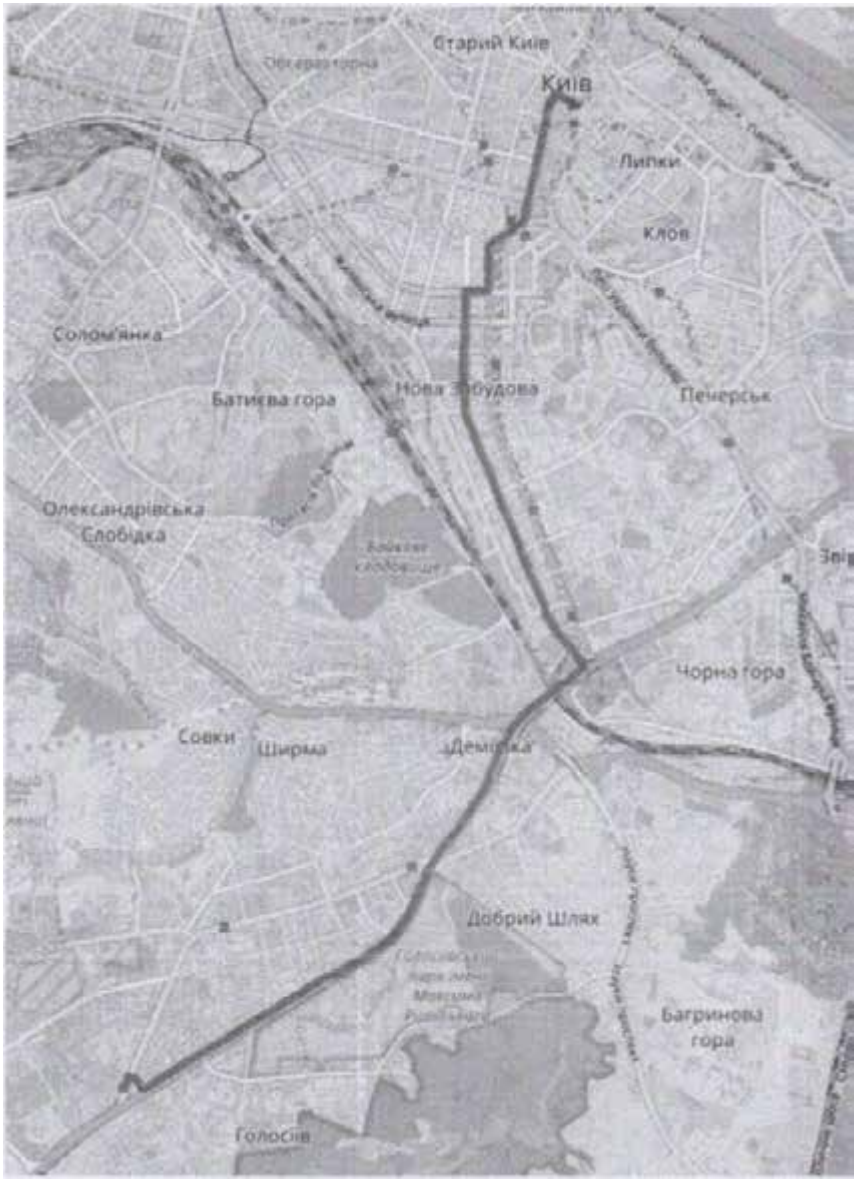

Рисунок 4.17 - Маршрут №5 - Contraction hierarchies

Час пошуку - 2,62 с.

Дистанція - 9,96 км.

A* - рис. 4.18:

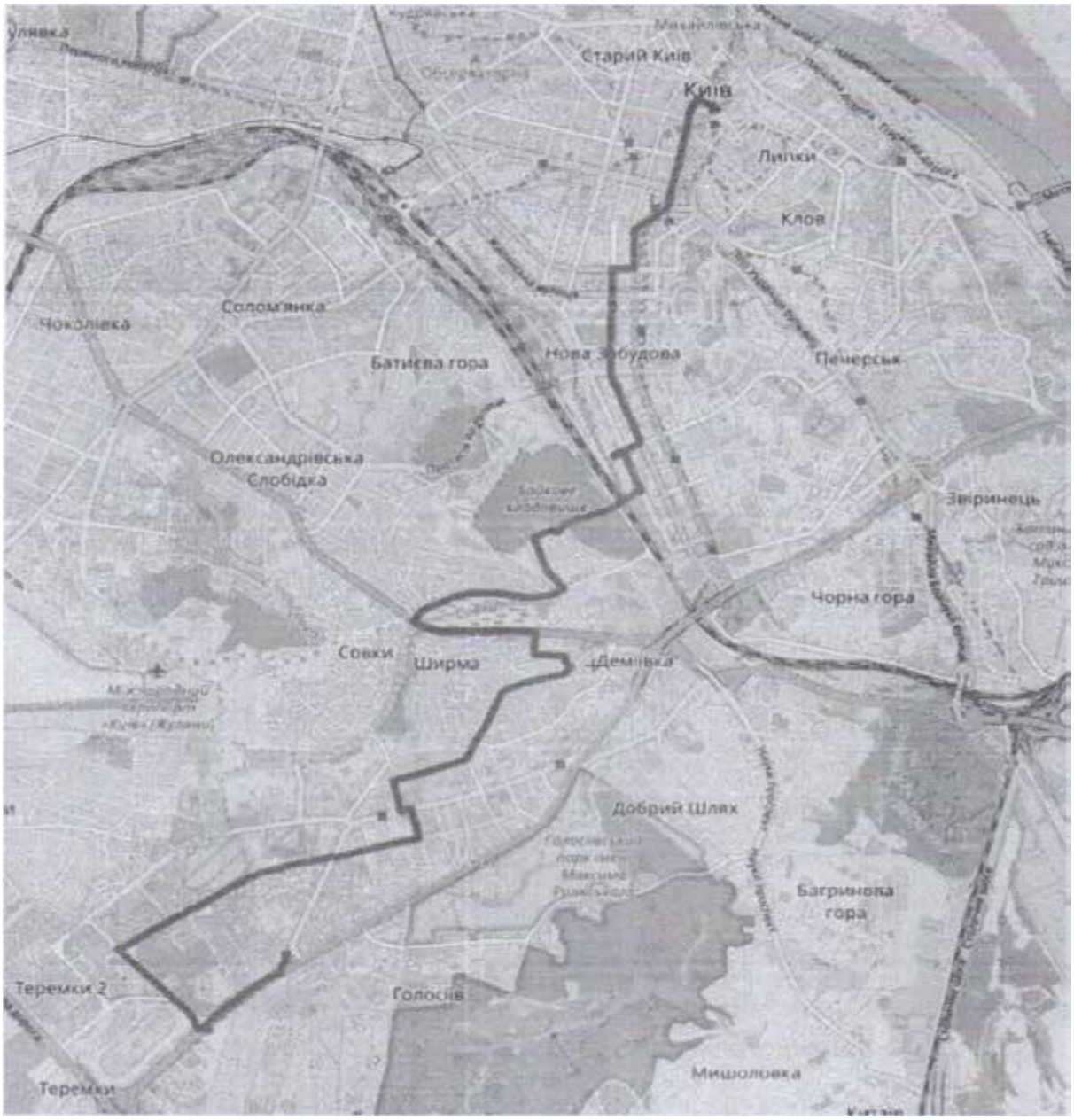

Рисунок 4.18- Маршрут №5 - A*

Час пошуку - 11,78 с.

Дистанція - 15,62 км.

A* двонаправлений - рис. 4.19:

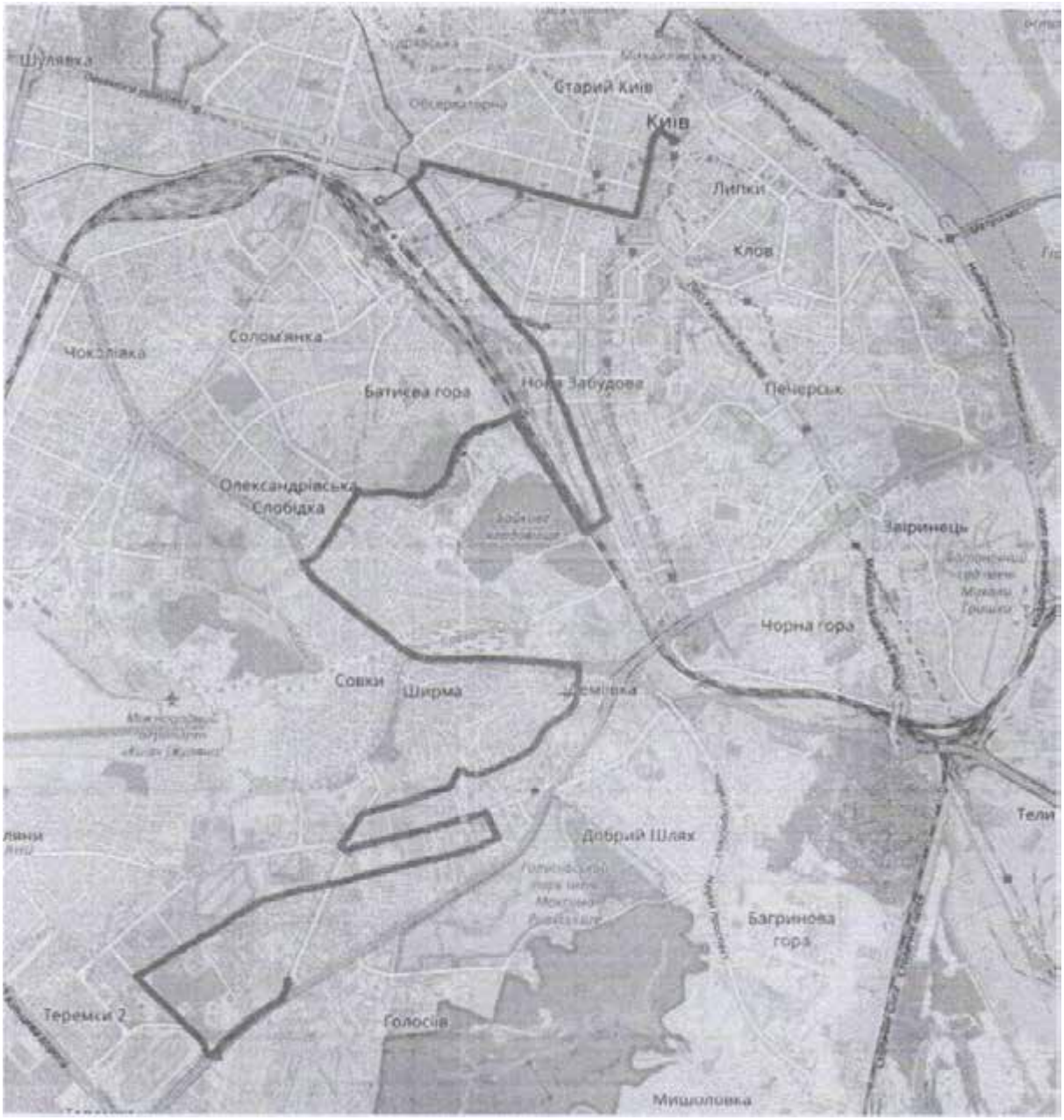

Рисунок 4.19 - Маршрут №5 - A* двонаправлений

Час пошуку - 82,82 с.

Дистанція 23,07 км.

Результати Google maps - рис. 4.20:

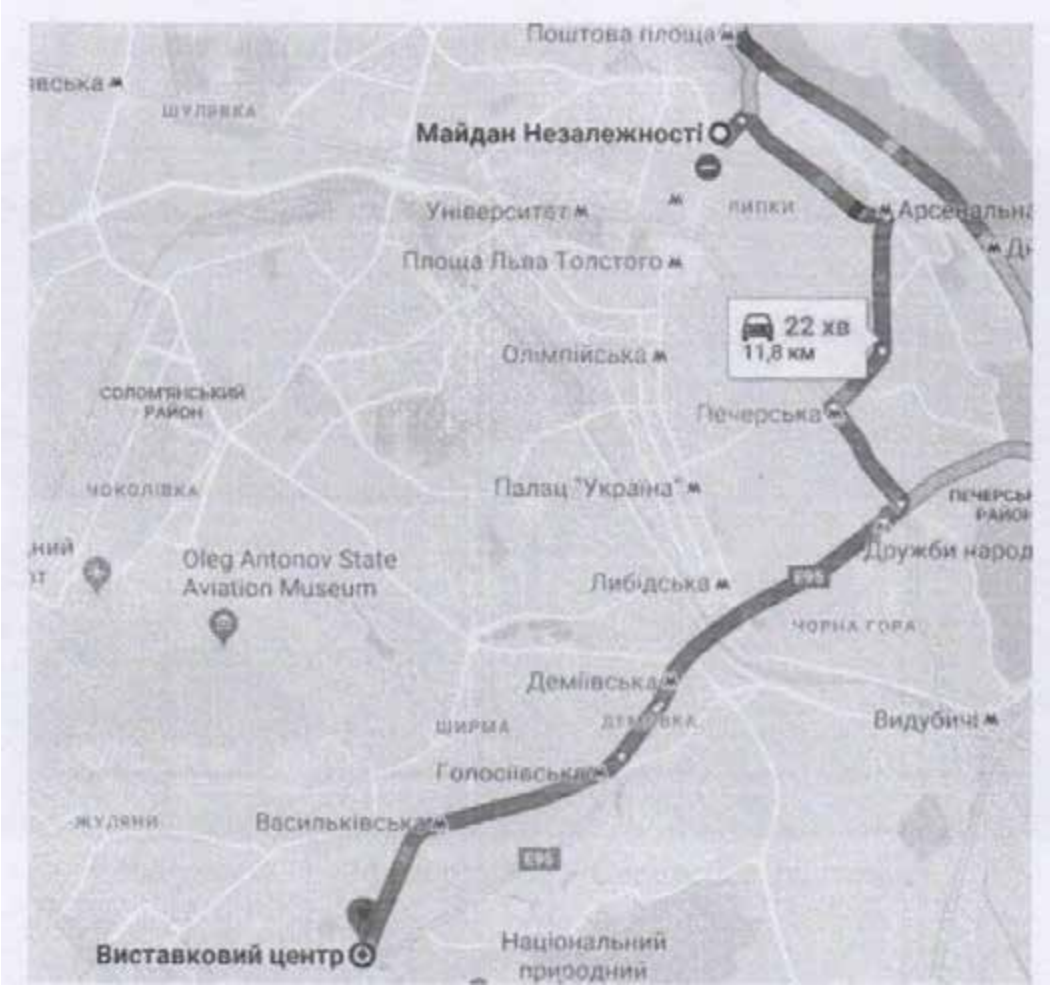

Рисунок 4.20 - Маршрут №5 - Google maps

Дистанція - 11,8 км.

На цьому складному маршруті, який починається і закінчується не на основних дорогах, алгоритми A* показують дуже погані результати, тоді як Contraction Hierarchies (CH) показує стабільний час і дуже добру відстань.

За результатами порівняння з Google Maps, CH навіть показав кращий результат, майже на 2 км коротший маршрут.Маршрут №6. Маршрут від станції метро Берестейська до станції метро Позняки. Довжина 18,58 км.

Contraction hierarchies - рис 4.21;

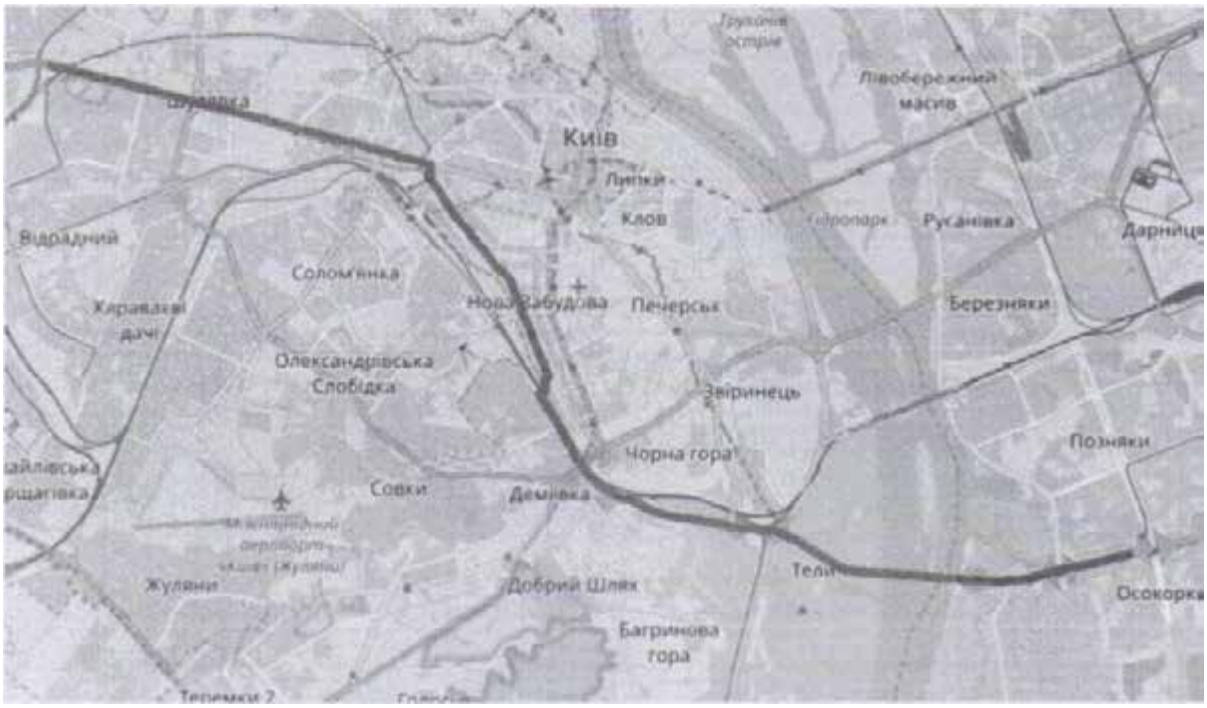

Рисунок 4.21 - Маршрут №6 - Contraction hierarchies

Час пошуку - 2,33 с.

Дистанція - 18,58 км.

A* - рис. 4.22:

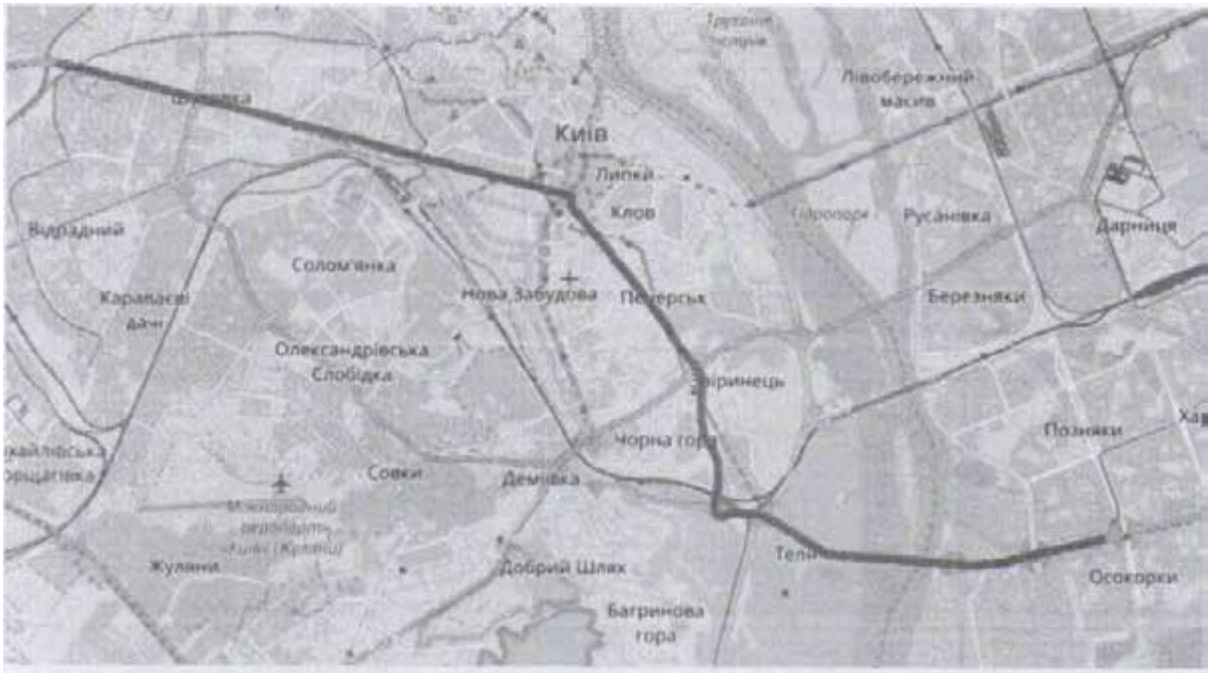

Рисунок 4.22 - Маршрут №6 – A*

Час пошуку - 3,36 с.

Дистанція - 18,71 км.

A* двонаправлений - рис. 4.23:

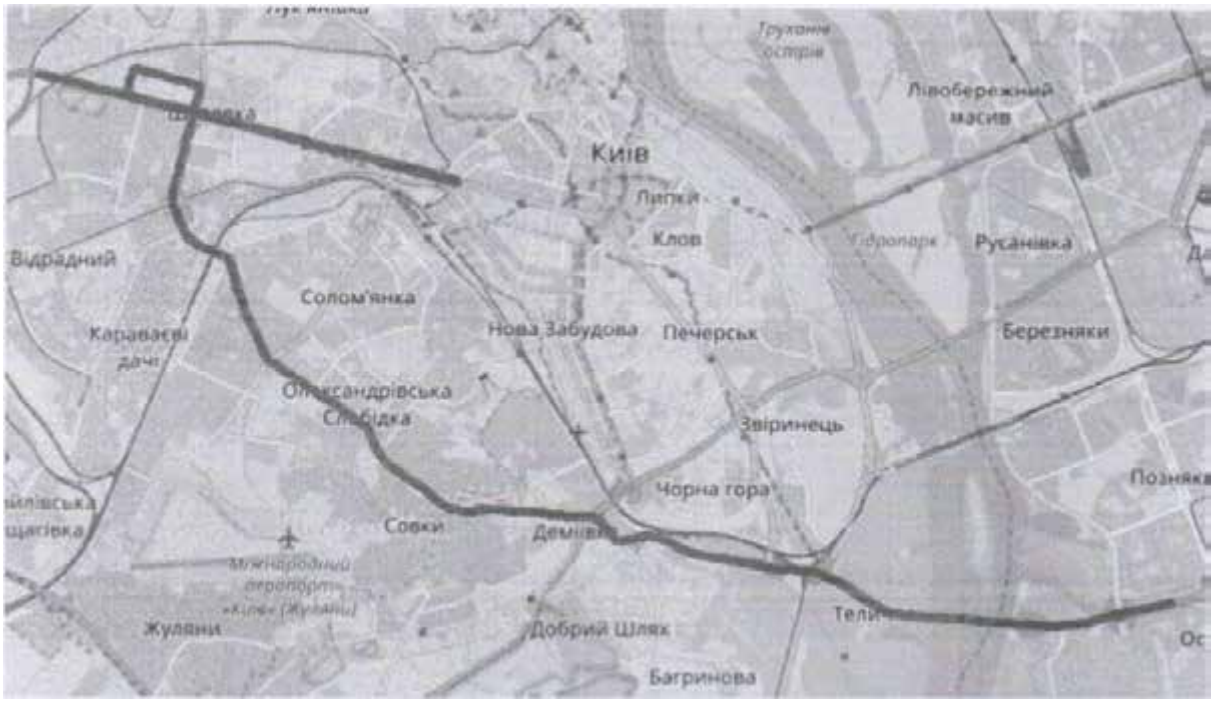

Рисунок 4.23 - Маршрут №6 - A* двонаправлений

Час пошуку - 36,89 с.

Дистанція - 28,48 км.

Результати Google maps - рис. 4.24:

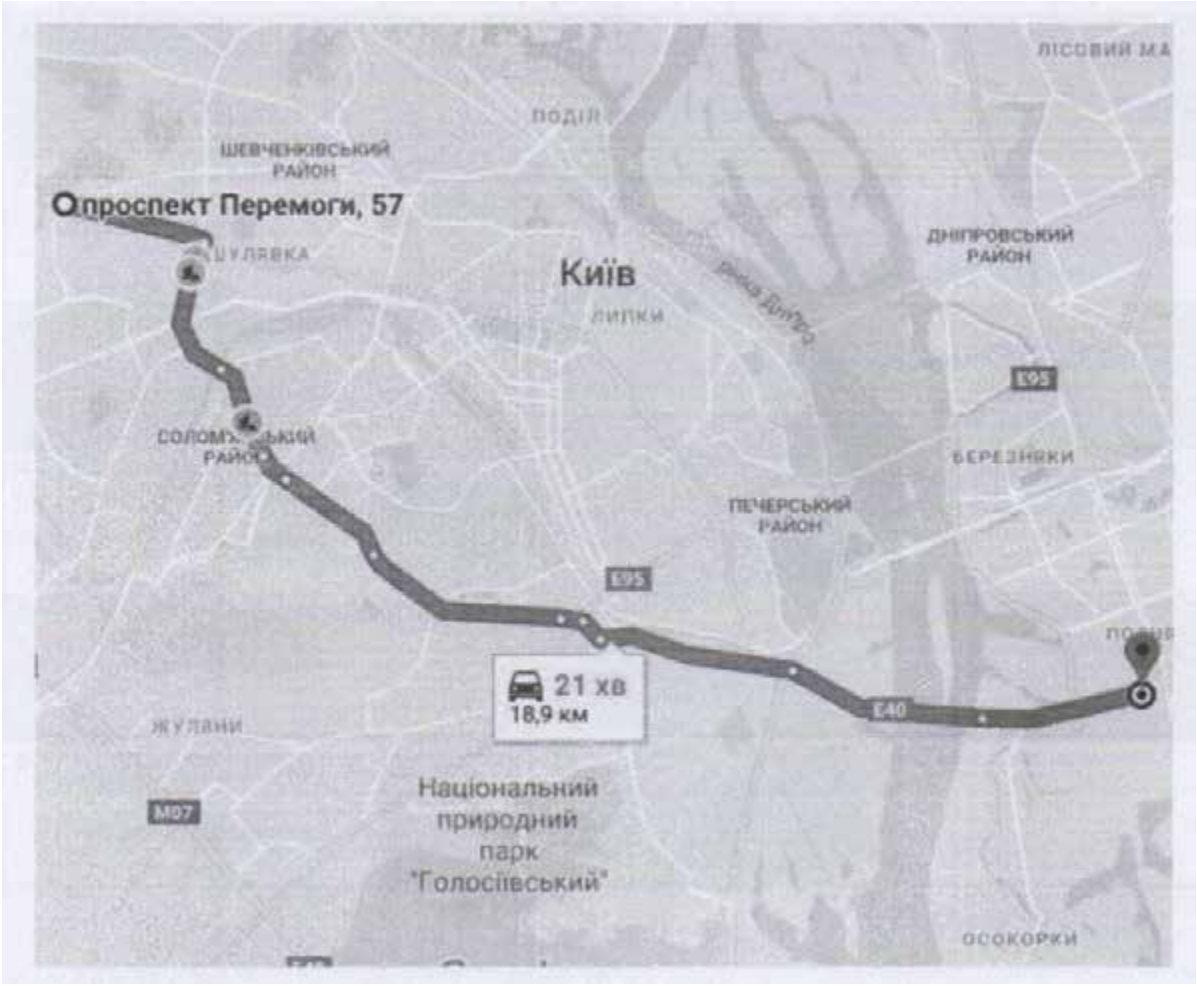

Рисунок 4.24 - Маршрут №6 -- Google maps

Дистанція - 20,4 км.

На найбільшому з аналізованих маршрутів алгоритм Contraction hierarchies демонструє стабільні і вражаючі результати з відстанню, навіть меншою, ніж у Google Maps, і швидким часом пошуку. Алгоритм A* також показує хороші результати, демонструючи прийнятний час і розумний маршрут.

Для зручності аналізу наведемо порівняльну таблицю результатів - таблиця 4.1.

Таблиця 4.1

Результати обробки маршрутів

№	Алгоритм	Час пошуку, с	Довжина, км	Кращий результат
1	Contraction hierarchies	1.23	2.19	A*
	A*	0.25	2.19	
	A* двонаправлений	0.31	2.19	
	Google maps	-	2.19	
2	Contraction hierarchies	1.14	3.33	CH
	A*	0.98	4.12	
	A* двонаправлений	1.04	4.48	
	Google maps	-	2.8	
3	Contraction hierarchies	0.96	2.23	CH
	A*	6.18	5.51	
	A* двонаправлений	16.52	4.84	
	Google maps	-	1.9	
4	Contraction hierarchies	1.48	8.78	A* двонаправлений
	A*	2.37	9.4	
	A* двонаправлений	1.17	8.78	
	Google maps	-	8.6	
5	Contraction hierarchies	2.62	9.96	CH
	A*	11.78	15.62	
	A* двонаправлений	82.82	23.07	
	Google maps	-	11.8	
6	Contraction hierarchies	2.33	18.58	CH
	A*	3.36	18.71	
	A* двонаправлений	36.89	28.48	
	Google maps	-	20.4	

Висновки за розділом 4

У даному розділі було проаналізовано результати обробки бази даних та функціонування реалізованих алгоритмів пошуку.

Створення графу доріг для Києва зайняло 33 хвилини, що можна вважати прийнятним, оскільки ця операція виконується одноразово.

Робота алгоритму Contraction hierarchies виявилася надзвичайно складною і тривалою, зайнявши понад 35 годин для неповної карти Києва. Незважаючи на те, що параметри обробки були спрямовані на швидкість, а не на якість отриманого графу, результати виявилися достатньо задовільними для

ефективної роботи пошукового алгоритму. Оптимізація технічної реалізації алгоритму має велике значення, існують можливості для його подальшого прискорення, включаючи оптимізацію коду та максимальне використання обчислювальних можливостей машини.

У розділі також було розглянуто роботу пошукових алгоритмів: A*, двонаправленого A* та Contraction hierarchies. У всіх шести випробуваннях A* і двонаправлений A* показали кращі результати, тоді як Contraction hierarchies був ефективнішим у чотирьох випадках. Важливо зазначити, що вибірка результатів не є достатньою для остаточних висновків, проте вже зараз можна зробити деякі припущення.

Зокрема, виявлено, що випадки, коли Contraction hierarchies показав гірші результати, різниця була незначною, і його результати були майже такими ж, як у кращого алгоритму. Також було встановлено, що час виконання алгоритмів A* та двонаправленого A* зростає зі збільшенням довжини маршруту, тоді як для Contraction hierarchies цей час практично залишається сталим, оскільки усі можливі вершини обробляються для вершин початку і кінця. Це надає значну перевагу останньому алгоритму.

Варто також відмітити, що швидкість пошуку маршруту залежить від якості обробленої бази даних, і в даному випадку якість обробки була недостатньою. Інвестування додаткового часу у вдосконалення цього процесу може призвести до ще більш ефективної роботи алгоритму.

5.РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проекту (товару, послуги, технології)

Опис ідеї стартап-проекту представлено в таблиці 5.1.

Таблиця 5.1- Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
В подорожах туристам часто бракує мобільного зв'язку та інтернету до якого всі звикли. Однією з головних незручностей є складність комунікації на відстані. У випадку коли одна людина з групи загубилася в незнайомому місті і може стати в нагоді даний програмний продукт. Суть його полягає у відображенні на карті мобільного пристрою групи людей у реальному часі.	1. Навігація	Швидкий пошук маршруту
	2. Орієнтування в незнайомому місті	Можливість знайти групу туристів в незнайомому місті
	3. Пошук друзів на карті	

Визначення сильних, слабких та нейтральних характеристик ідеї проекту представлено в таблиці 5.2.

Таблиця 5.2 - Сильні, слабкі та нейтральні характеристики ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(Потенційні) товари/концепції конкурентів			(слабка сторона)	(нейтр. стор.)	(сильна сторона)
		ТеатМар					
	Навігація						
	Можливість бачити інших користувачів на карті						
	Відображення ситуації на дорогах		т	—			
	Відомість бренду						
	Офлайн навігація						

5.2 Технологічний аудит ідеї проекту

Технологічна здійсненність ідеї проекту - таблиця 5.3.

Таблиця 5.3 - Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
	Навігація		Наявні. Необхідна розробка з використанням	Доступні
	Відображення кількох користувачів			Доступні
	Офлайн навігація			Доступні
Обрана технологія реалізації ідеї проекту: Проект буде реалізований на мові програмування C# з використанням технології мобільної розробки відповідних бібліотек для його і використання.				

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Попередня характеристика потенційного ринку стартап-проекту представлена в таблиці 5.4.

№ п/п	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	100 000(1000 од./
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі, %	20%

Таблиця 5.4 - Попередня характеристика потенційного ринку

Характеристика потенційних клієнтів стартап-проекту представлена в таблиці 5.5.

Таблиця 5.5 - Характеристика потенційних клієнтів стартап-проекту

Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп КЛІЄНТІВ	Вимоги споживачів до товару
Побудова маршрутів для групи людей.	Туристи, жителі великих міст	Туристами можуть бути люди будь-якого віку. Необхідно враховувати цей фактор при проектуванні інтерфейсу, щоб програма була зрозумілою для всіх. Загалом поведінка користувачів у контексті використання програми практично не буде відрізнятися.	Інтерфейс; Швидкодія; Можливість працювати без інтернет з'єднання.

Фактори загроз представлені в таблиці 5.6.

Таблиця 5.6 - Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
	Подібний функціонал з'явиться у більш відомих конкурентів	Люди будуть схильні користуватися вже відомою програмою, а не пробувати щось нове.	Активніша маркетингова кампанія

Фактори можливостей представлені в таблиці 5.7.

Таблиця 5.7 - Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
	Переклад на додаткові мови	Збільшення можливої кількості споживачів	
	Реалізація додатку на інших мобільних платформах	Збільшення можливої кількості споживачів	

Ступеневий аналіз конкуренції на ринку представлений в таблиці 5.8.

Таблиця 5.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	Олігополія. Є невелика кількість популярних програм з подібним функціоналом.	Необхідно, щоб у нас був особливий функціонал, який зможе зацікавити споживачів.
2.За рівнем конкурентної боротьби - локальний/національний/...	Глобальний. Люди по всьому світу можуть користуватися будь-якими мобільними додатками.	Позитивно вплинути можна лише перекладом додатка на різні мови. Але на початкових стадіях це не передбачається
3. За галузевою ознакою - міжгалузева/ внутрішньогалузева	Внутрішньогалузева.	
4. Конкуренція за видами товарів: оварно-родова оварно-видова - між бажаннями	Між бажаннями.	Необхідно випереджувати конкурентів у реалізації бажань клієнтів. Постійно монітори™ відгуки користувачів та робити оновлення функціоналу.
5. За характером конкурентних переваг - цінова / нецінова	Нецінова. Перевагою в кожного Виробника є особ-ливості функціоналу.	-
6. За інтенсивністю - марочна/не марочна	Не марочна	Торгова марка слабко впливає на позицію на ринку

Аналіз конкуренції в галузі за М. Портером - таблиці 5.9.

Таблиця 5.9 - Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
		Інформація відсутня			
Висновки:	Інтенсивність боротьби з існуючими конкурентами достатньо висока. Але ніяких перешкод виходу на ринок немає	Виграти конкуренцію можна тільки за рахунок переваг у функціоналі	-	-	

Обґрунтування факторів конкурентоспроможності наведено в таблиці 5.10.

Таблиця 5.10 - Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
	Інновації	Програмний продукт матиме функціонал, якого на даний момент немає в жодного з конкурентів
	Цінова політика	На початкових етапах, а, можливо, і надалі функціонал буде безкоштовним.
	Підтримка і оновлення	Додаток буде постійно оновлюватись для додавання нового функціоналу та згідно з побажаннями.

Порівняльний аналіз сильних та слабких сторін наведено в таблиці 5.11.

Таблиця 5.11 - Порівняльний аналіз сильних та слабких сторін

№ п/п	Фактор конкурентоспроможності	Бали 1-	Рейтинг товарів-конкурентів у порівнянні з Google						
	Інновації								
	Цінова політика								
	Підтримка і оновлення								

-аналіз стартап-проекту - таблиця 5.12.

Таблиця 5.12 - SWOT-аналіз стартап-проекту

Сильні сторони: новий функціонал, цінова політика	Слабкі сторони: мала відомість
Можливості: постійне розширення функціоналу	Загрози: повільна робота продукту, поява схожого функціоналу на ринку

Альтернативи ринкового впровадження стартап-проекту розглянуто в таблиці 5.13.

Таблиця 5.13 - Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
	Переклад на додаткові мови	Низька	3-6 місяці
	Реалізація додатка на різних мобільних платформах	Висока	6-12 місяців

5.4 Розроблення ринкової стратегії проекту

Вибір цільових груп потенційних споживачів наведено в таблиці 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 в сегмент
	Жителі міст	Для даної цільової групи продукт буде схожим на продукти конкурентів, тому перехід на використання нашого додатку ускладнений.	Попит низький.	Висока інтенсивність конкуренції.	Вхід в сегмент повільний і достатньо складний.
	Молоді туристи	Для цієї групи наш додаток матиме необхідний функціонал відмінний від конкурентів. До того ж молодь легко сприймає щось нове. Тому цей сегмент готовий до використання продукту	Високий попит завдяки особливостям функціонал у та його унікальні.	Інтенсивність конкуренції помірна.	Вхід в сегмент помірно-складний
	Люди старшого віку.	Дана група специфічна тим, що старші люди більш консервативні і, в контексті туризму, рідше подорожують великими групами, тому зникає необхідність використання особливого функціоналу, а консерватизм ще більше	Низький попит	Інтенсивність конкуренції помірна	Вхід в сегмент складний
	Мисливці, рибалки, грибники	Для людей цього сегменту програма буде допомагати не розгубитися. Ситуація схожа з туристичними групами.	Високий попит, проте група специфічна і кількість людей в групі невелика.	Конкуренція низька.	Вхід в сегмент простий.
Які цільові групи обрано: 1 - наймасовіша група, 2, 4					

Визначення базової стратегії розвитку наведено в таблиці 5.15.

Таблиця 5.15 - Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкуренто-спроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
	Розроблення нових функціональних можливостей	За рахунок функціоналу, який не можуть надати конкуренти	Групова навігація	

Визначення базової стратегії конкурентної поведінки наведено в таблиці

Таблиця 5.I6 - Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» 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
	Ні. Але є унікальний функціонал.	Обидва варіанти.	Буде. Базова навігація теж буде доступна, як і в продуктах конкурентів.	

Визначення стратегії позиціонування наведено в таблиці 5.17.

Таблиця 5.17 - Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту(три ключових)
	Навігація, робота офлайн, навігація для групи людей	Реалізація поставлених цілей. Внесення змін згідно побажань користувачів, отримуючи фідбек	Робота офлайн, навігація для групи осіб	Навігатор, туризм карти

5.5 Розроблення маркетингової програми стартап-проекту

Визначення ключових переваг концепції потенційного товару - таблиці

Таблиця 5.18 - Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
	Навігація	Забезпечує потребу	Переваг в даному контексті немає.
	Робота офлайн	Забезпечує потребу	Перевага перед тими конкурентами, які не мають такого функціоналу
	Навігація для групи людей	Забезпечує потребу	Перевага перед усіма конкурентами

Опис трьох рівнів моделі товару приведено в таблиці 5.19.

Таблиця 5.19 - Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Навігація, навігація для групи людей, робота офлайн, поради місць для відвідування		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	видкодія ористувацький інтерфейс ідмовостійкість ідтримка	Нм Нм Нм Нм	Тх/Тл/Е Е Тх/Тл Тх/Е
	Якість: стандарти відсутні. Проект покрито автоматизованими тестами.		
	Пакування: Play market, Apple store		
	Марка: TeaMMaP		
III. Товар із підкріпленням	До продажу: Додаток який надає функціонал навігатора з особливими функціями, такими як: навігація для групи людей та рекомендації місць для відвідування.		
	Після продажу: Швидкодія, зручний користувацький інтерфейс.		
Продукт буде з закритим вихідним кодом. Тобто готовий функціонал скопіювати не буде можливості, а захистити ідеї не вдасться.			

Визначення меж встановлення ціни - 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
				Продукт безкоштовний. Дохід отримується з реклами.

Формування системи збуту наведено в таблиці 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
	Клієнт завантажує безкоштовний додаток з			

Концепція маркетингових комунікацій приведена в таблиці 5.22.

Таблиця 5.22 - Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	— Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
	група-жителі міст	Реклама в соц. мережах, додатках пов’язаних з погодою	Карти офлайн.	Розповісти про існування додатку.	Навігація офлайн
	2 група - молоді туристи. Молодь користуються великою кількістю інших мобільних додатків.	Рекламу можна додавати в інших додатках пов’язаних з туризмом, соц. мережах.	Карти офлайн, навігація для групи	Розповісти про існування додатку.	Навігатор для туристів; Навігація офлайн.
	3 група - мисливці, рибалки, грибники	Реклама на форумах, соц мережах.	Навігація для групи людей. Карти офлайн.	Розповісти про існування додатку.	Навігація офлайн.

Висновки за розділом 5

Загальний аналіз проекту дозволяє зробити висновок, що його комерційний успіх є досить ймовірним, оскільки виявлено значний потенціал серед різних груп клієнтів. Незважаючи на наявність конкурентів на ринку, продукт може привернути увагу завдяки своїм особливостям. Підкреслюється необхідність подальшої імплементації проекту, а також важливість вибору стратегії, спрямованої на додавання унікального функціоналу для привернення більшої кількості клієнтів і популяризації продукту.

ВИСНОВКИ ПО РОБОТІ

У цій роботі детально розглянуто різні підходи до реалізації багатоагентних систем, зосереджуючись зокрема на створенні системи маршрутизації з використанням алгоритмів пошуку маршрутів у вагованих графах.

Розроблена система складається з двох ключових агентів: агента обробки бази даних та агента маршрутизації, яка може бути додатково розширена іншими агентами для забезпечення більш широкого функціоналу програмного продукту.

Агент обробки бази даних відповідає за створення та обробку графу доріг на основі картографічних даних, отриманих зі служби OpenStreetMap.

Агент маршрутизації включає в себе реалізацію таких алгоритмів пошуку маршрутів, як A*, A* двонаправлений та Contraction hierarchies.

Було проведено комплексне тестування розробленої системи і здійснено порівняння отриманих результатів з результатами, отриманими від сервісу Google Maps. Найкращим показником виявився алгоритм Contraction hierarchies, який продемонстрував стабільні результати як за часом, так і за якістю побудованих маршрутів як на коротких, так і на довгих відстанях. Маршрути, побудовані з його допомогою, були близькими до тих, що отримані від Google Maps, і в деяких випадках навіть були кращими.

Пропонований стартап-проект передбачає подальший розвиток продукту. На даному етапі було створено основу системи. Подальший розвиток залежатиме від наявності необхідних ресурсів згідно зі стратегією, обраною для стартап-проекту.

ПЕРЕЛІК ПОСИЛАНЬ

ерж К. Теория графов и ее приложения / К. Берж. — М.: ИЛ, 1962. — 320 с.

лексеев В.Е. Нахождения кратчайших путей в графе. / Алексеев В.Е., Таланов В.А. // В со.: Графы. Модели вычислений. Структуры данных. — Нижний Новгород: Издательство Нижегородского гос. университета, - 2005. —с. 236-237.

атт У. Теория графов *i* У. Татт — М.: Мир, 1988. — 424 с.

вами М. Графы, сети и алгоритмы / Свами М., Тхуласираман К. — М: Мир, 1984. — 455 с.

abeling Algorithm for Shortest Paths on Road Networks. / [Abraham I., Celling D., Goldberg A., Wemeck R.]. - Philadelphia. - Symposium on Experimental Algorithms, 2011. — pp. 230-241.

In.: Numerische Mathematik. - 1959. - V. 1.— pp. 269-271.

Ford L., Fulkerson D. — Princeton: Princeton University Press, 1962. — 253 p.

A

[

Graxx - Алгоритм Флойда-Уоршелла [Електронний ресурс]. - Режим доступу

o

ваний В. Алгоритмы: введение в разработку и анализ. / Ананий В., Левитин А. // В сб.: Introduction to The Design and Analysis of Aigorithms. — М.: «Вильямс», 2006. — с. 212-215.

E. Moore. - In.: Proceedings of the International Symposium on the Theory of Switching. — Harvard University Press, 1959. — pp. 285-292.

[

Агенты и мультиагентные системы [Електронний ресурс]. - Режим доступу:http://ai-

Мультиагентные системы [Електронний ресурс]. — Режим доступу:

M. Wooldridge — Liverpool: John Wiley & Sons, 2002. — p. 366

Построение маршрута с помощью алгоритма Дейкстры [Електронний ресурс]. -

Р

Є

[ж

[и

[м

[п

д

ф

р

р

у

н

и

й

й

р

р

р

у

у

р

р

Р

е

Р

ж

ж

м

м

д

о

о

[Електронний ресурс]. - Режим доступу: Coursera - Contraction hierarchies — Node ordering [Електронний ресурс]. - Режим доступу: