МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2024 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ МАРШРУТІВ ЕЛЕКТРОТРАНСПОРТУ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21810209

Виконав студент 4-го курсу, групи 402
_____В. В. Дмитрук
«19» червня 2024 р.

Керівник: д-р техн. наук, доцент
_____О. В. Козлов
«19» червня 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних
інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____2024 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Дмитруку Владиславу Вячеславовичу.

1. Тема кваліфікаційної роботи «Застосунок для планування маршрутів електротранспорту на основі інтелектуальних технологій».

Керівник роботи Козлов Олексій Валерійович, д-р техн. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «28» грудня 2023 р. № 271

2. Строк представлення кваліфікаційної роботи студентом «19» червня 2024 р.

3. Вхідні (початкові) дані до роботи: Data-set з координатами електротранспорту.

Очікуваний результат: застосунок для планування маршрутів на основі мурашиного алгоритму.

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

- аналіз існуючих методів пошуку маршрутів;
- мурашиний алгоритм в системі побудові маршрутів;
- опис архітектури та реалізація системи.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини:

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Алексєєва А.О., доцент кафедри екології	

Керівник роботи __д-р техн. наук, доцент Козлов О. В._____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Дмитрук В. В._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_14_» _____січня____ 2024 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: створення застосунку для планування маршрутів електротранспорту на основі інтелектуальних технологій.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників КРБ	10.11.2023	15.11.2023	Виконано
2	Отримання завдання на виконання КРБ	10.01.2024	15.01.2024	Виконано
3	Складання календарного плану роботи на весь період виконання КРБ	16.01.2024	30.01.2024	Виконано
4	Отримання завдання на переддипломну практику	15.04.2024	29.04.2024	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до КРБ	29.04.2024	11.05.2024	Виконано
6	Розробка звіту з переддипломної практики	12.05.2024	15.05.2024	Виконано
7	Виконання КРБ: аналіз існуючих методів пошуку маршрутів, мурашиний алгоритм в системі побудови маршрутів, опис архітектури та реалізації системи	13.05.2024	22.06.2024	Виконано
8	Перший попередній захист КРБ на засіданні комісії кафедри	27.05.2024	27.05.2024	Виконано
9	Доробка та остаточне оформлення КРБ	28.05.2024	09.06.2024	Виконано
10	Другий попередній захист КРБ на засіданні комісії кафедри	10.06.2024	10.06.2024	Виконано
11	Подання КРБ рецензенту	13.06.2024	13.06.2024	Виконано
12	Подання КРБ, її електронної копії та інших документів (відгуку, рецензії) до захисту	17.06.2024	21.06.2024	Виконано
13	Захист БКР перед екзаменаційною комісією (ЕК)	24.06.2024	28.06.2024	

Розробив студент _Дмитрук В. В._____ _____
(прізвище, ім'я, по батькові студента) (*підпис*)

Керівник роботи __ д-р техн. наук, доцент Козлов О. В. _____
(*посада, прізвище, ім'я, по батькові)*
(підпис)

«_29_» _____01_____ 2024 р.

АНОТАЦІЯ

кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Дмитрука Владислава Вячеславовича

Тема: «Застосунок для планування маршрутів електротранспорту на основі інтелектуальних технологій»

Актуальність кваліфікаційної роботи визначається зростанням популярності та значущості електротранспорту для сталого розвитку міст. Оптимізація маршрутів електротранспорту стає ключовим завданням для забезпечення ефективного та екологічного транспорту. Ця робота спрямована на розвиток і застосування інтелектуальних технологій у сфері планування маршрутів електротранспорту з метою підвищення ефективності та зручності пересування користувачів.

Об'єктом кваліфікаційної роботи є процеси планування маршрутів електротранспорту.

Предметом кваліфікаційної роботи є методи, моделі та алгоритми оптимізації маршрутів електротранспорту.

Метою даної кваліфікаційної роботи є підвищення ефективності функціонування мереж електротранспорту шляхом розробки застосунку для планування маршрутів на основі інтелектуальних технологій.

В результаті виконання роботи було досліджено і порівняно два методи оптимізації, а саме класичний алгоритм мурашиної колонії і алгоритм A*, визначені основні їх переваги та недоліки, а також розроблено програмне забезпечення, в якому реалізовані відповідні методи.

Дана робота складається з чотирьох розділів. Кожен розділ відповідно присвячений: аналіз методів пошуку маршруту, мурашиний алгоритм в системі побудови маршрутів, опис архітектури та реалізації застосунку, охорона праці. Загальний обсяг роботи 84 сторінок. Кваліфікаційна робота бакалавра містить 1 додаток, 42 рисунка, 1 таблицю і посилання на 48 літературних джерела.

Ключові слова: оптимізація транспортних маршрутів, задача комівояжера, , TSP, ACO-метод, Python, Flesk, Google API, Javascript.

ABSTRACT

Bachelor's qualification work of the student of 402 group of Petro Mohyla Black Sea National University

Dmytruk Vlad

"The relevance of the qualification work to increase the popularity and importance of electric transport for the sustainable development of the city. Optimization of transport routes becomes a key task of electricity supply to ensure efficient and ecological transport. This work is aimed at the development and application of intelligent technologies in the field of electric transport route planning to increase the efficiency and convenience of user movement.

The object of the qualification work is the process of planning electric transport routes.

The subject of the qualification work is methods, models and algorithms for optimizing electric transport routes.

The method of this qualification work is to increase the efficiency of the operation of the electric transport network by developing an application for planning routes based on intelligent technologies.

To achieve this goal, it is necessary to perform the following tasks:

– analyze existing methods and means of finding optimal transport routes;

– to research and implement the algorithm of ant colonies for use in the software system of forming routes for transportation;

– select some tools and technologies to create a high-quality route planning application;

– to design and develop an application that will allow you to abandon the construction of transportation routes using an ant algorithm;

– investigate the quality and speed of the developed application.

As a result of the work, two optimization methods were investigated and provided, as well as the classic ant colony algorithm and the A* algorithm, their main advantages and disadvantages were determined, and software was developed in which the corresponding methods were implemented

This work consists of four sections. Each chapter is respectively dedicated to: analysis of route search methods, ant algorithm in the route construction system, descriptions and implementation of the application, labor protection.

The qualification work contains 84 pages, 42 figures, 1 tables and 1 appendix. The paper uses 48 sources.

Keywords: optimization of transport routes, traveling salesman problem, TSP, ACO-method, Python, Flesk, Google API, and Javascript.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ЕМА — Евристичний мурашиний алгоритм

МА — Мурашиний алгоритм

ПЗ — Програмне забезпечення

ACO — Ant Colony Optimization (Оптимізація на основі мурашиних колоній)

ACS — Ant Colony System (Система мурашиних колоній)

ACSDP — Ant Colony Systems with Dynamic Parameters (Мурашина система з динамічними параметрами)

ASCEA — Ant Colony Systems with Elite Ants (Мурашина система з елітними мурахами)

BFS — Breadth-First Search (Пошук в ширину)

DFS — Depth-First Search (Пошук в глибину)

GPS — Global Positioning System (Глобальна система позиціонування)

JSP — Job Shop Scheduling Problem (Задача розкладу робіт)

MAS — Multi-Agent System (Багатоагентна система)

MMAS — Max-Min Ant System (Максимально-мінімальна мурашина система)

TSP — Traveling Salesman Problem (Задача комівояжера)

VRP — Vehicle Routing Problem (Задача маршрутизації транспортних засобів)

ВСТУП

Темою кваліфікаційної роботи бакалавра є створення застосунку для планування маршрутів електротранспорту на основі інтелектуальних технологій.

Багато міст світу активно переходять на електротранспорт як частину своїх стратегій розвитку, і створення застосунку для останнього відповідає цим глобальним тенденціям та потребам. Експлуатаційні витрати на електротранспорт зазвичай нижчі, оскільки електроенергія дешевша за паливо, а також менше потреб у технічному обслуговуванні через меншу кількість рухомих частин.

Проблема пошуку найкоротших шляхів є важливою та широко відомою в різних сферах. Для її вирішення розроблено низку алгоритмів, які забезпечують ефективність та точність в побудові маршрутів. Успіх програмного забезпечення у вирішенні цієї проблеми значною мірою залежить від алгоритму, який воно використовує для оптимізації. Згідно з нещодавніми дослідженнями, деякі методи, які використовують методи штучного інтелекту, такі як мурашиний алгоритм, можуть допомогти знайти хороші рішення для проблеми комівояжера та не тільки.

За основу роботи був взятий мурашиний алгоритм, який має ряд переваг. Він базується на природній поведінці мурах, які використовують феромони для знаходження найкоротшого шляху до їжі. Цей алгоритм працює паралельно, розглядаючи багато можливих рішень одночасно, що збільшує шанси знайти оптимальне рішення. Останній адаптується до змін у середовищі, використовує евристичну інформацію для ефективнішого пошуку, працює навіть за відсутності повних даних і ефективно використовує пам’ять, зберігаючи лише найкращі рішення.

Актуальність кваліфікаційної роботи базується у контексті росту популярності та важливості електротранспорту для сталого розвитку міст. Оптимізація маршрутів електротранспорту стає ключовим завданням у забезпеченні ефективного та екологічно чистого транспорту. Дана робота спрямована на розвиток та застосування інтелектуальних технологій у сфері

планування маршрутів електротранспорту з метою підвищення ефективності та зручності пересування для користувачів.

Об’єктом кваліфікаційної роботи є процеси планування маршрутів електротранспорту.

Предметом кваліфікаційної роботи є методи, моделі та алгоритми для оптимізації маршрутів електротранспорту.

Метою даної КРБ є підвищення ефективності функціонування мереж електротранспорту за рахунок розробки застосунку для планування маршрутів на основі інтелектуальних технологій.

Для досягнення поставленої мети у роботі необхідно виконати низку завдань:

– проаналізувати існуючі методи та засоби для пошуку оптимальних транспортних маршрутів;

– дослідити та реалізувати алгоритм мурашиних колоній для використання в програмній системі формування маршрутів для перевезень;

– здійснити підбір необхідних засобів та технологій для створення високоякісного застосунку для планування маршрутів;

– здійснити проєктування та розробку застосунку, який дозволить виконувати побудову маршрутів перевезень за допомогою мурашиного алгоритму;

– дослідити якість та швидкодію розробленого застосунку.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПОШУКУ МАРШРУТІВ

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

1.1 Алгоритм A* та мурашиний алгоритм

A* - це алгоритм, який шукає найоптимальніший шлях від початкової вершини до кінцевої, зосереджуючись на мінімізації вартості, і використовує перший найкращий збіг у процесі пошуку (рис 1.1).

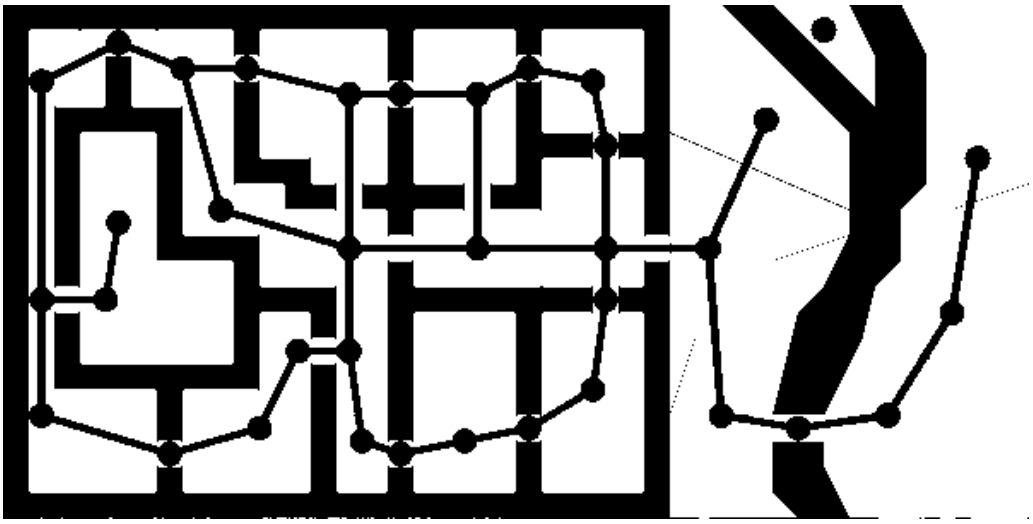

Рисунок 1.1 – Приклад роботи алгоритма A*[1]

Послідовність руху вузлів визначається за допомогою евристичної функції "відстань + вартість" (позначеної як $f(x)$). Ця функція є сумою двох інших: функції вартості досягнення розглянутої вершини (x) з початкової (зазвичай позначеної як $g(x)$ і може бути як евристичною, так і неевристичною) і евристичної оцінки відстані від розглянутої вершини до кінцевої (позначеної як $h(x)$). Функція $h(x)$

повинна бути припустимою евристичною оцінкою, тобто не переоцінювати відстані до цільової вершини. Наприклад, у задачі маршрутизації $h(x)$ може представляти собою відстань до мети в прямій лінії, оскільки це фізично найменша можлива відстань між двома точками.

Опис алгоритму. Алгоритм A* крок за кроком аналізує всі можливі шляхи від початкової вершини до кінцевої, шукаючи мінімальний. Схоже на інші інформовані алгоритми пошуку, він спочатку розглядає ті шляхи, які можливо найбільше підходять для досягнення мети. Відрізняючись від інших алгоритмів пошуку за першим найкращим збігом, A* при виборі вершини враховує весь пройдений до неї шлях.

На початку роботи алгоритм розглядає сусідні вузли з початковим, обираючи той, у якого значення $f(x)$ мінімальне, і розкриває його. На кожному кроці алгоритм опрацьовує багато шляхів від початкової точки до всіх ще не розкритих вершин графа, які розміщуються у черзі з пріоритетом. Пріоритет кожного шляху визначається як $f(x) = g(x) + h(x)$. Робота алгоритму триває до тих пір, поки значення $f(x)$ цільової вершини не стане меншим за будь-яке значення в черзі або поки весь граф не буде пройдений. Якщо існує кілька можливих шляхів, обирається той, що має найменшу вартість.

Якщо відомо, що рішення існує, або якщо алгоритм пошуку оптимального маршруту вміє відсікати цикли, можна пропустити велику кількість вже пройдених вершин, що перетворює алгоритм на пошук по дереву рішень.

Властивості. Алгоритм A* завжди забезпечує знаходження рішення у всіх випадках.

Умова допустимості евристичної функції h, коли вона ніколи не переоцінює дійсну мінімальну вартість досягнення мети, робить A* також допустимим (або оптимальним), якщо не відсікаються пройдені вершини. Щоб алгоритм був оптимальним, крім того, необхідно, щоб $h(x)$ була монотонною евристикою. Монотонність означає, що якщо є шляхи $A - B - C$ та $A - C$, то оцінка вартості

шляху від A до C повинна бути менше або дорівнювати сумі оцінок шляхів $A - B$ і $B - C$. Математично, це можна виразити як для всіх шляхів x, y:

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

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

Окремі випадки. В загальному випадку, пошук в глибину і пошук в ширину представляють собою два різні підходи, які можна розглядати як окремі варіанти алгоритму A*. У випадку пошуку в глибину використовується глобальна змінна - лічильник, який початково ініціалізується великим значенням. Кожного разу, коли розглядається вершина, суміжним вершинам присвоюється значення лічильника, що зменшується на одиницю після кожного операції присвоєння. Це означає, що вершини, які розглядаються раніше, отримають більше значення $h(x)$, і, отже, будуть розглянуті в останню чергу. Якщо встановити $h(x) = 0$ для всіх вершин, то отримаємо ще один особливий випадок - алгоритм Дейкстри.

Особливості реалізації. Існує кілька важливих аспектів реалізації та прийомів, які можуть суттєво вплинути на ефективність алгоритму. Перше, на що варто звернути увагу, - це спосіб обробки зв'язків між вершинами в черзі з пріоритетом. Якщо вершини додаються у порядку "перший прийшов - останній вийшов", то в разі однакової оцінки алгоритм A* віддасть перевагу руху в глибину. У випадку, коли вершини додаються у порядку "перший прийшов - перший вийшов", алгоритм буде виконувати пошук в ширину для вершин з однаковою оцінкою. Це може суттєво вплинути на продуктивність в багатьох сценаріях.

Після завершення роботи алгоритму і отримання маршруту зазвичай кожна вершина зберігає посилання на батьківський вузол. Це дозволяє легко відновити оптимальний маршрут. Важливо, щоб одна і та ж вершина не додавалася у чергу двічі. Зазвичай цю проблему вирішують перевіркою перед додаванням, чи вже є

вершина в черзі. Якщо так, то оновлюється запис так, щоб зберігалась мінімальна вартість. Багато стандартних алгоритмів для пошуку вершини в кроні дерева потребують часу $O(n)$. Використання хеш-таблиці може значно скоротити цей час.

Оцінка складності. Ефективність алгоритму A* залежить від якості використовуваної евристики. У найгіршому випадку, кількість вершин, які алгоритм вивчає, зростає експоненціально у порівнянні з довжиною найкоротшого шляху. Однак складність стає поліноміальною, якщо евристика задовольняє наступну умову:

$$|h(x) - h^*(x)| \leq O(log\, h^*(x)) \tag{1.2}$$

Де h^*- представляє собою оптимальну евристику, яка точно визначає відстань від вершини x до мети. Іншими словами, похибка $h(x)$ не зростає швидше, ніж логарифм від оптимальної евристики.

Цей алгоритм має кілька переваг, серед яких гарантоване знаходження кращого розв'язку у випадку його наявності та можливість розкриття найменшої кількості вершин, на відміну від інших алгоритмів пошуку.

Проте, недоліками цього методу є тимчасова складність, що була описана раніше, залежність від точності евристичного алгоритму та велике споживання алгоритмом ресурсів пам'яті. Зазвичай вимагається зберігання експоненціальної кількості вузлів у пам'яті. Саме через велике споживання ресурсів цей алгоритм може бути непридатним для застосування у деяких системах.

Перейдемо до мурашиного алгоритму (рис. 1.2). Основна ідея методу мурашиних колоній полягає в наслідуванні поведінки мурах для пошуку найкоротшого шляху від мурашника до джерела їжі та адаптації до змін у зовнішньому середовищі. Під час переміщення мурахи залишають за собою слід феромону, що вказує на пройдений шлях, і інші мурахи використовують цю інформацію. Таким чином, для взаємодії між мурашами застосовується непряма

комунікація.

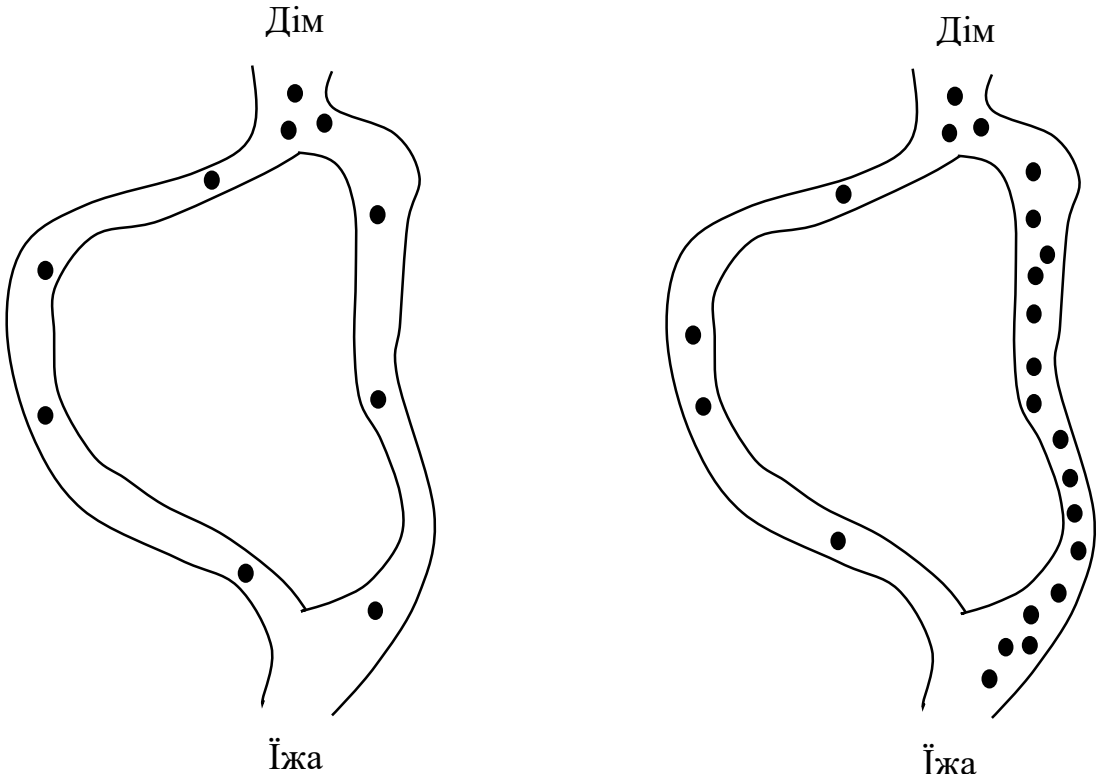

Рисунок 1.2 – Приклад роботи мурашиного алгоритму

Якщо розглядати рух мурах на графі, де кожен ребро представляє потенційний маршрут, можна помітити, що крім позитивного зворотного зв'язку (коли мурахи залишають феромон), існує й негативний зворотній зв'язок у вигляді випаровування феромону. Наявність цього випаровування гарантує, що мурахи будуть розглядати інші маршрути від мурашника до джерела їжі, забезпечуючи таким чином різноманіття вибору траєкторій. Оптимальним рішенням задачі стане маршрут з найбільшою кількістю феромону на його ребрах.

1.2 Алгоритм Джонсона

Алгоритм Джонсона (рис. 1.3) призначений для пошуку найкоротших шляхів між усіма парами вершин у зваженому орієнтованому графі. Він працює як у

випадку позитивних, так і у випадку негативних ваг ребер, за умови, що граф не має циклів з негативною вагою.

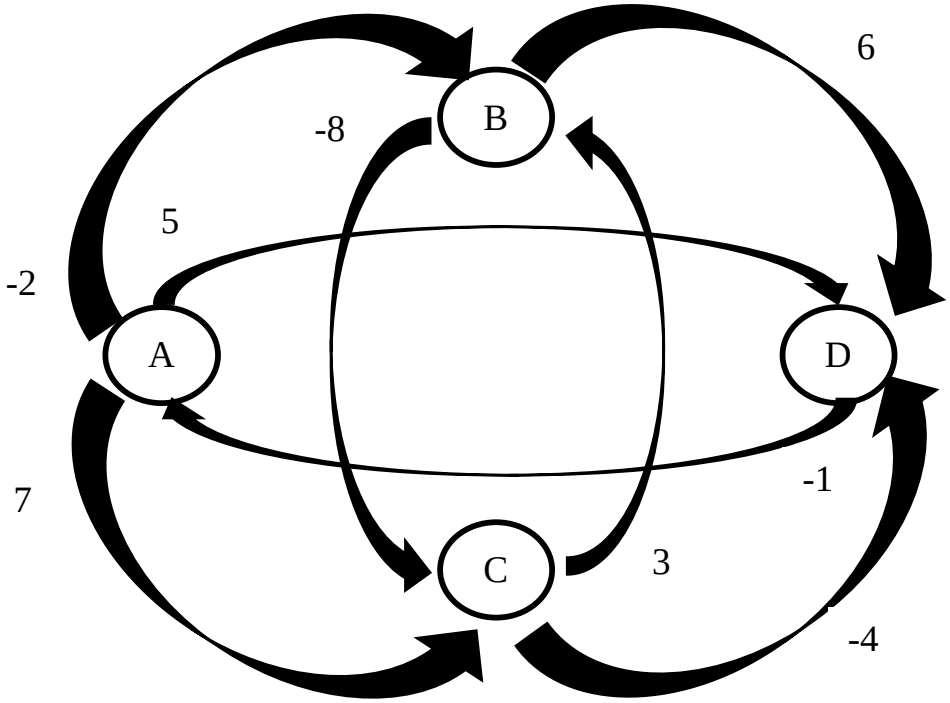

Рисунок 1.3 – Алгоритм Джонсона на орграфі з від'ємними дугами

Алгоритм складається з наступних кроків:

1. Для графу $G = (V, E)$ з ваговою функцією ω: $E \rightarrow R$, якщо всі ваги ребер невід'ємні, можна знайти найкоротші шляхи між усіма парами вершин, застосовуючи алгоритм Дейкстри для кожної вершини один раз.

2. Якщо граф містить ребра з негативною вагою, але немає циклів з негативною вагою, можна перерахувати ваги ребер таким чином, щоб усі вони стали невід'ємними, а потім використовувати аналогічний метод.

3. Нові ваги ребер повинні задовольняти певні властивості:

– для всіх ребер (u, v) нова вага $\omega(u, v) > 0$;

– для всіх пар вершин (u, v) шлях є найкоротшим використанням нових ваг, тоді і тільки тоді, коли він також є найкоротшим шляхом з використанням вихідних ваг.

Зберігання найкоротших шляхів. Нехай дано зважений орієнтований граф $G = (V, E)$ з ваговою функцією $\omega: E \to R$, і нехай $h: E \to R$ - довільна функція, яка відображає ребра на дійсні числа. Для кожного ребра $(u, v) \in E$ визначаємо:

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

Нехай $p = < v_0, v_1, \ldots, v_k >$ - будь-який шлях від вершини v_0 до вершини v_k. Шлях p є найкоротшим шляхом з ваговою функцією ω тоді і тільки тоді, коли він є найкоротшим шляхом з ваговою функцією ω, тобто рівність $\omega(\mathrm{p}) = \delta(v_0, v_1)$ еквівалентна $\omega(\mathrm{p}) = \delta(v_0, v_1)$. Крім того, граф G містить цикл з негативною вагою використаних вагової функції ω тоді і тільки тоді, коли він містить цикл з негативною вагою з використаних вагової функції ω.

Зміна ваги полягає в наступному: для даного графа створюється новий граф $G' = (V', E')$, де $V' = V \cup \{ s \}$, для деякої нової вершини $s \notin V$, а $E' = E \cup \{ (s, v) : v \in V \}$. Виконується розширення вагової функції ω таким чином, щоб для всіх вершин $v \in V$ виконувалась рівність $\omega(s, v) = 0$. Далі визначається для всіх вершин $v \in V$ величина $h(v) = \delta(s, v)$, і нові ваги для всіх ребер $\omega(u, v) = \omega(u, v) + h(u) - h(v) \geq 0$. Недоліком цього алгоритму полягає в тому, що загалом робота алгоритму може зайняти значний проміжок часу.

1.3 Алгоритм пошуку в глибину (ширину)

Метод пошуку в глибину (DFS) є одним із відомих способів проходження графа. Його алгоритм можна описати так: для кожної вершини, яка ще не була відвідана, потрібно знайти всі суміжні невідвідані вершини і продовжити пошук з них. DFS використовується як підпрограма в алгоритмах пошуку одно- і двозв'язних компонент, а також топологічного сортування.

Приклад порядку обходу дерева в глибину наведений на рис. 1.4.

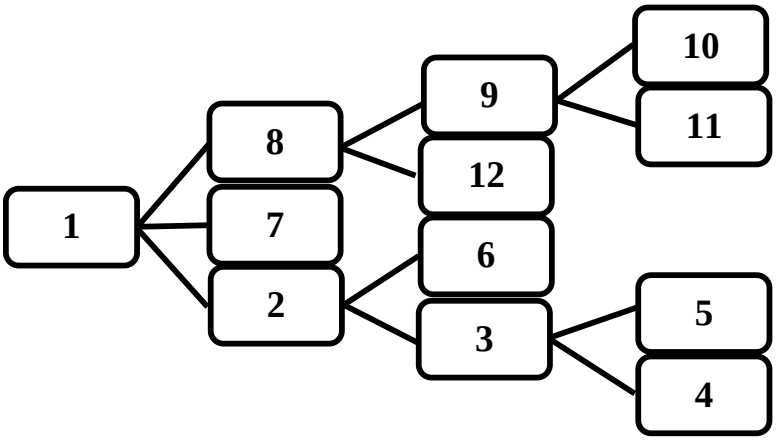

Рисунок 1.4 – Порядок обходу дерева в глибину

Пошук в глибину можна описати наступним чином: спочатку обираємо будь-яку невідвідану вершину з графа та починаємо з неї DFS. Після цього ми позначаємо вершину, яку вже обробили, як чорну. Ці кроки повторюються для всіх невідвіданих вершин доти, доки не залишиться жодної білої вершини.

Недоліком цього алгоритму є можливість повторного проходження кроків та відсутність здатності знаходити цільовий вузол.

Пошук в ширину (BFS) - інший метод проходження та позначення вершин графа. У BFS спочатку вершині, з якої починається обхід, приписується мітка 0, всім її сусідам - мітка 1, потім розглядаються вершини з міткою 1, їхнім сусідам приписується мітка 2, і так далі. Якщо граф зв'язний, BFS позначить всі його вершини. Приклад такого обходу наведено на рисунку 1.5.

Подвійний пошук в ширину - це алгоритм, який часто вдосконалює звичайний BFS, використовуючи два паралельних обходи починаючи з початкової та кінцевої вершин. Він завершується, коли вершина з одного напряму знаходить сусідню вершину з іншого напряму, що часто призводить до подвоєння швидкості обходу в порівнянні зі звичайним BFS.

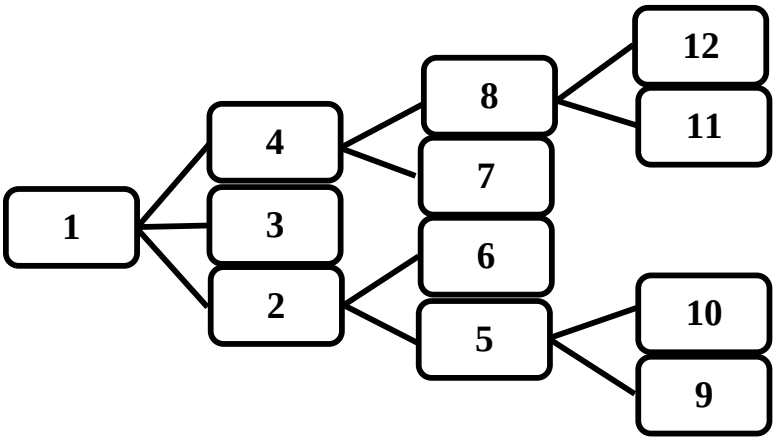

Рисунок 1.5 – Порядок обходу дерева в ширину

Однак, найбільш складним випадком для цього методу є ситуація, коли мета визначена лише неявним описом певної (і можливо, дуже об'ємної) множини цільових станів. У такому разі, для зворотного пошуку було б доцільно створити компактні описи всіх можливих станів, які можуть бути досягнуті за певну кількість кроків.

Найбільш серйозним недоліком пошуку в ширину є велике використання оперативної пам'яті, оскільки потрібно зберігати всі рівні дерева для створення наступних.

1.4 Застосунки мурашина логістика та Waze

Опис. Мурашина логістика — це сучасний український мобільний додаток, створений для полегшення планування та оптимізації маршрутів доставки. Його основна мета — допомогти логістичним компаніям та кур'єрським службам ефективніше управляти своїми ресурсами та знижувати витрати на доставку. Додаток пропонує широкий спектр функцій, які дозволяють автоматично розраховувати оптимальні маршрути з урахуванням різних факторів, таких як відстань, трафік та кількість зупинок. На рисунку 1.6 наведений логотип компанії,

що досліджує мурашиний алгоритм.

Рисунок 1.6 – Логотип компанії ANT-Logistics [5]

Переваги застосунку. Однією з ключових переваг Мурашиної логістики є можливість відстеження транспортних засобів у режимі реального часу. Це дозволяє менеджерам завжди бути в курсі місцезнаходження кур'єрів, що значно полегшує контроль за виконанням замовлень та забезпечує точне виконання термінів доставки. Крім того, додаток надає інструменти для аналізу ефективності маршрутів, що допомагає виявити слабкі місця у логістичних процесах та знайти шляхи для їх покращення.

Інтерфейс додатку (рис. 1.7) інтуїтивно зрозумілий та зручний у використанні, що дозволяє швидко освоїтися як водіям, так і менеджерам. Мурашина логістика підтримує роботу з різними типами транспортних засобів, що робить його універсальним інструментом як для невеликих кур'єрських служб, так і для великих логістичних компаній.

Мурашина логістика також забезпечує можливість налаштування параметрів доставки відповідно до специфічних потреб кожної компанії. Це включає встановлення пріоритетів для замовлень, обмежень по вазі та об'єму вантажу, а також вимог до часу доставки. Така гнучкість дозволяє компаніям адаптувати процеси під свої індивідуальні потреби, що сприяє підвищенню ефективності та задоволеності клієнтів.

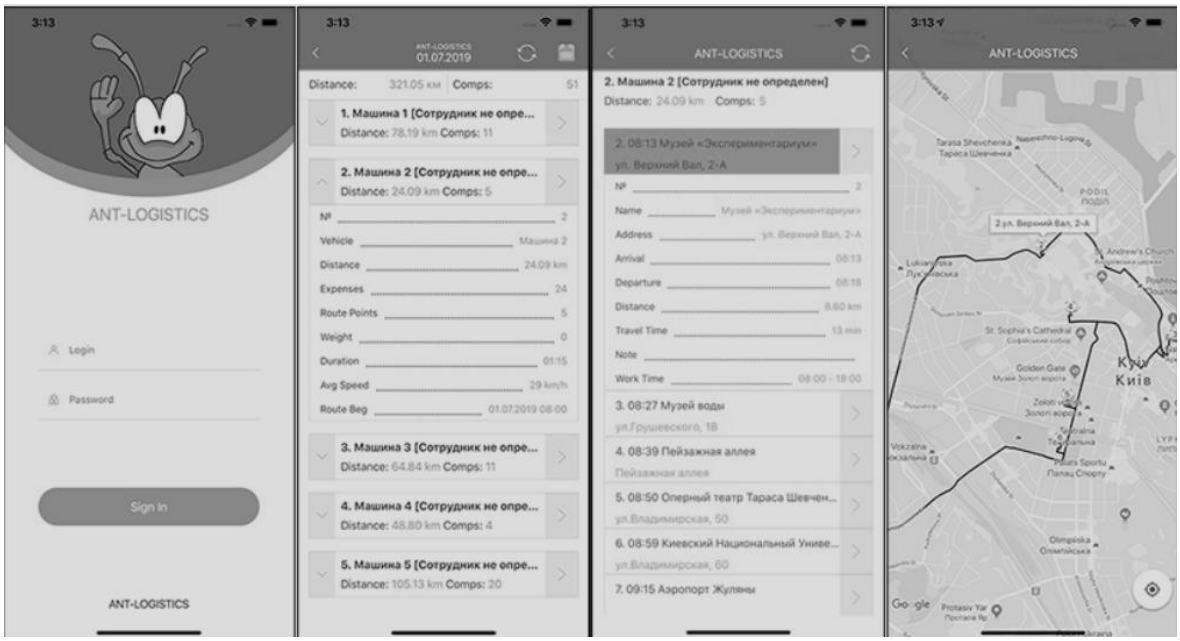

Рисунок 1.7 – Інтерфейс застосунку ANT-Logistics [5]

Завдяки інтеграції з різними системами управління та обліку, Мурашина логістика може стати невід'ємною частиною загальної інформаційної інфраструктури компанії. Це полегшує процес обміну даними між різними відділами та забезпечує більш ефективне управління ресурсами. Крім того, система надає детальні звіти та аналітику, які допомагають керівникам приймати обґрунтовані рішення на основі точних даних.

Для водіїв додаток пропонує простий та зрозумілий інтерфейс з інструкціями по маршруту, що дозволяє їм зосередитись на дорозі та зменшує ризик помилок. Функція голосової навігації допомагає водіям не відволікатися від водіння, а підтримка різних мов робить додаток доступним для широкого кола користувачів.

Мурашина логістика також активно працює над забезпеченням безпеки даних. Використання сучасних технологій шифрування та захисту інформації гарантує, що всі дані користувачів залишаються конфіденційними та захищеними від несанкціонованого доступу.

Крім того, розробники Мурашиної логістики регулярно оновлюють додаток, додаючи нові функції та вдосконалюючи існуючі, що робить його ще більш

корисним та зручним у використанні. Команда підтримки завжди готова допомогти користувачам вирішити будь-які питання або проблеми, що виникають під час роботи з додатком.

Недоліки. Мурашина логістика, попри свої переваги, має також деякі недоліки, які можуть впливати на користувацький досвід та ефективність її використання.

Одним з основних недоліків є висока вартість впровадження та підтримки додатку. Для невеликих компаній або стартапів це може стати значною перешкодою, оскільки вони можуть не мати достатнього бюджету для покриття витрат на придбання та налаштування програмного забезпечення, а також на навчання персоналу.

Мурашина логістика потребує стабільного інтернет-з'єднання для оновлення інформації про трафік, оптимізації маршрутів та відстеження транспортних засобів у реальному часі. У районах зі слабким інтернет-покриттям або в умовах, коли зв'язок тимчасово відсутній, можливості додатку значно обмежуються. Це може створювати проблеми під час роботи в віддалених або сільських місцевостях.

Хоча додаток підтримує інтеграцію з іншими системами управління логістикою та обліку, процес налаштування та інтеграції може бути складним і вимагати значних технічних зусиль. Особливо це стосується компаній, які використовують нестандартні або застарілі системи, що може призвести до додаткових витрат та часу на адаптацію.

Без підключення до інтернету багато ключових функцій додатку, таких як оновлення даних про трафік і оптимізація маршрутів, стають недоступними. Це значно обмежує корисність додатку в умовах відсутності зв'язку і може призвести до неефективного планування маршрутів і затримок у доставці.

Ефективність роботи додатку багато в чому залежить від точності введених даних та актуальності інформації про трафік. Якщо ці дані є неточними або застарілими, це може призвести до неправильних рішень щодо маршрутів, що в

свою чергу може спричинити затримки в доставці та підвищення витрат на паливо.

Незважаючи на інтуїтивно зрозумілий інтерфейс, нові користувачі можуть стикатися зі складнощами при освоєнні всіх можливостей та налаштувань додатку. Це може вимагати додаткового навчання або підтримки, що створює додаткові витрати часу та ресурсів для компанії.

Для дуже великих компаній з великою кількістю транспортних засобів та складною логістикою додаток може не завжди забезпечувати необхідну масштабованість та швидкість обробки даних. Це може обмежувати його ефективність у масштабних операціях і вимагати додаткових рішень для покращення продуктивності.

Деякі функції, такі як дані про трафік та оптимізація маршрутів, можуть бути менш точними або зовсім недоступними в певних регіонах або країнах. Це обмежує корисність додатку для міжнародних компаній або тих, що працюють у віддалених або менш розвинених регіонах.

Тривале використання додатку, особливо в режимі навігації, може швидко розряджати батарею смартфона. Це може бути проблематично для водіїв, які знаходяться на дорозі протягом тривалого часу і не завжди мають можливість зарядити свої пристрої.

Незважаючи на ці недоліки, Мурашина логістика залишається потужним інструментом для оптимізації логістичних процесів. Вона допомагає знижувати витрати на паливо та робочу силу, покращувати якість обслуговування клієнтів та підвищувати загальну продуктивність компанії. Завдяки своїм можливостям та функціональності, Мурашина логістика стає незамінним помічником у сфері доставки та логістики.

Висновок. Мурашина логістика є комплексним рішенням для оптимізації логістичних процесів, яке може значно підвищити ефективність роботи компанії, знизити витрати та покращити якість обслуговування клієнтів. Завдяки своїм широким можливостям та гнучкості, цей додаток стає незамінним інструментом

для сучасних логістичних компаній.

Перейдемо до застосунку Waze, який можна розглядати як соціально-технічну систему, що об'єднує в собі аспекти інформаційних технологій, соціальних взаємодій та географічної навігації. Цей застосунок працює на основі збору та обробки даних, включаючи геолокаційні дані, відгуки користувачів та інші вхідні параметри, з метою надання оптимальних маршрутів та інформації про дорожні умови.

З технічної точки зору, Waze (рис. 1.8) використовує сучасні технології глобального позиціонування (GPS), обробки геоданих та зв'язку з мобільними мережами для забезпечення навігаційних функцій. Він також використовує алгоритми машинного навчання для аналізу великих обсягів даних і виявлення патернів у руху транспортних засобів та дорожніх умовах.

Застосунок може бути предметом досліджень у сфері географії та транспортної логістики, де вивчаються його вплив на рух транспорту, ефективність маршрутів та стратегії управління транспортним потоком.

Переваги. Waze допомагає заощадити твій час. Завдяки своїй унікальній системі, яка аналізує дані про дорожні умови в реальному часі, застосунок прокладає для тебе оптимальний маршрут, уникнувши заторів та затримок.

Застосунок забезпечує твою безпеку на дорозі. Він попереджає про небезпечні відрізки доріг, аварії та інші потенційні загрози, щоб ти міг уникнути їх або прийняти необхідні заходи безпеки. Також застосунок надає інформацію про швидкість руху, дозволяючи тобі пристосувати свій спосіб водіння до умов дороги та дотримуватися правил безпеки.

Waze створює спільноту водіїв, яка ділиться корисною інформацією та досвідом. Ти можеш брати участь у цій спільноті, доповідаючи про події на дорозі, надаючи зворотний зв'язок та отримуючи актуальні повідомлення від інших користувачів.

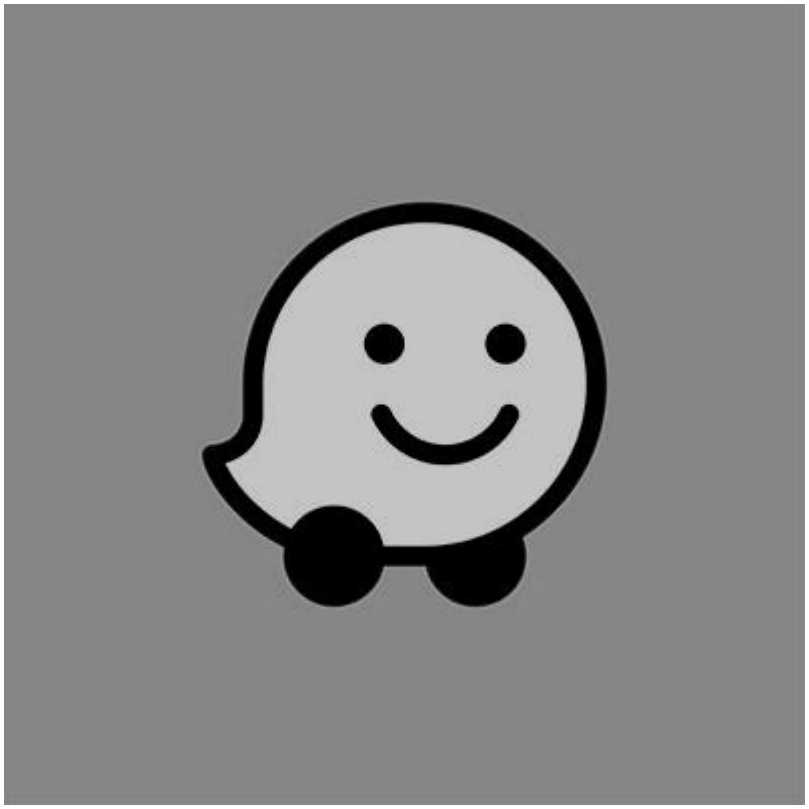

Рисунок 1.8 – Логотип застосунку Waze [11]

Недоліки. Хоча Waze є потужним і корисним інструментом для навігації, він має й свої недоліки.

Залежність від Інтернет-з'єднання може бути проблемою. Опинившись у місці з поганим прийомом сигналу або зовсім поза межами зони покриття мобільного оператора, Waze може втратити зв'язок із супутником, що призведе до втрати навігації.

Застосунок може бути вразливим до неточностей у данних. Хоча велика спільнота користувачів надає важливу інформацію про дорожні умови, іноді ця інформація може бути недостовірною або застарілою, що призводить до неочікуваних заторів або інших проблем на дорозі.

Іноді Waze може прокладати не найоптимальніший маршрут. Це може статися через обмежені дані про дорожні умови або через алгоритм, який не завжди враховує усі можливі фактори, такі як попередження про ремонт доріг чи тимчасові

перешкоди. Під час використання цього застосунку варто бути обережним і ретельно аналізувати інформацію, яку він надає, а також враховувати інші фактори, які можуть вплинути на подорож.

Ще одним недоліком Waze є можливість відволікання від дороги. Оскільки останній часто надає користувачам багато додаткової інформації, такої як повідомлення про події на дорозі, місця для зупинок та рекламні оголошення, це може відволікти водія від процесу водіння та створити небезпечні ситуації на дорозі.

Крім того, деякі користувачі відзначають, що інтерфейс Waze (рис. 1.9) може бути складним для розуміння або використання, особливо для новачків. Це може стати проблемою, особливо коли водії використовують застосунок вже під час поїздки, коли важко відволікатися на складні меню та опції.

Нарешті, як і у будь-якого іншого застосунку, існує ризик збоїв або помилок. Хоча Waze розроблений з високим рівнем надійності та стабільності, іноді можуть виникати технічні проблеми, такі як затримки у завантаженні карти або неправильна робота функцій навігації.

Рисунок 1.9 – Інтерфейс застосунку Waze [11]

Висновок. Застосунок Waze, як інноваційний продукт у сфері навігації та транспорту, відображає синергію між технологічними можливостями та соціальним впливом. Його успіх полягає в здатності об'єднувати велику спільноту користувачів, використовуючи засоби мобільних технологій для збору та обробки даних, що впливають на дорожні умови та навігацію.

Технічна складова Waze використовує передові технології, такі як глобальне позиціонування, машинне навчання та обробка великих обсягів даних, для надання користувачам точних та актуальних інформаційних послуг. Соціальний аспект застосунку сприяє активному обміну інформацією між користувачами, що підвищує якість інформації та ефективність навігації.

Застосунок Waze став не тільки незамінним інструментом для користувачів у подорожах, але й об'єктом досліджень у багатьох наукових дисциплінах, таких як географія, транспортна логістика та інформаційні технології. Вивчення впливу Waze на дорожні умови, рух транспорту та споживання енергії може мати значний інтерес для розвитку міської інфраструктури та транспортних систем.

Висновки до розділу 1

Під час аналізу наявних алгоритмів пошуку оптимального маршруту було виявлено переваги та недоліки кожного з них. Наприклад, аналіз показав, що алгоритм A* може бути неефективним у випадку, коли існує кілька локальних рішень з однаковою евристикою, яка відповідає вартості оптимального шляху. У той же час, метод гілок і меж використовується для пошуку оптимальних рішень різних оптимізаційних задач, але кожен раз він відкидає підмножину допустимих рішень, що не містять оптимальних, ускладнюючи процес пошуку та збільшуючи час виконання.

Також, метод найближчого сусіда, хоча й є жадібним і простим, часто має

велику похибку в результаті, що робить його неприйнятним для більшості практичних задач. Алгоритми пошуку в ширину та глибину, використовані в деяких алгоритмах, обмежуються застосуванням лише до певних типів задач, тим самим скорочуючи їх універсальність.

Алгоритм Дейкстри знаходить найкоротші відстані від однієї вершини графа до всіх інших, але це працює лише для графів з позитивною вагою ребер, що обмежує його застосування. Алгоритм Джонсона, хоча і здатний знаходити найкоротші шляхи між усіма парами вершин у зваженому орієнтованому графі, також має свої обмеження, наприклад, він не працює без циклів з негативною вагою та потребує значних ресурсів для обчислення.

Метод мурашиного алгоритму моделює поведінку колонії мурах, які шукають найкоротший шлях від мурашника до джерела їжі та адаптуються до змін у середовищі. Цей метод дозволяє знаходити оптимальні рішення навіть в умовах постійних змін, та уникати застрягання на локальних оптимальних рішеннях.

Мурашина логістика - це потужний інструмент для оптимізації логістичних процесів, який надає компаніям можливість ефективно керувати доставкою та знижувати витрати. Незважаючи на деякі недоліки, такі як висока вартість та залежність від інтернету, переваги цього застосунку, такі як оптимізація маршрутів та реальний час відстеження, роблять його незамінним інструментом для сучасних логістичних компаній.

Застосунок Waze є не тільки потужним інструментом для навігації та планування маршрутів, але й соціальною спільнотою користувачів, яка обмінюється актуальною інформацією про дорожні умови. Його успіх базується на поєднанні передових технологій глобального позиціонування та соціального взаємодії, що робить його незамінним інструментом для подорожей та об'єктом досліджень у багатьох наукових дисциплінах. Waze відображає потенціал технологій для покращення якості життя та оптимізації міської інфраструктури.

Отже, для системи маршрутів електротранспорту був обраний метод

мурашиного алгоритму. Його вибір обумовлений не лише здатністю до отримання рішень, які не гірші за раніше розглянуті, але й його ефективністю в умовах постійних змін параметрів і можливістю уникнути застрягання на локальних оптимальних рішеннях. Наприклад, в умовах інтенсивного дорожнього трафіку, де можливість виникнення заторів та інших непередбачених ситуацій велика, цей метод може допомогти знайти швидше оптимальний маршрут.

2 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ МУРАШИНОГО АЛГОРИТМУ В СИСТЕМАХ ПЛАНУВАННЯ МАРШРУТІВ

2.1 Природний принцип алгоритму

Як відомо, комахи, зокрема мурахи різних видів, терміти та деякі види бджіл і ос, живуть у колоніях, що складаються з великої кількості індивідуумів, які взаємодіють між собою. Колонії комах здатні вирішувати різноманітні завдання оптимізації, які жодна комаха самотужки не змогла б виконати (наприклад, знаходження найкоротших шляхів до джерел їжі, розподіл робочої сили та кластеризація при організації гнізд).

Для великої кількості комах потрібна певна форма взаємозв'язку, яка дозволяє їм співпрацювати при вирішенні повсякденних завдань. Цей зв'язок між індивідуумами колонії може бути прямим, залежно від виду. Наприклад, коли бджола знаходить джерело продовольства, вона повідомляє напрямок і відстань до нього іншим бджолам, виконуючи характерний танець. Це приклад прямого зв'язку, оскільки інші бджоли повинні бачити танець, який виконує одна бджола, щоб визначити місцезнаходження джерела їжі. Інші форми прямого зв'язку включають фізичний контакт, обмін продовольством або деякими рідинами.

Непрямий зв'язок між особинами колонії передбачає, що представник виду змінює навколишнє середовище таким чином, щоб це змінило поведінку інших особин, які будуть взаємодіяти з цим зміненим середовищем у майбутньому. Одним із прикладів непрямого зв'язку є утворення феромонних слідів, що використовують деякі види мурах. Мураха при добуванні їжі позначає доріжку, залишаючи певну кількість феромону на своєму шляху, що спонукає інших особин слідувати по цьому шляху в процесі добування їжі.

Принцип зміни навколишнього середовища, щоб вплинути на поведінку, відомий як стігмергія, є ключовим методом організації в мурашиних колоніях, що

дозволяє їм самоорганізовуватися. Термін "самоорганізація" використовується для опису складної поведінки, що виникає при взаємодії простих агентів. За допомогою самоорганізації мурахи можуть розв'язувати складні завдання, що виникають у них щодня, завдяки її розподіленому та ефективному характеру. Мурашині колонії можуть підтримувати цю поведінку, навіть якщо багато мурах не приймають участь у процесі.

Для кращого розуміння механізму та здатності мурашиних колоній знаходити оптимальні рішення при пошуку найкоротшого шляху від гнізда до джерела їжі, були проведені деякі експерименти. У одному з таких експериментів колонії мурах з Аргентини були представлені двома шляхами однакової довжини. Після певного часу спостерігали, як мурахи збігалися на одному шляху, що призвело до відкидання альтернативних шляхів. Для підтвердження вибору найбільш короткого шляху був проведений подвійний експеримент "міст", де мурахи були змушені обирати між коротким та довгим шляхами двічі (рис. 2.1).

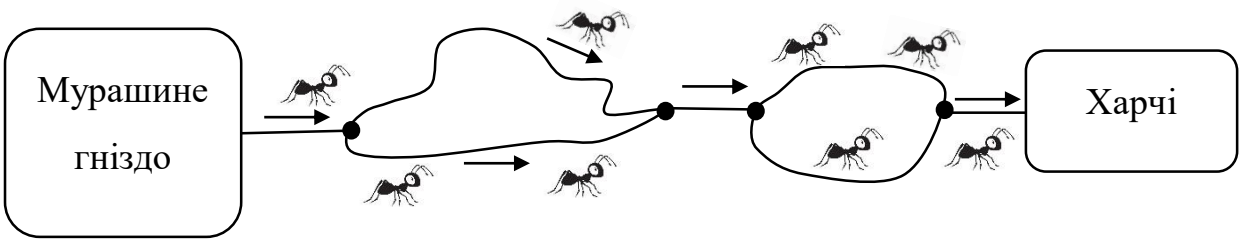

Рисунок 2.1 – Подвійний експеримент моста [25]

Мурахи з Аргентини, фактично, не мають можливості прямо визначити найкоротший шлях через те, що вони є фактично сліпими. Однак, незважаючи на цю обставину, колонія мурах демонструє здатність виявляти найкоротший шлях, що сполучає гніздо з ділянкою, де знаходиться їжа.

Спочатку всі мурахи знаходяться на ділянці гнізда. Частина мурах вирушає в пошуках їжі, при цьому кожна з них залишає за собою слід феромону, і досягає першого пункту в точці А. Оскільки мурахи не мають інформації про те, який шлях

обрати, кожна мураха обирає випадково, чи йти вправо, чи вліво.

Після певного часу близько половини мурах обирають коротший шлях, а решта - довший шлях до перетину B.

Мурахи, які обирають коротший шлях, досягають перетину швидше і мають вибрати, чи повернутися назад. Тоді виникає ситуація, коли немає жодної інформації для орієнтування, і деякі мурахи повертаються до гнізда, в той час як інші продовжують рухатися до джерела їжі.

Мурахи, які обирають довший шлях, також досягають перетину B, але, оскільки слід феромону на шляху назад до гнізда більш концентрований, більшість мурах повертається до гнізда, в той час як інші мурахи, які обрали довший шлях, також повертаються. Таким чином, наступні мурахи, які поки що не вирушили, будуть віддавати перевагу коротшому шляху через гніздо.

Поведінка мурах на ділянці між пунктами C і D повторюється подібно до того, як це відбулося на першому мосту між перетинами A і B. В кінцевому підсумку, багато мурах досягнуть кінцевого пункту і принесуть харчі до гнізда.

Після досягнення пункту D, мурахи віддають перевагу коротшому шляху за тим самим принципом, що й мурахи, які рухалися швидше. Аналогічна ситуація повториться в пункті B.

Оскільки кількість феромону між пунктами A і C на зворотному шляху до гнізда приблизно дорівнює загальній кількості феромону на цих двох ділянках від гнізда, то цей шлях стає найбільш ймовірним найкоротшим зворотнім шляхом.

Постійне розподілення мурахами феромону в процесі їх руху призводить до того, що короткий шлях неперервно посилюється за участю більшої кількості мурах, поки феромон на цьому шляху не стане настільки інтенсивним, що затінить інші можливі маршрути. В результаті практично всі мурахи використовуватимуть найкоротший шлях, оскільки система самозміцнюється.

Слід відзначити, що феромон, який використовують мурахи, поступово випаровується з часом. Дійсно, шляхи, які не використовувалися для певного часу,

стають досить довгими, і через випаровування феромону після певного періоду часу вони майже не містять слідів феромону. Це збільшує ймовірність вибору мурахами коротких шляхів.

Одним з перших завдань, на які застосовувався метод мурашиних колоній, були задачі комівояжера. Вибір цієї задачі обумовлений тим, що вона передбачає пошук найкоротшого шляху між пунктами, тому аналогія з методом мурашиних колоній досить легко застосовується для її розв'язання.

Для вирішення цієї задачі було розроблено кілька різних методів, які базуються на оптимізації за допомогою мурашиних колоній. Першим з цих методів був метод мурашиних систем, який потім став основою для розробки інших алгоритмів, що працюють на засадах мурашиних колоній.

У методі мурашиних систем агент формує своє рішення під час переміщення від одного вузла до іншого на графі рішень. Метод працює до виконання певної кількості ітерацій. На кожній ітерації агенти формують свої рішення на основі правила вибору наступного вузла, яке залежить від вибору агента, що знаходиться на певному вузлі.

Спочатку було запропоновано три методи мурашиних систем, які відрізнялися способом оновлення шляхів - ребер: щільнісний, кількісний і циклічний. У щільнісному і кількісному методах агенти залишали феромони під час формування рішення, тоді як у циклічному методі феромони залишалися після завершення переміщення, тобто після вибору рішення.

Експерименти показали, що циклічний метод набагато краще справляється з задачею в порівнянні з іншими двома методами. Через це два менш ефективні методи були відкинуті, і тепер під методом мурашиних колоній розуміється саме циклічний метод мурашиних систем.

Початкова популяція. Після формування популяції мурахи рівномірно розподіляються між вузлами мережі. Це необхідно для того, щоб кожен вузол мав однакові можливості стати стартовою точкою. Якщо всі мурахи рухатимуться з

однієї точки, це може здатися оптимальним варіантом, але насправді ми не знаємо, яка точка є найбільш ефективною для старту.

Для кожної мурахи перехід з пункту i в пункт j залежить від трьох факторів: пам'яті мурахи, видимості та віртуального сліду феромону. Пам'ять мурахи (TabuList) це список, що містить бітовий масив, який відображає відвідані та невідвідані вузли. Кожен вузол повинен бути відвіданий мурахою лише один раз, а вузли у поточному шляху (Path) розташовані у порядку їх відвідування. Цей список очищується після кожної ітерації алгоритму. Позначимо через $J\ (i,k)$ список міст, які ще не відвідав мурасі k, який знаходиться в місті i.

Видимість (n_{ij}) - це величина, обернена відстані між пунктами i та j. Це локальна статична інформація, яка відображає необхідність відвідати найближчий пункт. Однак лише використання цієї величини недостатньо для знаходження найоптимальнішого маршруту.

Віртуальний слід феромону на ребрі (ij) виражає бажання відвідати пункт j з пункту i. Цей параметр відображає перевагу вибору даного шляху в процесі переміщення. Кількість феромону, яка залишається на ребрі, пропорційна якості маршруту, що складений мурахою. Цей підхід дозволяє забезпечити безпосередній пошук найкращого рішення.

2.2 Оптимізація задач із застосуванням мурашиного алгоритму

Проблема розкладу робіт (Job Scheduling Problem) передбачає розподіл набору робіт серед певної кількості ресурсів (наприклад, машин або працівників) з метою мінімізації загального часу виконання (makespan) або інших критеріїв оптимізації, таких як витрати або запізнення.

Мурашині алгоритми використовуються для цієї задачі шляхом моделювання кожного ресурсу як маршруту, який повинні пройти мурахи. Кожна мураха створює потенційний розклад, намагаючись мінімізувати заданий критерій. Після

декількох ітерацій, оптимальні або близькі до оптимальних розклади виявляються шляхом концентрації феромонів на більш ефективних розподілах.

Проблема маршрутизації транспортних засобів (Vehicle Routing Problem, VRP) є розширенням задачі комівояжера і включає планування маршрутів для автопарку транспортних засобів, які повинні доставити товари до набору клієнтів. Мета полягає в мінімізації загальної довжини маршрутів або інших витрат, враховуючи обмеження, такі як місткість транспортних засобів і часові вікна для доставок.

Мурашині алгоритми ефективно застосовуються до VRP, де кожна мураха створює потенційний маршрут для кожного транспортного засобу. Використовуючи інформацію про феромони та евристичну інформацію, мурахи можуть знаходити оптимальні або близькі до оптимальних маршрути. Оновлення феромонів допомагає спрямовувати пошук у найбільш перспективні області.

Маршрутизація в телекомунікаційних мережах. Іншим важливим застосуванням мурашиних алгоритмів є маршрутизація даних у телекомунікаційних мережах. У цій задачі необхідно визначити найкращі маршрути для передачі пакетів даних через мережу з метою мінімізації затримок і максимізації пропускної здатності.

Мурашині алгоритми використовуються для динамічного визначення маршрутів у реальному часі. Мурахи представляють собою агентів, які рухаються через мережу, збираючи інформацію про затримки та пропускну здатність, і залишаючи феромони на найкращих маршрутах. Це дозволяє мережі адаптуватися до змін у трафіку та обирати оптимальні маршрути для передачі даних.

Проблема кластеризації даних. Кластеризація даних є важливою задачею в машинному навчанні та аналізі даних, де необхідно розподілити набір об'єктів у групи (кластери) таким чином, щоб об'єкти в одному кластері були схожі між собою, а об'єкти з різних кластерів були максимально відмінними.

Мурашині алгоритми застосовуються до кластеризації шляхом моделювання

кожного об'єкта як вузла, між якими мурахи будують шляхи. Феромони використовуються для позначення подібності між об'єктами, а евристична інформація допомагає визначити потенційні кластери. Мурахи поступово знаходять оптимальні розподіли об'єктів по кластерах, концентруючись на найбільш схожих групах.

2.3 Поняття задачі комівояжера

Задача комівояжера — це одна з найвідоміших та найважчих задач комбінаторної оптимізації. Вона полягає в пошуку найкоротшого маршруту, який пройде через всі міста (або точки) один раз і повернеться в початкове місто.

Формально, задача комівояжера може бути сформульована так: нехай є набір міст, де кожне місто з'єднане з кожним іншим дорогою певної довжини (або ваги). Метою є знайти найкоротший шлях, який проходить через кожне місто рівно один раз та повертається в початкове місто.

Задача комівояжера є NP-повною, що означає, що немає ефективного алгоритму, який знайде оптимальне рішення для будь-якого вхідного набору даних за прийнятним часом. Тим не менш, існують різні методи наближеного розв'язання цієї задачі, такі як генетичні алгоритми, мурашині алгоритми та інші метаевристичні методи.

Застосування задачі комівояжера включає маршрутизацію в транспортних мережах, планування маршрутів для продажу та доставки, організацію логістичних систем та багато іншого. Вона також має важливе теоретичне значення у вивченні обчислювальної складності та оптимізаційних алгоритмів.

Зважаючи на широке застосування задачі комівояжера, вона зацікавлює як дослідників, так і практиків. Особливо важливим є її використання в сфері транспорту та логістики, де оптимізація маршрутів може значно зменшити витрати на паливо, час та інші ресурси.

Однією з ключових властивостей задачі комівояжера є її комбінаторний характер, що робить знаходження оптимального розв'язку важким завданням. Це також призводить до того, що для великих наборів даних, складність обчислень може бути надто великою для ефективного розв'язання.

Застосування метаевристичних методів, таких як генетичні алгоритми або мурашині алгоритми, може допомогти знайти прийнятні наближені розв'язки в області, де точні методи стають непрактичними через великий обсяг обчислень.

Загалом, задача комівояжера залишається важливою проблемою в області оптимізації, і дослідження в цьому напрямку продовжуються з метою знаходження нових ефективних методів розв'язання та впровадження їх у практику.

Поза сферою транспорту та логістики, задача комівояжера також знаходить застосування в інших галузях. Наприклад, у біології вона може використовуватися для планування маршрутів обстеження або дослідження молекулярних структур. В хімії вона може допомагати в оптимізації процесів синтезу речовин або аналізі хімічних сполук. У виробництві вона може бути корисною для мінімізації часу та ресурсів, необхідних для виготовлення деталей або збирання виробів на виробничих лініях.

Завдяки своїй універсальності та широкому спектру застосувань, задача комівояжера залишається актуальною для досліджень і розвитку нових методів оптимізації. Також варто зазначити, що в останні роки з'явилися методи, які поєднують у собі елементи штучного інтелекту, машинного навчання та оптимізації для знаходження більш ефективних розв'язків у складних умовах.

Таким чином, задача комівояжера залишається важливим об'єктом досліджень у багатьох галузях науки та технологій і відіграє важливу роль у вирішенні практичних завдань, пов'язаних з оптимізацією маршрутів та ресурсів.

Однією з ключових особливостей задачі комівояжера є її комбінаторна природа. Основне завдання полягає в тому, щоб знайти найкоротший можливий маршрут, що проходить через всі міста (точки) лише один раз і повертається в

початкове місто. Оскільки кількість можливих маршрутів зростає експоненціально зі збільшенням кількості міст, знаходження точного оптимального розв'язку для великих наборів даних стає вельми складною задачею.

Ще однією особливістю є NP-повнота задачі комівояжера, що означає, що для точного розв'язку потрібно виконати всі можливі комбінації маршрутів, що займає значний обсяг обчислювальних ресурсів для великих проблемних наборів даних. Це робить задачу комівояжера важкою навіть для потужних обчислювальних систем.

Ще однією особливістю є те, що в задачі комівояжера відсутній унікальний розв'язок, який би відповідав усім умовам. Різні алгоритми можуть знаходити різні маршрути, які, хоч і можуть бути оптимальними або наближеними до оптимальних, але не обов'язково ідентичними.

Для вирішення задачі комівояжера застосовуються різні методи, такі як евристичні алгоритми (наприклад, генетичні або мурашині алгоритми), точні алгоритми (наприклад, метод гілок та границь), а також методи наближення та зближення, які намагаються знайти оптимальне або найкраще можливе рішення за прийнятний час.

2.4 Модифікації мурашиного алгоритму

Модифікації мурашиного алгоритму включають в себе різноманітні вдосконалення та зміни, спрямовані на покращення ефективності і здатності алгоритму до вирішення різних завдань. Модифіковані мурашині системи впроваджують різноманітні удосконалення, такі як випадкові початкові розв'язки, механізми відбору мурах та зміни параметрів в процесі пошуку. Ці модифікації спрямовані на підвищення ефективності та здатності алгоритму до роботи з різноманітними задачами оптимізації, забезпечуючи більш гнучке та точне рішення.

Деякі з найпоширеніших модифікацій включають:

Таблиця 2.1 – Різновиди методів мурашиних колоній

Назва	Опис
Мурашина система колоній (Ant Colony System, ACS)	Модифікація мурашиного алгоритму, яка використовує феромонні еліти та додаткові правила вибору шляху для покращення швидкості та точності пошуку оптимальних рішень.
Мурашиний Алгоритм з Мінімальним Сприйняттям (Max-Min Ant System, MMAS)	Модифікація мурашиного алгоритму, яка обмежує діапазон значень феромонів, що дозволяє уникнути перебору та збільшує швидкість збіжності.
Модифікованіа мурашина система (Modified Ant Colony Systems, MAS	Модифікація мурашиного алгоритму, яка включає різноманітні удосконалення, такі як випадкові початкові розв'язки, механізми відбору мурах та зміни параметрів в процесі пошуку.
Мурашина система з динамічними параметрами (Ant Colony Systems with Dynamic Parameters, ACSDP)	Модифікація мурашиного алгоритму, що використовує динамічні параметри, які змінюються в процесі пошуку відповідно до поточних умов. Це дозволяє адаптувати алгоритм до змінних умов середовища і покращує його здатність знаходити оптимальні рішення.
Мурашина система з елітними мурахами (Ant Colony Systems with Elite Ants, ASCEA)	Модифікація мурашиного алгоритму, яка використовує концепцію елітних мурах. Ці мурахи обирають та передають найкращі шляхи з одного покоління мурах до наступного, що прискорює пошук оптимального розв'язку.

Спільни риси. Модифікації мурашиного алгоритму поділяють спільні риси, які забезпечують їх ефективність у вирішенні задач оптимізації. Вони всі базуються на використанні феромонів для комунікації між мурахами та оцінки якості шляхів. Це дозволяє мурашам передавати інформацію про найдошуканіші маршрути, що сприяє в пошуку оптимального рішення. Далі, модифікації імітують природні механізми поведінки мурах, вибираючи шлях на основі концентрації феромонів та співпрацюючи для досягнення найкращого результату. Крім того, вони зазвичай використовують паралельну обробку інформації, що дозволяє мурашам працювати над декількома шляхами одночасно, збільшуючи швидкість пошуку. Нарешті, модифікації адаптуються до змін умов оточення, що дозволяє покращити ефективність пошуку в різних середовищах.

Відмінності. Модифікації мурашиного алгоритму мають відмінності, які визначають їхню унікальність у вирішенні задач оптимізації. Наприклад, Мурашині системи з динамічними параметрами використовують параметри, які змінюються в процесі пошуку, що дозволяє адаптувати алгоритм до змінюючихся умов та покращити ефективність пошуку. У той же час, Мурашині системи колоній використовують феромонні еліти, які вказують на кращі шляхи, що допомагають мурахам швидше відшукати оптимальний шлях. Також, Максимальна феромонна мурашині системи обмежують діапазон значень феромонів, щоб уникнути перебору та збільшити швидкість збіжності. Кожна з цих модифікацій має свої унікальні особливості, які роблять їх ефективними у різних випадках вирішення оптимізаційних завдань.

2.5 Застосування базового мурашиного алгоритму для розв'язання задачі пошуку оптимального маршруту

У цьому розділі розглядатимуться основні принципи роботи базового мурашиного алгоритму для вирішення задачі комівояжера. Маршрут або шлях

визначається як послідовність ребер у неорієнтованому графі, де кінець кожного ребра збігається з початком наступного. Довжина маршруту визначається кількістю ребер у ньому.

Для розв'язання алгоритмічних задач застосовують різноманітні стратегії та інструменти, які постійно розробляються для пошуку рішень у сфері високопродуктивних обчислень. Особливий інтерес викликають алгоритми, натхнені природними законами. До таких алгоритмів належать еволюційні алгоритми, які імітують певні аспекти поведінки та еволюційні риси людського геному. Крім того, такі алгоритми можуть бути натхненні природною поведінкою тварин. Основна мета останніх – забезпечити реалістичні, релевантні та водночас економічні рішення для задач, які раніше не могли бути розв'язані звичайними засобами.

На основі таких еволюційних алгоритмів було розроблено різні методи оптимізації, що призвело до появи метаевристик. Метаевристика походить від грецьких слів "meta" (що означає "понад" або "вище") і "heuriskein" (що означає "знаходити"). Метаевристика – це алгоритмічна структура високого рівня, яка є незалежною від конкретної проблеми і надає набір вказівок або стратегій для створення евристичних алгоритмів оптимізації. Прикладами таких алгоритмів є Метод рою часток (Particle Swarm Optimization, PSO) і Мурашиний алгоритм. Ройовий інтелект, до якого належать ці алгоритми, спрямований на розробку інтелектуальних багатоагентних систем, натхненних колективною поведінкою соціальних комах (таких як мурахи, терміти, бджоли, оси) та інших груп тварин (таких як зграї птахів або риб).

Техніка оптимізації колоній мурах повністю натхнена їхньою поведінкою під час пошуку їжі, вперше представлена Марко Доріго в 1990-х роках. Мурахи є еусоціальними комахами, які віддають перевагу виживанню та підтримці спільноти, а не індивідуальному існуванню. Еусоціальні істоти живуть у кооперативних групах, де зазвичай одна самка і кілька самців є репродуктивно

активними, а інші особини піклуються про молодняк або захищають і забезпечують групу. Вони спілкуються за допомогою звуків, дотиків та феромонів. Феромони – це органічні хімічні сполуки, що виділяються мурахами і викликають соціальну реакцію у представників того ж виду. Ці речовини діють як гормони поза тілом особини, що виділяє секрет, впливаючи на поведінку інших особин, які їх сприймають. Оскільки більшість мурах живуть на землі, вони залишають феромонні сліди на поверхні ґрунту, за якими можуть стежити інші мурахи.

Мурахи живуть у громадських гніздах, і основний принцип АСО полягає в спостереженні за рухом мурах від їхніх гнізд до джерел їжі найкоротшим можливим шляхом. Спочатку мурахи випадково переміщуються в пошуках їжі навколо своїх гнізд. Цей випадковий пошук відкриває кілька маршрутів від гнізда до джерела їжі. Після виявлення їжі, мурахи повертають частину їжі до гнізда, залишаючи феромонні сліди на зворотному шляху. Відповідно до концентрації феромонів, ймовірність вибору певного шляху наступними мурахами зростає. Ця ймовірність залежить від концентрації феромонів та швидкості їх випаровування. Таким чином, довжину кожного шляху можна легко визначити на основі швидкості випаровування феромонів.

На рис. 2.2, ми спростили ситуацію до розгляду лише двох можливих маршрутів між джерелом їжі та мурашиним гніздом. Процес можна розбити на такі етапи:

– усі мурахи знаходяться у своєму гнізді. Спочатку немає слідів феромонів у середовищі;

– мурахи розпочинають пошук з рівною ймовірністю (0,5 кожна) вздовж обох маршрутів. Очевидно, що довший шлях займе більше часу для проходження, тому ймовірність вибору коротшого шляху вища;

– мурахи, які обирають коротший шлях, досягають джерела їжі швидше. Тепер вони стикаються з аналогічною дилемою вибору, але через феромонний слід вздовж коротшого шляху, ймовірність його обрання ще більше зростає;

– з часом більше мурах обирають коротший шлях, що призводить до зростання концентрації феромонів на ньому. Також, через випаровування, концентрація феромонів на довшому шляху зменшується, що зменшує ймовірність його вибору. Таким чином, колонія поступово переходить на коротший шлях з вищою ймовірністю.

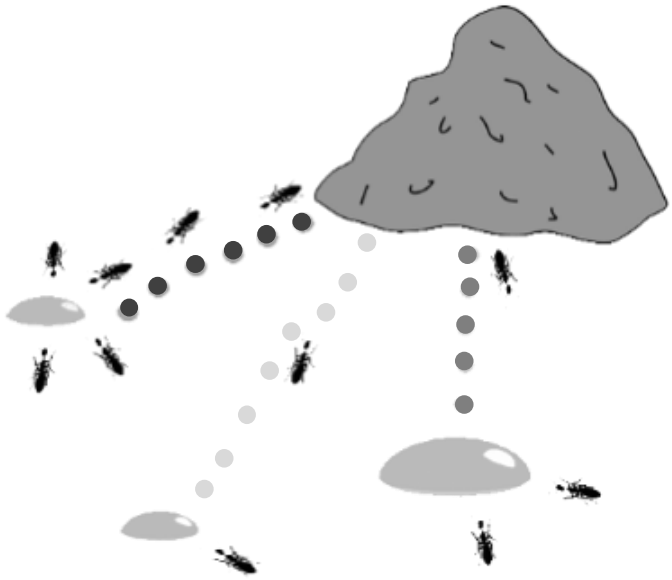

Рисунок 2.2 – Схематичне зображення роботи мурашиного алгоритму [26]

Таким чином, досягається оптимальний маршрут.

Описана вище поведінка мурах може бути використана для розробки алгоритмічного дизайну. Для спрощення, ми розглянули лише одне джерело їжі та одну мурашину колонію з двома можливими маршрутами. Цей сценарій можна легко втілити за допомогою зважених графів, де мурашине гніздо та джерело їжі є вершинами (вузлами), маршрути - ребрами, а значення феромонів - вагами, пов'язаними з ребрами.

Більш детальну схему мурашиного алгоритму можна побачити на рис. 2.3.

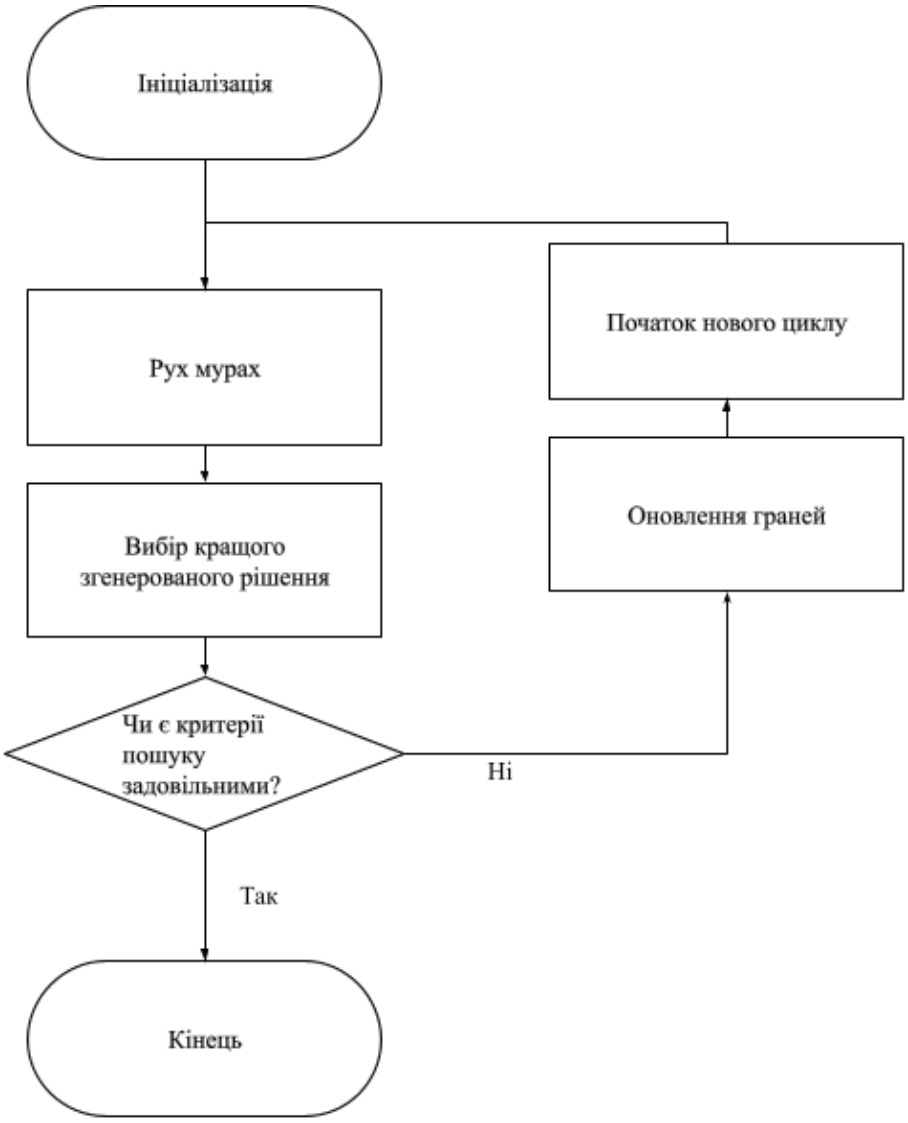

Рисунок 2.3 – Схема роботи мурашиного алгоритму [28]

Нехай граф $G = (V, E)$, де V, E – ребра та вершини графа. Вершини відповідно до нашого розгляду є V_s (вершина джерела – колонія мурах) і V_d (вершина призначення – джерело їжі), два ребра є E_1 і E_2 з довжинами L_1 і L_2, призначеними кожному. Тепер пов'язані значення феромонів (що вказують на їх силу) можна вважати R_1 і R_2 для вершин E_1 і E_2відповідно. Таким чином, для кожної мурахи початкова ймовірність вибору шляху (між E_1 і E_2) може бути виражена таким чином утворюється рівняння (2.1):

$$P_i = \frac{R_i}{R_1 + R_2};\ i = 1,2. \qquad (2.1)$$

Очевидно, що якщо $R_1 > R_2$, то ймовірність вибору E_1 вища і навпаки. Тепер під час повернення цим найкоротшим шляхом, скажімо E_1, значення феромону оновлюється для відповідного шляху. Оновлення виконується на основі довжини шляхів, а також швидкості випаровування феромону. Таким чином, оновлення можна поетапно реалізувати наступною формулою (2.2):

Відповідно до довжини шляху:

$$R_i \leftarrow R_i + \frac{K}{L_i}. \qquad (2.2)$$

У наведеному вище оновленні $i = 1,2$ і K служить параметром моделі. Крім того, оновлення залежить від довжини шляху. Чим коротший шлях, тим більше феромонів.

Відповідно до швидкості випаровування феромону за формулою (2.3):

$$R_i \leftarrow (1 - \upsilon) * R_i. \qquad (2.3)$$

Параметр υ належить до інтервалу (0, 1], який регулює випаровування феромону. Далі $i = 1,2$.

На кожній ітерації всі мурахи знаходяться в початковій вершині V_s (мурашиній колонії). Після першого кроку вони переміщуються від V_s до V_d (джерела їжі). Потім всі мурахи повертаються назад і посилюють обраний шлях на основі другого кроку.

Базовим математичним алгоритмом вважається той, що ґрунтується на поведінці мурашиної колонії – відзначенні успішних маршрутів значною кількістю феромонів. Процес починається з розміщення мурах у вершинах графа (містах), а

потім рух мурах визначається за допомогою ймовірнісного методу, використовуючи формулу (2.4):

$$P_i = \frac{l_i^q * f_i^p}{\sum_{k=0}^{N} l_k^q * f_k^p};\ i = 1,2. \quad (2.4)$$

де P_i -ймовірність вибору шляху i;

l_i - величина, обернена до довжини (ваги) i-ого переходу;

f_i - кількість феромонів на i-ому переход;

f_i - величина,яка визначає «жадібність» алгоритму;

f_i - величина,яка визначає «стадність» алгоритму i;

$q + p = 1$.

Отриманий результат може бути навіть не точним і може бути одним з гірших. Однак, завдяки ймовірному характеру рішення, повторні запуски алгоритму можуть дати досить точний результат.

Важливо зазначити, що феромони, що залишаються мурахами, поступово випаровуються з часом. Дійсно, для шляхів, які не вибиралися протягом певного часу, кількість феромонів стає майже незначною через їх випаровування, і, як наслідок, ймовірність вибору мурахами коротших шляхів зростає.

Висновки до розділу 2

У цьому розділі ми детально розглянули мурашиний алгоритм та його застосування в системах побудови маршрутів. На основі аналізу теоретичних основ і практичних застосувань, можна зробити такі висновки:

Мурашині алгоритми базуються на поведінці реальних мурах, які використовують феромони для знаходження найкоротших шляхів до джерел їжі. Цей природний принцип дозволяє штучним агентам (мурахам) ефективно вирішувати складні оптимізаційні задачі. Завдяки колективній поведінці мурах і

здатності феромонів до випаровування, алгоритм забезпечує збалансований пошук між дослідженням нових маршрутів і експлуатацією відомих хороших рішень.

Формування початкової популяції мурах є критично важливим етапом у роботі мурашиного алгоритму. Початкова популяція закладає основу для подальшого пошуку оптимального рішення. Від правильної ініціалізації залежить швидкість збіжності алгоритму та його здатність уникати локальних мінімумів. Дослідження показують, що рівномірний розподіл мурах по вузлах графу та різноманітність початкових маршрутів сприяють кращій продуктивності алгоритму.

Мурашині алгоритми продемонстрували високу ефективність у розв'язанні широкого спектра задач оптимізації, включаючи задачі комівояжера, розкладу задач, маршрутизації транспортних засобів та кластеризації даних. Використання мурашиного алгоритму дозволяє знаходити близькі до оптимальних рішення в умовах складних і великих просторів пошуку. Основні переваги включають гнучкість, масштабованість та здатність адаптуватися до різних умов і обмежень.

Задача комівояжера є класичною задачею комбінаторної оптимізації, яка полягає у знаходженні найкоротшого маршруту, що проходить через заданий набір міст і повертається до початкового міста. Мурашині алгоритми, завдяки своїй природній здатності знаходити оптимальні шляхи, є одним з найефективніших підходів для розв'язання TSP. Вони використовують механізми феромонів і евристичну інформацію для поступового покращення знайдених рішень.

Існує багато модифікацій мурашиного алгоритму, спрямованих на підвищення його ефективності та продуктивності. Серед них можна виділити такі підходи, як ACS, MMAS, ASCEA, MAS, ACSDP, та інші гібридні методи, які поєднують мурашиний алгоритм з іншими оптимізаційними техніками. Ці модифікації дозволяють покращити конвергенцію алгоритму, зменшити ймовірність попадання в локальні мінімуми та підвищити якість знайдених рішень.

Застосування базового мурашиного алгоритму для пошуку оптимального

маршруту показало його високу ефективність у реальних задачах. Алгоритм успішно використовується для планування маршрутів у логістиці, транспорті, телекомунікаційних мережах та інших сферах, де необхідно знайти оптимальні шляхи з урахуванням різних обмежень. Завдяки можливості адаптації до змінних умов та динамічному оновленню феромонів, мурашиний алгоритм забезпечує надійні та ефективні рішення.

Мурашині алгоритми є потужним інструментом для розв'язання складних задач оптимізації, зокрема в системах побудови маршрутів. Вони ефективно імітують природні принципи поведінки мурах, використовуючи колективну динаміку для знаходження оптимальних рішень. Початкова популяція, механізм оновлення феромонів, та різноманітні модифікації алгоритму дозволяють успішно застосовувати мурашині алгоритми для широкого спектра задач, включаючи TSP, розклад задач, маршрутизацію та кластеризацію. Завдяки цим характеристикам, мурашині алгоритми продовжують залишатися важливим і актуальним інструментом у галузі оптимізації.

3 РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПЛАНУВАННЯ МАРШРУТІВ ЕЛЕКТРОТРАНСПОРТУ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

У даному розділі буде детально розглянуто архітектуру програмного забезпечення, алгоритми оптимізації, які були використані, а також специфіку інтеграції з реальними даними. Ми обговоримо вибір технологій та інструментів, що були застосовані при розробці цього застосунку, і пояснимо, чому саме ці технології були визнані найефективнішими для вирішення поставленої задачі.

Перед тим як перейти до опису програмної реалізації застосунку для оптимізації маршруту електротранспорту потрібно провести дослідження який алгоритм обрати для оптимізації маршруту, було вибрано два популярних алгоритма а саме, алгоритм мурашиної колонії та алгоритм A*, теоретичний аспект цих двох алгоритмів було розглянуто у розділу вище, Тепер розглянемо їх програмну реалізацію.

3.1 Реалізація алгоритму мурашиної колонії

Як було відомо що алгоритм мурашиної колонії (Ant Colony Optimization, ACO) — це метод оптимізації, натхненний поведінкою реальних мурах при пошуку найкоротшого шляху від колонії до джерела їжі. Перед програмно потрібно розглянути блок схему алгоритму .

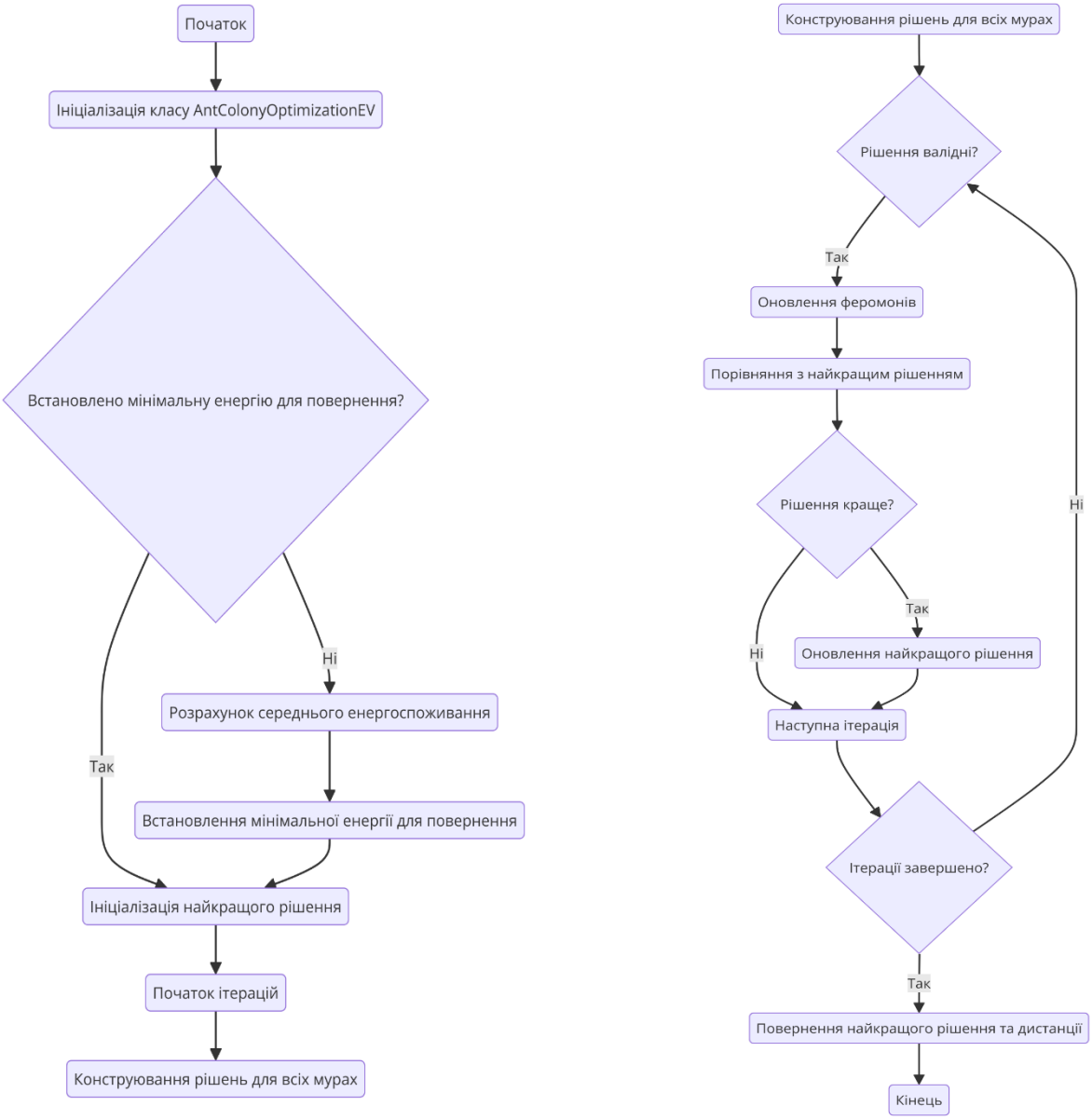

Рисунок 3.1 – Блок схема алгоритму мурашиної колонії [34]

На цій блок-схемі показано, що спочатку потрібно провести ініціалізацію класу AntColonyOptimizationEV, оскільки алгоритми, які він реалізує, можуть бути використані в програмній реалізації. Потім необхідно перевірити, чи потрібно встановити мінімальну енергію для повернення електротранспорту. Якщо це потрібно, розраховується середнє енергоспоживання і встановлюється мінімальна енергія для повернення. Якщо не потрібно, ініціалізується найкраще рішення.

Після цього починається ітерація мурах, де кожна мураха будує свій шлях.

Далі перевіряється якість отриманих рішень. Якщо рішення є прийнятними, оновлюються феромони і порівнюються з найкращим рішенням. Якщо нове рішення краще, то оновлюється найкраще рішення, інакше продовжується наступна ітерація. Після завершення всіх ітерацій повертається найкраще знайдене рішення і відповідні дистанції. Розглянемо окремі функції програмного коду алгоритму.

```
1 usage
def _construct_solution(self):
    solutions = []
    for ant in range(self.num_ants):
        solution = []
        visited = np.zeros(self.num_nodes, dtype=bool)
        current_node = np.random.randint( low: 0, self.num_nodes)
        solution.append(current_node)
        visited[current_node] = True
        current_energy = self.battery_capacity

        while len(solution) < self.num_nodes:
            visibility = 1 / self.distances[current_node]
            next_node = self._choose_next_node(self.pheromones[current_node], visibility, visited, current_energy)
            travel_cost = self.distances[current_node, next_node] * self.energy_consumption_per_unit

            if current_energy <= 0 and self.charging_point is not None:
                charging_cost = self.distances[current_node, self.charging_point] * self.energy_consumption_per_unit
                current_energy += self.battery_capacity - charging_cost
                current_node = self.charging_point
                solution.append(current_node)
            else:
                if current_energy - travel_cost >= 0:
                    solution.append(next_node)
                    visited[next_node] = True
                    current_node = next_node
                    current_energy -= travel_cost
                else:
                    solution.append(next_node)
                    visited[next_node] = True
                    current_node = next_node
                    current_energy = 0

        solutions.append(solution)
    return solutions
```

Рисунок 3.2 – Скріншот коду функції construct_solution

Функція що на рисунку 3.2 використовується для побудови поточних рішень для кожної мурахи, та обирається випадковий початковий вузол, після чого виконується ітеративний процес додавання наступних вузлів до маршруту до тих пір, поки всі вузли не будуть відвідані. Під час побудови маршруту враховується енергетичне обмеження, і якщо енергія закінчується, мураха може повернутися на

зарядне місце (якщо воно задане) тобто інтерпретуючи в задачу якщо електротранспорт буде розряджений, то він не зможе повернутися та маршрут закінчується (рис. 3.3).

```
1 usage
def _choose_next_node(self, pheromone, visibility, visited, current_energy):
    prob = (pheromone ** self.alpha) * (visibility ** self.beta)
    prob[visited] = 0
    total_prob = np.sum(prob)
    if total_prob == 0:
        return np.random.choice(np.where(visited == False)[0])
    prob = prob / total_prob
    return np.random.choice(range(self.num_nodes), p=prob)
```

Рисунок 3.3 – Скріншот коду функції _choose_next_node

Ця функція визначає наступний вузол для відвідування з урахуванням феромонів вже відвіданих вузлів і поточної енергії. Використовуються феромони та міра привабливості для обчислення ймовірності відвідання коженого доступного вузла, з урахуванням того, що відвідані вже вузли які привели до виснаження енергії, виключаються.

```
def _update_pheromones(self, solutions):
    self.pheromones *= self.decay
    for solution in solutions:
        distance = self._calculate_total_distance(solution)
        for i in range(len(solution) - 1):
            self.pheromones[solution[i], solution[i + 1]] += 1.0 / distance
            self.pheromones[solution[i + 1], solution[i]] += 1.0 / distance
```

Рисунок 3.4 – Скріншот коду функції _update_pheromones

Ця функція що зображена на рисунку 3.4 оновлює рівні феромонів на основі знайдених рішень. Кожен маршрут використовується для оновлення феромонів на кожному ребрі графа. Рівні феромонів зменшуються на кожній ітерації

```
def _calculate_total_distance(self, solution):
    distance = 0
    for i in range(len(solution) - 1):
        distance += self.distances[solution[i], solution[i + 1]]
    if len(solution) > 1:
        distance += self.distances[solution[-1], solution[0]]
    return distance
```

Рисунок 3.5 – Скріншот коду функції _update_pheromones

Ця функція що зображено на рисунку 3.5 обчислює загальну відстань для заданого маршруту, додаючи відстані між послідовними вузлами та відстань від останнього вузла до першого вузла.

```
def solve(self):
    if self.min_energy_for_return is None:
        self._set_min_energy_for_return()
    best_solution = None
    best_distance = float('inf')
    for iteration in range(self.num_iterations):
        solutions = self._construct_solution()
        valid_solutions = [sol for sol in solutions if len(sol) == self.num_nodes]
        if not valid_solutions:
            continue
        self._update_pheromones(valid_solutions)
        for solution in valid_solutions:
            distance = self._calculate_total_distance(solution)
            if distance < best_distance:
                best_distance = distance
                best_solution = solution
    return best_solution, best_distance  # Повертаємо тільки два значення
```

Рисунок 3.6 – Скріншот коду функції _ solve

Ця функція реалізує основний цикл алгоритму мурахи для знаходження найкращого маршруту. По завершенні кожної ітерації оновлюються феромони, а кращий маршрут зберігається для подальшого використання.

Далі було реалізовано ще один алгоримт який вірішується для задачи оптимізації маршутів а саме алгоримт A*.

3.2 Реалізація алгоритму A*

Завдяки своїй здатності знаходити оптимальні рішення з використанням інформованого пошуку, алгоритм A* є одним з найпопулярніших методів у задачах пошуку шляху і часто застосовується у реальних додатках, де потрібно швидко знайти найкоротший шлях з точки A до точки B.

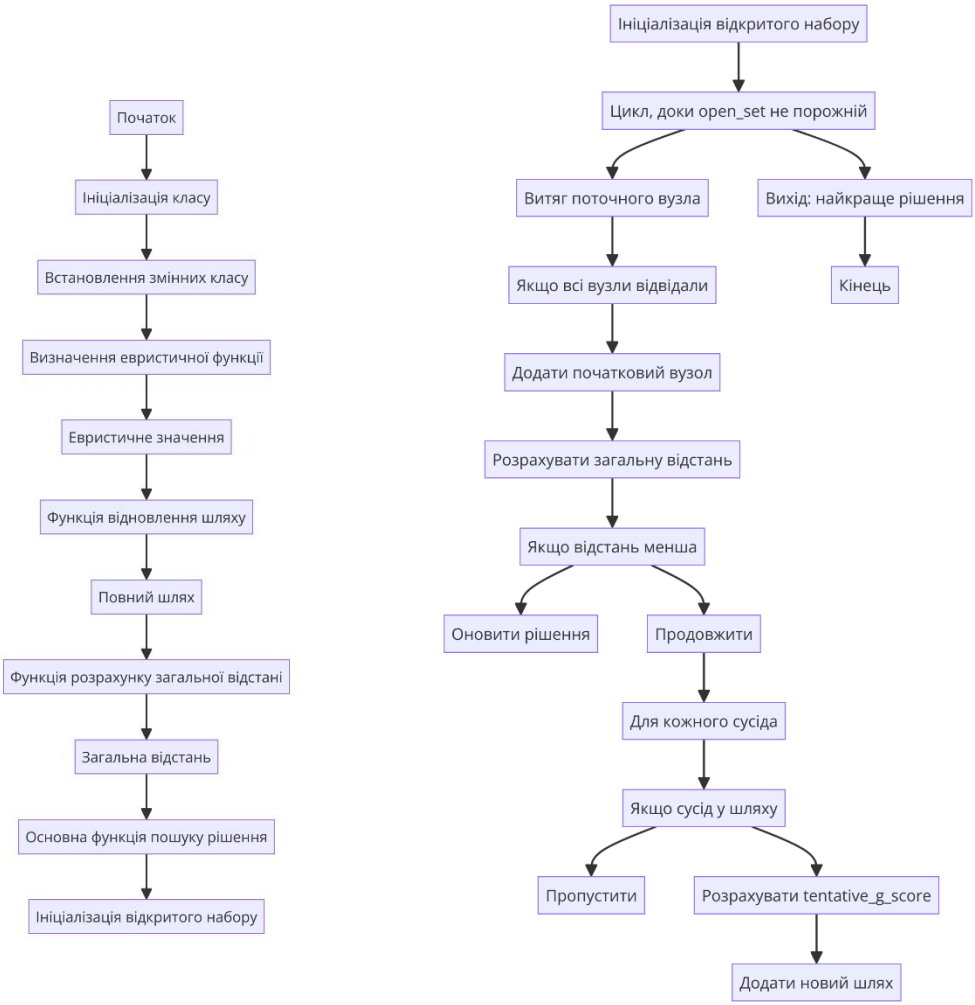

Рисунок 3.7 – Блок схема алгоритму A* [41]

Як видно з блок-схеми, як і минулого разу, так і цього разу було створено клас під назвою AStarOptimizationEV, що означає потенційне використання цього алгоритму у застосунку для оптимізації шляху.

Спочатку потрібно ініціалізувати клас і встановити змінні класу. Було необхідно визначити евристичну функцію та отримати евристичне значення. Далі створили функцію, головним завданням якої було відновлення шляху та отримання повного маршруту. Потім написали функцію для розрахунку загальної відстані.

Основна функція пошуку рішень ініціалізує відкритий набір, після чого запускається цикл, який триває, поки список шляху не стане порожнім. Якщо список порожній, виводиться найкращий шлях і алгоритм завершується. В іншому випадку, поточний вузол витягується і перевіряється, чи всі вузли вже відвідані. Якщо так, поточний вузол додається, і розраховується загальна відстань. Далі перевіряється, чи стала відстань меншою, і якщо так, оновлюється рішення, інакше алгоритм продовжується. Цей процес повторюється для кожного сусіда: якщо сусід у шляху, його пропускають; якщо ні, розраховується тимчасова оцінка найкоротшої відстані і додається новий шлях. Розглянемо окремі функції програмного коду алгоритму (рис. 3.8).

```
1 usage
def _heuristic(self, current_node, goal_node):
    return np.linalg.norm(self.coordinates[current_node] - self.coordinates[goal_node])
```

Рисунок 3.8 – Скріншот коду функції _heuristic

Функція *_heuristic* обчислює евристичну оцінку відстані між поточним вузлом і цільовим вузлом. Вона використовує норму (або евклідову відстань) для вимірювання прямої лінії між двома точками. Ця оцінка допомагає алгоритму A* орієнтувати пошук у напрямку цільового вузла, зменшуючи кількість оброблених вузлів та прискорюючи знаходження оптимального шляху.

```python
def _reconstruct_path(self, came_from, current):
    total_path = [current]
    while current in came_from:
        current = came_from[current]
        total_path.append(current)
    return total_path[::-1]
```

Рисунок 3.9 – Скріншот коду функції _reconstruct_path

Функція *_reconstruct_path* (рис. 3.9) відновлює найкращий шлях з кінцевого вузла до початкового, використовуючи словник *came_from*, який зберігає попередники кожного вузла. Вона починає з кінцевого вузла і проходить через всі вузли-попередники, додаючи їх до списку *total_path*. Після завершення проходження, список перевертається, щоб представити шлях у правильному порядку від початку до кінця.

```python
def _calculate_total_distance(self, path):
    distance = 0
    for i in range(len(path) - 1):
        distance += self.distances[path[i], path[i + 1]]
    distance += self.distances[path[-1], path[0]]  # Return to start
    return distance
```

Рисунок 3.10 – Скріншот коду функції _calculate_total_distance

Функція *_calculate_total_distance* (рис. 3.10) обчислює загальну відстань для заданого шляху. Вона проходить через всі пари сусідніх вузлів у списку path і сумує відстані між ними, використовуючи матрицю відстаней distances. Після обчислення відстаней між усіма вузлами на шляху, додається відстань від останнього вузла до початкового, щоб замкнути маршрут.

```
1 usage
def solve(self, start_node):
    open_set = []
    heapq.heappush(open_set,  _item: (0, start_node, [start_node], 0))
    best_solution = None
    best_distance = float('inf')

    while open_set:
        _, current, path, g_score = heapq.heappop(open_set)

        if len(path) == self.num_nodes:
            path.append(start_node)
            total_distance = self._calculate_total_distance(path)
            if total_distance < best_distance:
                best_solution = path
                best_distance = total_distance
            continue

        for neighbor in range(self.num_nodes):
            if neighbor in path:
                continue
            tentative_g_score = g_score + self.distances[current, neighbor]
            f_score = tentative_g_score + self._heuristic(neighbor, start_node)
            new_path = path + [neighbor]
            heapq.heappush(open_set,  _item: (f_score, neighbor, new_path, tentative_g_score))

    return best_solution, best_distance
```

Рисунок 3.11 – Скріншот коду функції _ solve

Функція *solve* (рис. 3.11) реалізує основний алгоритм пошуку шляхів, заснований на A*. Вона використовує чергу з пріоритетом *open_set*, щоб зберігати вузли для обробки. Кожен елемент черги містить значення *f_score*, поточний вузол, шлях до цього вузла, та накопичене значення *g_score*.

На кожному кроці вузол з найменшим *f_score* витягується для обробки. Якщо шлях охоплює всі вузли, він завершується і розраховується загальна відстань.

Якщо шлях краще за попередній найкращий, він зберігається як новий найкращий. Для кожного сусіднього вузла, що ще не в шляху, розраховується новий *g_score* і *f_score*, після чого новий шлях додається до черги для подальшої обробки. Функція повертає найкращий знайдений шлях та його відстань.

3.3 Вибір найкращого алгоритму

Після реалізації 2 алгоритмів потрібно було вибрати який із них краще, та буде інтерпретований в програмну реалізацію застосунку оптимізації маршрутів електротранспорту. Розгляне функціонал даної програми для аналізу алгоритмів.

Основна частина коду це функції для побудови різних графів але є декілька цікавих функції, проведемо їх огляд.

```
def generate_random_points(num_points, x_range, y_range):
    points = np.random.rand(num_points, 2)
    points[:, 0] *= x_range
    points[:, 1] *= y_range
    return points
```

Рисунок 3.12 – Скріншот коду функції generate_random_points

Функція *generate_random_points* (3.12) генерує випадкові точки у двовимірному просторі за вказаною кількістю та діапазоном координат. Вона приймає аргументи *num_points* (кількість точок), *x_range* (діапазон значень по осі X) та *y_range* (діапазон значень по осі Y). У цій функції використовується бібліотека *NumPy* для генерації випадкових чисел та обробки даних. Точки генеруються за допомогою np.random.rand, а потім масштабуються до вказаних діапазонів.

```python
def main():
    num_points = 8
    x_range = 100
    y_range = 100
    num_ants = 5
    num_iterations = 50
    decay = 0.5
    alpha = 1
    beta = 1
    battery_capacity = 100
    energy_consumption_per_unit = 1
    num_runs = 10  # Number of times to run A* for comparison

    points = generate_random_points(num_points, x_range, y_range)
    distances = calculate_distances(points)

    # Plot all routes
    plot_all_routes(points)

    # Ant Colony Optimization
    tracemalloc.start()
    start_time = time.time()
    aco = AntColonyOptimizationEV(
        coordinates=points,
        distances=distances,
        num_ants=num_ants,
        num_iterations=num_iterations,
        decay=decay,
        alpha=alpha,
        beta=beta,
        battery_capacity=battery_capacity,
        energy_consumption_per_unit=energy_consumption_per_unit
    )
```

Рисунок 3.13 – Скріншот коду функції main

Функція *main* (3.13) виконує основну роботу програми. Вона визначає параметри для алгоритмів Ant Colony Optimization (ACO) та A* Algorithm, генерує випадкові точки, обчислює відстані між ними, а потім викликає обидва алгоритми для пошуку найкращого маршруту. Після цього функція відображує знайдені маршрути за допомогою графіків і виводить метрики оцінки ефективності обох алгоритмів, такі як найкраща відстань, використання пам'яті та час виконання.

Для порівняння було обрано 8 випадкових точок

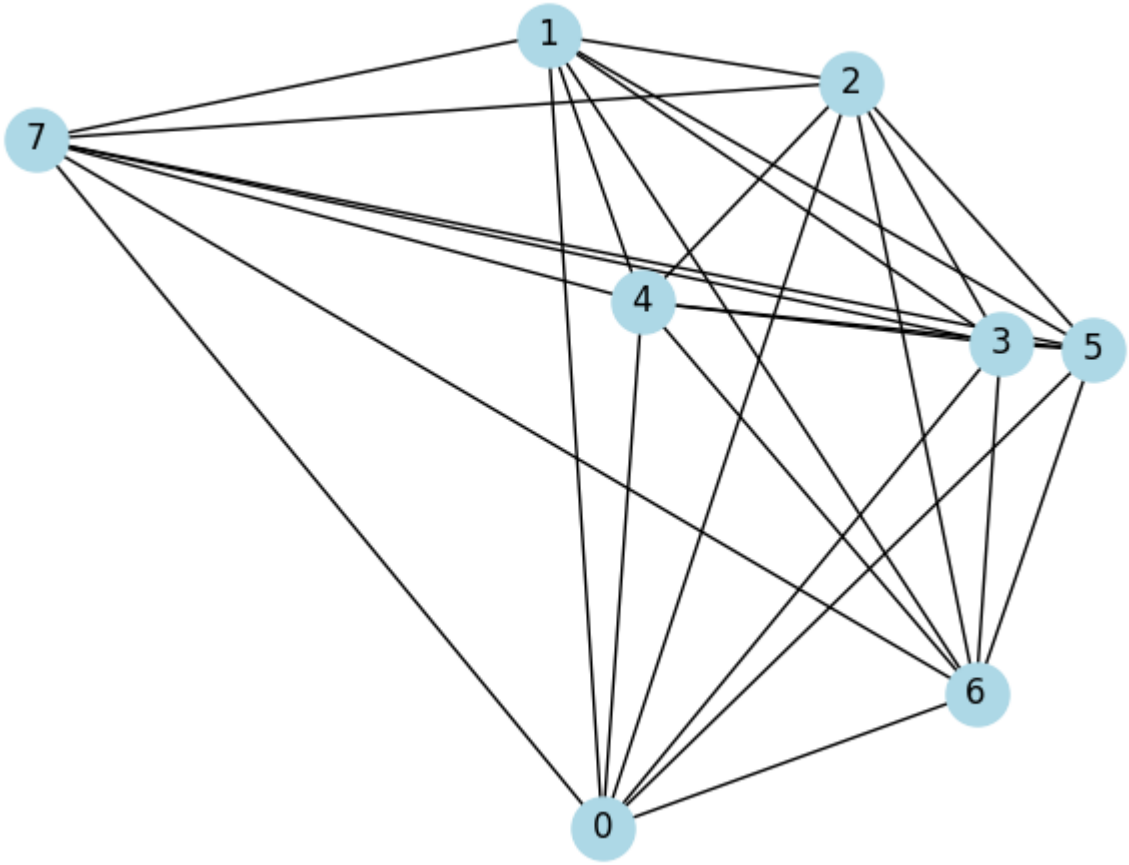

Рисунок 3.14 – Граф можливих маршрутів восьми випадкових точок

На цьому графіку (3.14) можна побачити розміщення згенерованих восьми випадкових точок, а з’єднані ребра відповідають за шляхи.

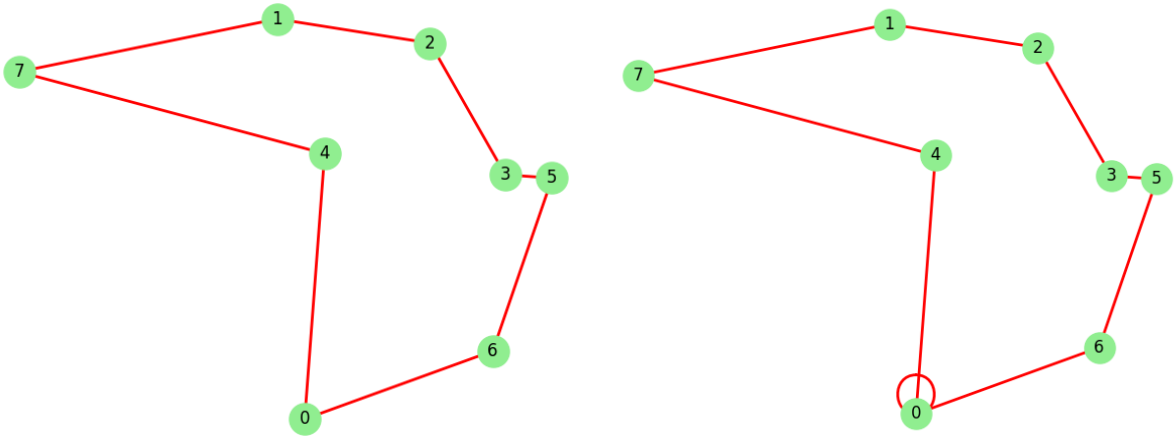

Рисунок 3.15 — Графи найкращого маршруту алгоритму алгоритму мурашиної колонії та A*

На даному графіку (рис. 3.15) можна побачити те що алгоритм * намалював коло в старті шляху це означає те що самого початку алгоритм не зміг вибрати точку до якої рухатися тому повернувся до першої точки та перерахував ребра.

```
Evaluation Metrics
ACO Best Distance: 279.38631983113044
A* Best Distances: 279.3863198311305
ACO Memory Usage: 91.0537109375 KB
A* Memory Usages: 75.70546875 KB
ACO Time: 0.38788676261901855 Seconds
A* Times: 0.5582975625991822 Seconds
```

Рисунок 3.16 – Результат метрик для двох алгоритмів із восьма випадковими точками

Як можна побачити за результатом (рис. 3.16) алгоритм A* менше витрачає оперативної пам’яті але більше витрачає часу на пошук найкращого шляху, а в мурашиного алгоритму навпаки більше використовує оперативної пам’яті але менше часу використовує на пошук найкращого шляху. Отже виходячи із результатів можна вибрати найкращий алгоритм, враховуючи те що задача застосунку швидко знаходити та оптимізувати шлях, з даним завдання добре справиться алгоритм мурашиної колонії адже він хоч і витрачає більше оперативної пам’яті але результат дає фактично без затримки.

3.4 Реалізація і тестування застосунку

Для реалізації застосунку оптимізації маршруту електротранспорту було вибрано мову програмування python та фреймворк flask. Flask - це легковагий веб-фреймворк для мови програмування Python, який дозволяє швидко створювати веб-додатки. Він має простий та зрозумілий синтаксис, що дозволяє швидко розпочати

розробку веб-додатків без зайвого навантаження. Flask базується на бібліотеці WSGI (Web Server Gateway Interface) і надає ряд корисних інструментів і бібліотек для створення веб-сайтів та веб-додатків, включаючи маршрутизацію URL, шаблони Jinja2 для HTML-сторінок, вбудований сервер розробки та підтримку розширень для розширення функціоналу за необхідності.

Front-end був написаний за допомогою мови програмування JavaScript, мовою гіпертекстовою розмітки HTML, та мова, що використовується для опису вигляду та форматування веб-сторінок CSS (рис. 3.17).

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-076-910
```

Рисунок 3.17 – Запуск застосунку

Для реалізації застосунку спочатку потрібно було написати зовнішній вигляд застосунку. Після запуску застосунку можна побачити в консольному режимі виведено посилання на локальну адресу веб застосунку. Розглянемо зовнішній вигляд застосунку (рис. 3.18).

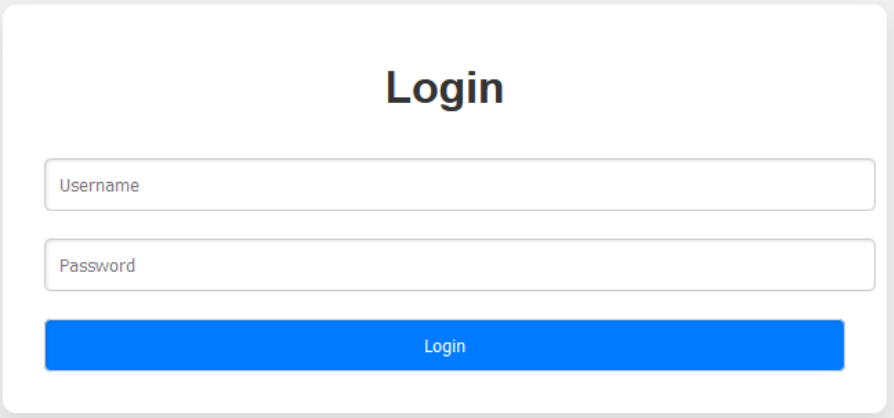

Рисунок 3.18 – Сторінка авторизації

Так як в перспективі, цім веб застосунком можуть користуватися компанії перевезень наприклад посилок або людей. Було додано авторизацію Поки що існує тільки один акаунт але при можливості можна додати базу даних де будуть зберігатися логіни та паролі, форму реєстрації не було надано щоб могли користуватися нею тільки компанії, а не всі охочі користувачі.

Рисунок 3.19 – Скріншот інтерфейсу застосунку

Застосунок поділений на дві частини (рис. 3.19) перша це поле що в собі включає поля для воду адреси та імені водія або номера транспорту. Та декількох

кнопок а саме додати , видалити водія, та додати , видалити адресу, підрахувати шлях, та вийти із системи. В другу частину входить карта світу.

Перед ти як розглядати роботу застосунку потрібно розглянути back-end застосунку (рис. 3.20).

```python
# Mock user database
users = {'admin': {'password': 'admin'}}

2 usages
class User(UserMixin):
    def __init__(self, username):
        self.id = username

@login_manager.user_loader
def load_user(user_id):
    return User(user_id)

@app.route( rule: '/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            return redirect(url_for('index'))
        else:
            return 'Invalid credentials'
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))
```

Рисунок 3.20 – Скріншот back-end коду авторизації

Як можна побачити цей код використовує фреймворк Flask для створення простої системи аутентифікації користувачів. було створено Клас *User* якого є головна задача встановити ідентифікатор користувача у функції *load_user* задача цієї функції завантажити користувача з ідентифікатором *user_id*. Також було створено маршрут *login* він обробляє GET та POST запити для сторінки входу, та маршрут *logout* цей маршрут відповідає за вихід користувача із системи.

```python
@app.route('/')
@login_required
def index():
    return render_template('index.html')

@app.route( rule: '/add_point', methods=['POST'])
@login_required
def add_point():
    address = request.json.get('address')
    coordinates = google_maps_api.get_coordinates(address)
    if coordinates:
        return jsonify({'address': address, 'coordinates': coordinates}), 200
    else:
        return jsonify({'error': 'Invalid address'}), 400

@app.route( rule: '/optimize_route', methods=['POST'])
```

Рисунок 3.21 – Скріншот back-end головної сторінки

функція index (рис. 3.21) відповідає за обробку запиту до кореневого маршруту. Декоратор *login_required* гарантує, що тільки автеризовані користувачі мають доступ до цієї сторінки. Це означає, що користувач повинен бути увійти в систему, інакше його перенаправлять на сторінку входу. У функції використовується *render_template*, щоб повернути HTML-сторінку index.html як відповідь.

Функція add_point обробляє POST-запити до маршруту *add_point*. Знову ж таки, Ця функція приймає JSON-дані, які містять адресу. Потім використовується клас *google_maps_api* в якому прописана логіка взаємодії з google API, в цьому класі реалізована функція *get_coordinates* для отримання координати адрес які були введені користувачем. Якщо координати знайдені, функція повертає JSON-відповідь з адресою та координатами і HTTP-статусом 200. Якщо адреса невірна або не вдалося знайти координати, повертається повідомлення про помилку і статус 400.

```python
def optimize_route():
    points = request.json.get('points')
    addresses = [point['address'] for point in points]
    coordinates = [tuple(point['coordinates']) for point in points]

    if len(coordinates) < 2:
        return jsonify({'error': 'Not enough points for optimization'}), 400

    distance_matrix = google_maps_api.get_distance_matrix(coordinates, coordinates)
    if distance_matrix is None:
        return jsonify({'error': 'Error getting distance matrix'}), 500

    num_ants = 10
    num_iterations = 100
    decay = 0.5

    aco = AntColonyOptimizationEV(coordinates, distance_matrix, num_ants, num_iterations, decay)
    best_solution, best_cost = aco.solve()  # Expect only two values

    routes = []
    for i in range(len(best_solution) - 1):
        route = google_maps_api.get_route(coordinates[best_solution[i]], coordinates[best_solution[i + 1]])
        if route:
            routes.extend(route)

    final_route = google_maps_api.get_route(coordinates[best_solution[-1]], coordinates[best_solution[0]])
    if final_route:
        routes.extend(final_route)

    formatted_routes = [{'lat': step[0], 'lng': step[1]} for step in routes]

    return jsonify({'routes': formatted_routes, 'addresses': addresses}), 200
```

Рисунок 3.22 – Скріншот back-end оптимізації маршруту

Функція *optimize_route* (рис. 3.22) реалізує процес оптимізації маршруту для заданих точок за допомогою алгоритму оптимізації на основі колонії мурах На початку функція отримує список точок з запиту у форматі JSON. З кожної точки вилучаються адреси та координати, щоб далі працювати з ними окремо. Якщо кількість координат менше двох, повертається повідомлення про помилку з відповідним статусом 400, оскільки оптимізація маршруту потребує принаймні двох точок.

Наступним кроком є отримання матриці відстаней між усіма парами точок за допомогою Google Maps API. Якщо запит на отримання матриці відстаней не вдається, повертається помилка зі статусом 500. Далі задаються параметри для

алгоритму оптимізації: кількість мурах, кількість ітерацій, та коефіцієнт випаровування феромонів.

Алгоритм оптимізації колонії мурах використовується для знаходження найкращого маршруту, що мінімізує загальну відстань. Результатом виконання алгоритму є найкраще рішення (порядок відвідування точок) та його вартість (загальна довжина маршруту). Потім для кожної пари послідовних точок у знайденому маршруті запитується маршрут за допомогою Google Maps API і додається до списку маршрутів. Додатково отримується маршрут від останньої точки назад до першої для замикання маршруту в кільце.

Отримані маршрути форматуються у вигляді списку координат (широта і довгота) для кожного кроку маршруту. Наприкінці функція повертає JSON-об'єкт, що містить відформатовані маршрути та адреси початкових точок, з відповідним статусом 200. Далі потрібно розлянути реалізацію ключових фукції для роботи з Google API (3.23).

```
def get_coordinates(self, address):
    try:
        geocode_result = self.gmaps.geocode(address)
        if geocode_result:
            location = geocode_result[0]['geometry']['location']
            return location['lat'], location['lng']
        else:
            return None
    except Exception as e:
        print("Error getting coordinates:", e)
        return None
```

Рисунок 3.23 – Скріншот функції отримання координат із адреси

Метод *get_coordinates* приймає адресу як вхідний параметр і використовує API геокодування Google Maps для отримання координат (широти та довготи) цієї адреси. Спочатку він намагається отримати результати геокодування за допомогою

методу geocode. Якщо результат не порожній, він витягує координати з першого результату в списку, отриманого від API, і повертає широту та довготу. Якщо ж результат порожній, метод повертає None. У разі виникнення будь-якої помилки під час запиту, вона перехоплюється, друкується повідомлення про помилку, і метод також повертає None.

```
1 usage
def get_distance_matrix(self, origins, destinations):
    try:
        distance_matrix = self.gmaps.distance_matrix(origins, destinations, mode='driving')
        distances = np.array([[element['distance']['value'] / 1000.0 if element['distance']['value'] > 0 else np.inf
                               for element in row['elements']]
                              for row in distance_matrix['rows']])
        return distances
    except Exception as e:
        print("Error getting distance matrix:", e)
        return None
```

Рисунок 3.24 – Скріншот функції отримання матриці дистанції

Метод *get_distance_matrix* (рис. 3.24) приймає два списки: origins і destinations, які містять початкові та кінцеві точки, відповідно. Він використовує API Google Maps для отримання матриці відстаней між цими точками. За допомогою методу distance_matrix, він отримує результати для водіння. Результуюча матриця відстаней зберігається у вигляді двовимірного масиву NumPy, де кожен елемент відповідає відстані між парою точок у кілометрах. Якщо відстань не визначена або недоступна, встановлюється значення np.inf. У разі виникнення помилки метод перехоплює її, виводить повідомлення про помилку та повертає None.

```
2 usages
def get_route(self, origin, destination):
    try:
        directions_result = self.gmaps.directions(origin, destination, mode='driving')
        if directions_result:
            route = directions_result[0]['overview_polyline']['points']
            points = googlemaps.convert.decode_polyline(route)
            return [(point['lat'], point['lng']) for point in points]
        else:
            return None
    except Exception as e:
        print("Error getting route:", e)
        return None
```

Рисунок 3.25 – Скріншот функції отримання маршруту

Метод *get_route* (рис. 3.25) використовує API напрямків Google Maps для отримання маршруту між цими двома точками. Він викликає метод directions, щоб отримати маршрут для водіїв. Якщо результат успішний, метод витягує з нього полілінію маршруту, яка кодується у вигляді рядка. Потім ця полілінія декодується за допомогою decode_polyline у список точок з координатами (широта і довгота). Цей список точок повертається як результат роботи методу. Якщо маршрут не знайдений або виникає помилка, метод повертає None після виведення відповідного повідомлення про помилку.

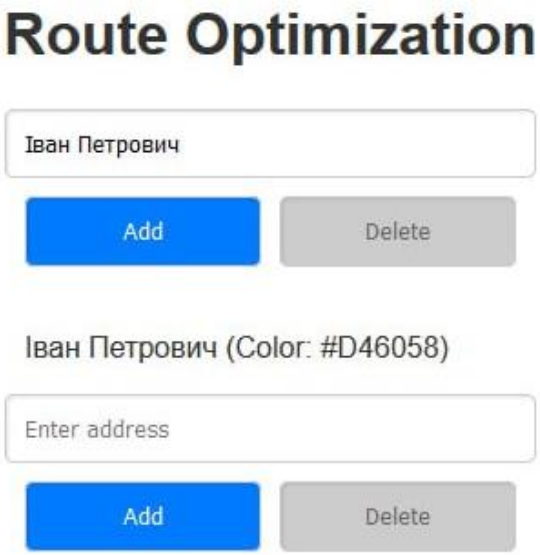

Рисунок 3.26 – Скріншот результату додавання імені водія

Як можна побачити на скріншоті (рис. 3.26) після введення імені водія з’явилася можливість давати адресу але якщо писати адресу прямо зараз буде повідомлення що не було обрано водія в списку

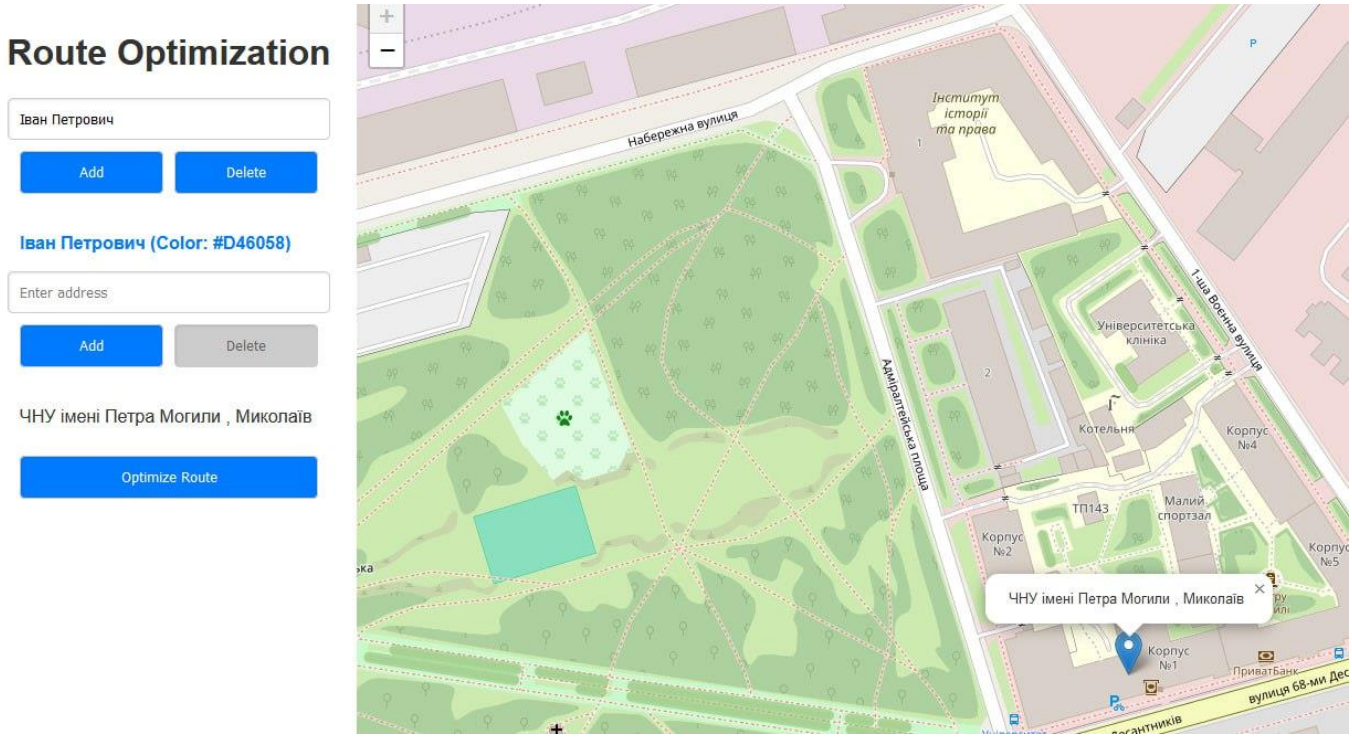

Рисунок 3.27 – Скріншот результату додавання адреси

Як можна побачити на рисунку 3.27 після того як зробили активним водія з’явилася можливість додати адресу, після додавання адреси з’являється маркер на карті з назвою точки та точним місце знаходження цієї точки.

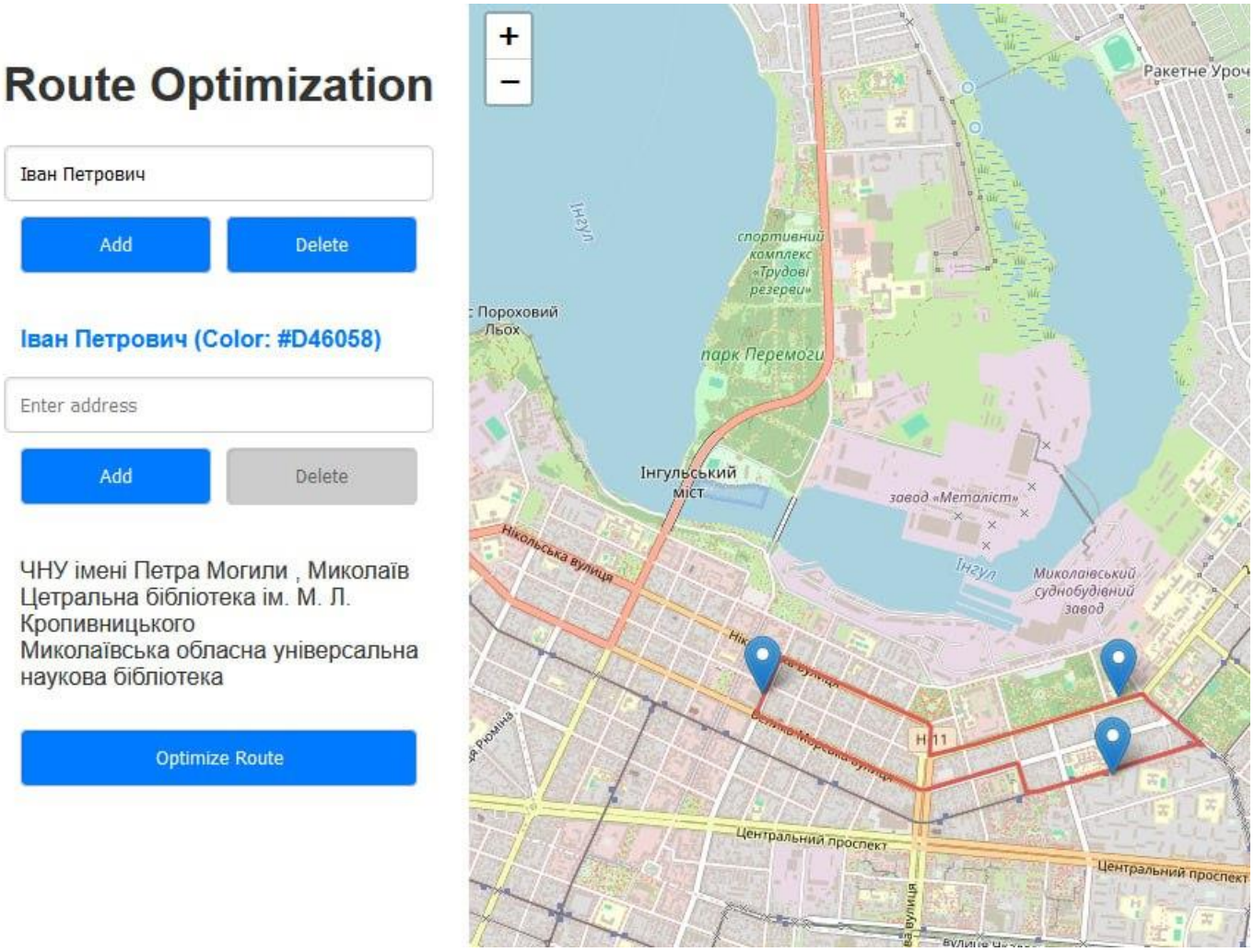

Рисунок 3.28 – Скріншот результату побудови маршруту

Як можна побачити на скріншоті (рис. 3.28) після додавання списку адреси, можна оптимізувати маршрут, після натискання на кнопку за допомогою мурашиного алгоритму знаходиться найкращий маршрут та будує його на мапі.

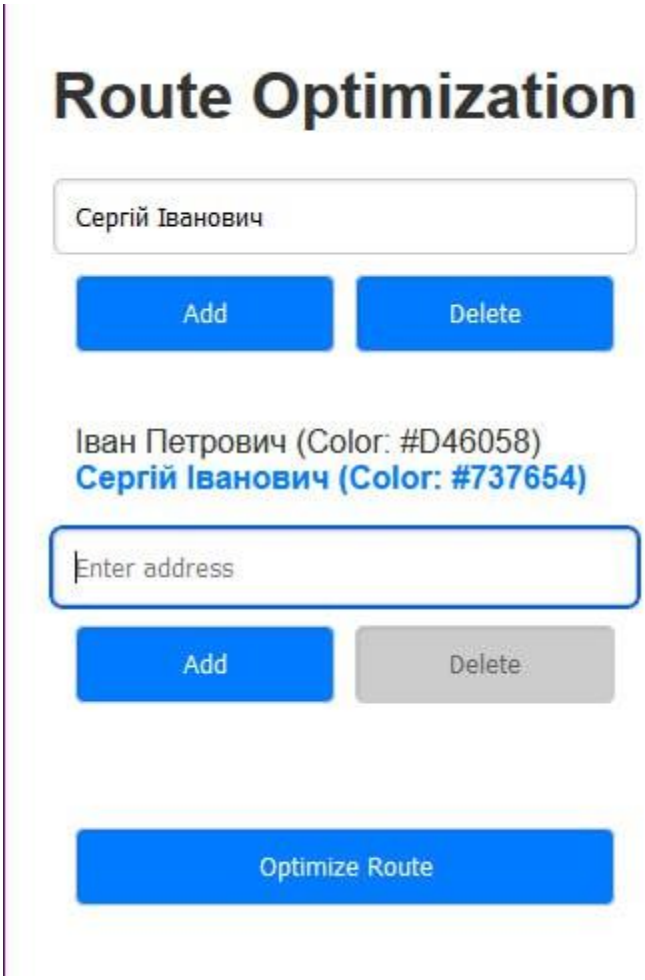

Рисунок 3.29 – Скріншот результату додавання ще одного водія

Також в застосунку (рис. 3.29) додана можливість додати водія, при тому якщо зробити активним нового водія із списку візуально зникають адреси першого водія, це було зроблено щоб користувачі не змогли розібрати яка адреса закріплена за водієм.

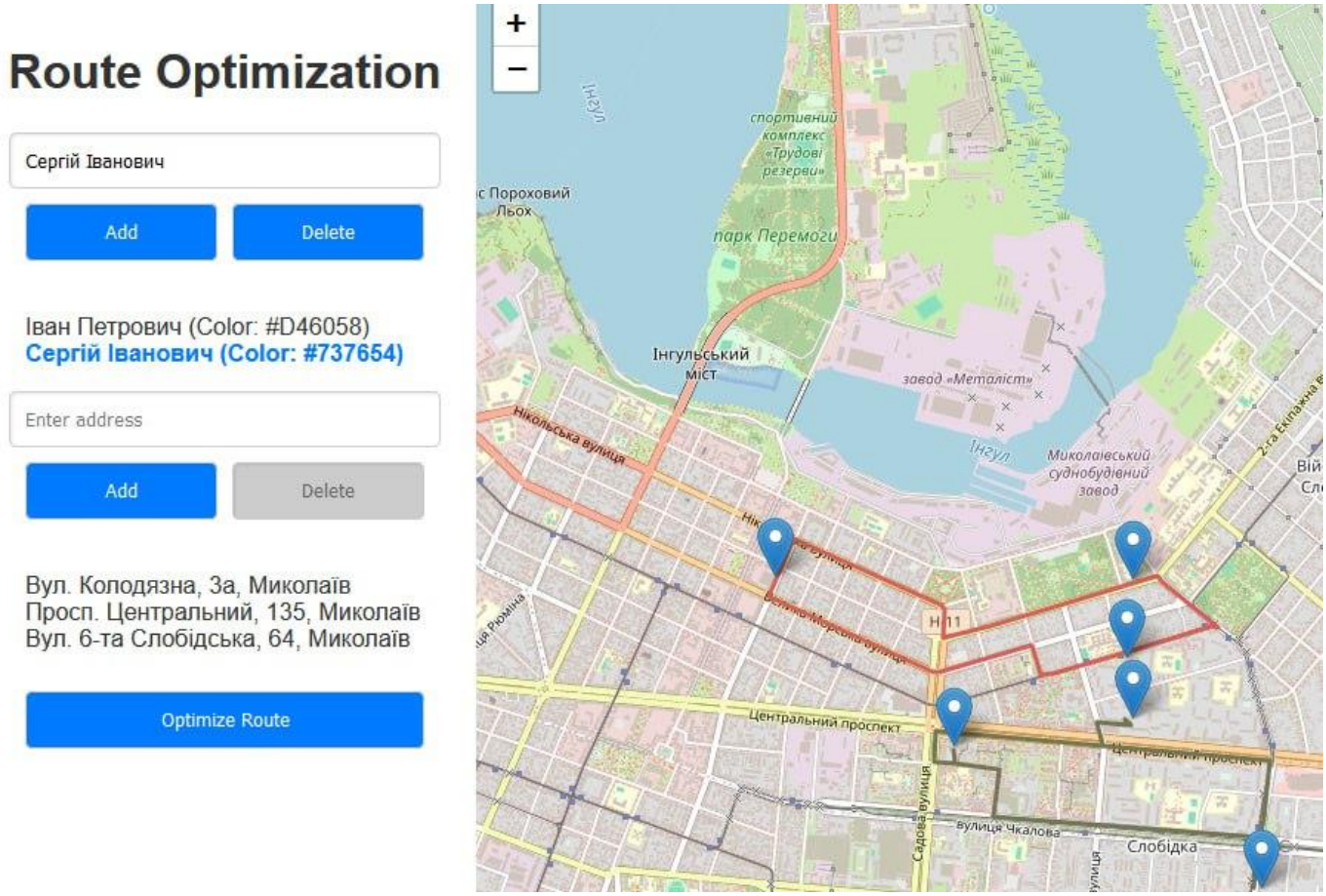

Рисунок 3.30 – Скріншот результату оптимізації маршруту ще для одного водія

Після того як додали нові адреси до нового водія та натиснули на кнопку для оптимізації маршруту (рис. 3.30), на мапу було додано новий маршрут це було зроблено для того щоб компанія змогла занести декілька водіїв на карту та оптимізувати їх маршрут в режимі реального часу.

Висновки до розділу 3

У процесі вибору найкращого алгоритму для оптимізації маршрутів електротранспорту було реалізовано та протестовано два алгоритми: Ant Colony Optimization (ACO) та A* Algorithm. Кожен з них мав свої переваги та недоліки, які було ретельно проаналізовано.

Ефективність пам'яті та час виконання обох алгоритмів значно відрізнялися. Алгоритм A* демонстрував менше споживання оперативної пам'яті, однак потребував більше часу для пошуку найкращого маршруту. Натомість, алгоритм мурашиної колонії (ACO) витрачав більше оперативної пам'яті, але значно швидше знаходив оптимальний маршрут. Це є важливим фактором, враховуючи специфіку завдання, де швидкість є критично важливою.

При побудові маршрутів алгоритм A* іноді на початку повертався до стартової точки для перерахунку шляхів, що могло затримувати процес оптимізації. Водночас, ACO з самого початку ефективніше обирав напрямки, що дозволяло швидше досягти результатів. Враховуючи поставлену задачу — швидко знаходити та оптимізувати маршрути, оптимальним вибором є алгоритм мурашиної колонії. Хоча він вимагає більше ресурсів пам'яті, проте забезпечує швидке знаходження найкращого маршруту, що є критичним для роботи застосунку в реальних умовах.

У процесі реалізації було розроблено програму, яка включає декілька ключових функцій, що забезпечують ефективну оптимізацію маршрутів електротранспорту. Генерація випадкових точок здійснюється за допомогою функції generate_random_points, яка створює випадкові точки у двовимірному просторі. Це є основою для подальшого аналізу алгоритмів.

Основна функція main відповідає за визначення параметрів для алгоритмів Ant Colony Optimization (ACO) та A* Algorithm, генерацію точок, обчислення відстаней між ними та виклик обох алгоритмів для пошуку найкращого маршруту. Після цього здійснюється візуалізація знайдених маршрутів за допомогою графіків, а також виведення метрик оцінки ефективності обох алгоритмів, таких як найкраща відстань, використання пам'яті та час виконання.

Реалізація застосунку також забезпечує високий рівень інтерактивності. Користувачі можуть вводити імена водіїв, адреси, оптимізувати маршрути та візуалізувати їх на карті. Ці функції забезпечують зручність використання та

ефективне управління маршрутами, що є важливим для компаній, які займаються перевезеннями.

Таким чином, реалізований застосунок ефективно вирішує задачу оптимізації маршрутів електротранспорту, надаючи користувачам швидкі та зручні інструменти для управління та оптимізації маршрутів. Застосування мови програмування Python та фреймворку Flask забезпечує гнучкість та надійність розробки, а використання сучасних веб-технологій, таких як JavaScript, HTML та CSS, дозволяє створювати зручний та функціональний інтерфейс користувача.

ВИСНОВКИ

Створення застосунку для планування маршрутів електротранспорту на основі інтелектуальних технологій має ряд потенційних переваг, як для користувачів, так і для операторів електротранспорту. Користувачі матимуть доступ до більш точних та зручних маршрутів, а оператори електротранспорту зможуть покращити ефективність своїх мереж.

Під час виконання роботи було проведено детальний аналіз існуючих методів пошуку оптимальних маршрутів, що дозволило виділити їх переваги та недоліки. Алгоритм A* є потужним інструментом для знаходження оптимальних шляхів, проте він може бути неефективним у випадках, коли існує кілька локальних рішень з однаковою евристикою. Це може призводити до збільшення часу виконання задачі. Метод гілок і меж відкидає підмножини допустимих рішень, що ускладнює процес пошуку і збільшує час виконання. Метод найближчого сусіда є простим у реалізації, але має велику похибку, що робить його неприйнятним для більшості практичних задач. Алгоритми пошуку в ширину та глибину обмежені застосуванням до певних типів задач, що скорочує їх універсальність. Алгоритм Дейкстри ефективний для знаходження найкоротших шляхів у графах з позитивною вагою ребер, але не підходить для графів з негативними вагами. Алгоритм Джонсона здатний знаходити найкоротші шляхи між усіма парами вершин, але має обмеження щодо графів без циклів з негативною вагою і потребує значних ресурсів для обчислення.

Мурашиний алгоритм ефективно імітує природні принципи поведінки мурах, використовуючи колективну динаміку для знаходження оптимальних рішень. Завдяки адаптації до змінних умов та динамічному оновленню феромонів, вони забезпечують надійні та ефективні рішення.

Формування початкової популяції мурах є критично важливим етапом, оскільки від правильної ініціалізації залежить швидкість збіжності алгоритму та

його здатність уникати локальних мінімумів.

Мурашині алгоритми продемонстрували високу ефективність у розв'язанні широкого спектра задач оптимізації, включаючи задачі комівояжера, маршрутизації транспортних засобів та кластеризації даних .

В рамках роботи було досліджено та реалізовано два методи оптимізації: класичний мурашиний алгоритм та алгоритм A*. Їхні основні переваги та недоліки були визначені, що дозволило створити програмне забезпечення, в якому ці методи були реалізовані.

Розроблене програмне забезпечення дозволяє ефективно планувати маршрути для електротранспорту, підвищуючи його ефективність та зручність для користувачів

Таким чином, проведене дослідження показало, що мурашині алгоритми є потужним інструментом для розв'язання складних задач оптимізації маршрутів. Вони забезпечують ефективні та надійні рішення завдяки використанню колективної поведінки та адаптивних механізмів оновлення феромонів. Розроблене програмне забезпечення дозволяє застосовувати ці алгоритми на практиці, підвищуючи ефективність роботи електротранспорту.

Отже під час виконання роботи було досягнено мети, а саме розробка програмної системи побудови маршрутів перевезень електротранспорту з використанням мурашиного алгоритму.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Левітін А. Алгоритми. Введення в розробку та аналіз 2016. – 453 с.
2. Малихіна М.П. Бази даних: бази проектування використання: навчальний посібник. - 2-ге вид. - СПб.: БХВ-Петербург, 2016. - 528 с.
3. Павленко А.І., Титов Ю.П. Метод мурашиної оптимізації у завданнях розподілу ресурсів – МАІ, 2018.
4. Суботін С.А., Олійник Ан.А., Олійник Ал.А. Інтелектуальні мультиагентні методи (swarm intelligence).
5. Прахов Н.С. Основи програмування мовою Python. – Київ: Либідь, 2017. – 320 с.
6. Жук П.М. Алгоритми та структури даних. – Львів: ЛНУ, 2015. – 412 с.
7. Мілевський І.В. Основи розробки веб-застосунків. – Харків: ХНУРЕ, 2019. – 289 с.
8. Черненко О.С. Розподілені системи: теорія і практика. – Одеса: ОНУ, 2018. – 355 с.
9. Глушков В.М. Основи кібернетики. – Київ: Наукова думка, 2016. – 290 с.
10. Тема 6. Розподільча логістика: навчальний матеріал. URL: https://mk.nmu.org.ua/en/source/Logisticl6.pdf (дата звернення: 02.06.2024).
11. Ковальчук Л.М. Інтелектуальні системи. – Дніпро: ДНУ, 2019. – 314 с.
12. Матвєєв В.В. Обчислювальна техніка і програмування. – Суми: СумДУ, 2015. – 372 с.
13. Сидоренко М.П. Методи оптимізації. – Київ: КПІ, 2018. – 296 с.
14. Борисенко І.В. Комп’ютерні мережі. – Львів: ЛП, 2016. – 275 с.
15. Кравчук Д.В. Основи штучного інтелекту. – Чернівці: ЧНУ, 2019. – 318 с.
16. Петров В.А. Математичне моделювання. – Івано-Франківськ: ІФНТУНГ, 2017. – 267 с.
17. Гриценко С.І. Теорія ймовірностей і математична статистика. – Ужгород:

УжНУ, 2018. – 249 с.

18. Кучма Р.М. Основи обчислювальної техніки. – Луцьк: ЛНТУ, 2017. – 301 с.

19. Конспект лекції № 6 Тема № 6. Транспортна задача: навчальний матеріал. URL: https://financial.lnu.edu.ua/wp-content/uploads/2019/09/ME-lektsiia-6.pdf (дата звернення: 02.06.2024).

20. Федоренко О.В. Основи дискретної математики. – Запоріжжя: ЗНУ, 2016. – 278 с.

21. ДСТУ 2873-94. Системи оброблення інформації. Програмування. Терміни та визначення.

22. Соколов В.О. Об'єктно-орієнтоване програмування. – Київ: КНЕУ, 2018. – 326 с.

23. Кондратенко Ю. П. Оптимізація процесів прийняття рішень в умовах невизначеності. Методичні вказівки до курсового проектування з дисципліни «Теорія прийняття рішень». Миколаїв : ЧДУ ім. Петра Могили, 2005. 46 с.

24. Шевченко П.С. Комп’ютерна графіка. – Донецьк: ДНТУ, 2019. – 299 с.

25. Тарасенко І.О. Розробка програмного забезпечення. – Рівне: РДГУ, 2018. – 321 с.

26. Бондаренко М.П. Інформаційні системи. – Миколаїв: ЧНУ, 2016. – 298 с.

27. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ, 2016. 26 с. URL: http://www.knmu.kharkov.ua/attachments/3659_3008-2015.PDF (дата звернення: 31.05.2024).

28. Тимошенко В.Ю. Алгоритмічні структури. – Полтава: ПНТУ, 2017. – 269 с.

29. Мартиненко С.О. Комп'ютерна інженерія. – Вінниця: ВНТУ, 2018. – 358 с.

30. Поляков О.М. Методи штучного інтелекту. – Дніпро: ДНУ, 2016. – 305 с.

31. Ключка Т. А., Шаповалов С. П., Довбиш А. С. Комп’ютерний порівняльний аналіз алгоритмів Дініца та Форда-Фалкерсона: випускна робота. Суми, 2020. 44 с. URL: https://essuir.sumdu.edu.ua/bitstream-download/123456789/80062/1/Kluthka_bac_rob.pdf (дата звернення: 29.05.2024).

32. Семенов О.О. Комп'ютерні системи. – Суми: СумДУ, 2018. – 299 с.

33. Кузьменко О.В. Обчислювальні методи. – Київ: КНУ, 2016. – 278 с.

34. Сапронов С.М. Математичні основи інформатики. – Запоріжжя: ЗНУ, 2017. – 290 с.

35. Duan H. B. The Principle and Application of Ant Colony Algorithm: Science Press. Beijing, 2005. 16 с.

36. Корнієнко В.М. Інформаційні технології. – Донецьк: ДНТУ, 2019. – 283 с.

37. Воронін С.І. Розробка алгоритмів. – Тернопіль: ТНТУ, 2016. – 325 с.

38. Zhong X. H. On the approximation ratio of the 3-Opt algorithm for the (1,2)-TSP, Oper. Res. Lett., 2021. 521 с. URL: https://doi.org/10.1016/j.orl.2021.05.012 (дата звернення: 01.06.2024)

39. Гладченко В.П. Об'єктно-орієнтоване програмування на Java. – Івано-Франківськ: ІФНТУНГ, 2017. – 308 с.

40. Андрущенко О.М. Алгоритми і структури даних. – Рівне: РДГУ, 2018. – 290 с.

41. Петров В.І. Розробка інформаційних систем. – Миколаїв: ЧНУ, 2016. – 296 с.

42. Ковальчук В.П. Комп’ютерні мережі. – Полтава: ПНТУ, 2018. – 294 с.

43. Wolpert D. H., Macready W. G. No free lunch theorems for optimization, IEEE Trans. Evol. Comput., 1997. 82 с. URL: https://doi.org/10.1109/4235.585893 (дата звернення: 02.06.2024)

44. Шевченко О.М. Інформаційні системи. – Вінниця: ВНТУ, 2019. – 307 с.

45. Борисенко П.М. Основи програмування. – Львів: ЛП, 2016. – 285 с.

46. ТРАНСПОРТНА ЛОГІСТИКА, Сутність і завдання транспортної логістики, Види транспорту - Логістика - Навчальні матеріали онлайн: вебсайт. URL: https://pidru4niki.com/18800413/ekonomika/transportna_logistika (дата звернення: 30,05,2024).

47. Кравчук А.В. Комп'ютерні технології. – Чернівці: ЧНУ, 2017. – 298 с.

48. Іванов В.А. Алгоритмічні структури. – Дніпро: ДНУ, 2018. – 313 с.

ДОДАТОК А

Код програмної реалізації

GoogleMapsAPI.py

```
import googlemaps
import numpy as np
class GoogleMapsAPI:
    def __init__(self, api_key):
        self.api_key = api_key
        self.gmaps = googlemaps.Client(key=api_key)

    def get_coordinates(self, address):
        try:
            geocode_result = self.gmaps.geocode(address)
            if geocode_result:
                location = geocode_result[0]['geometry']['location']
                return location['lat'], location['lng']
            else:
                return None
        except Exception as e:
            print("Error getting coordinates:", e)
            return None

    def get_distance_matrix(self, origins, destinations):
        try:
            distance_matrix = self.gmaps.distance_matrix(origins, destinations,
mode='driving')
            distances = np.array([[element['distance']['value'] / 1000.0 if
element['distance']['value'] > 0 else np.inf
                                   for element in row['elements']]
                                  for row in distance_matrix['rows']])
            return distances
        except Exception as e:
            print("Error getting distance matrix:", e)
            return None

    def get_route(self, origin, destination):
        try:
            directions_result = self.gmaps.directions(origin, destination, mode='driving')
```

```
            if directions_result:
                route = directions_result[0]['overview_polyline']['points']
                points = googlemaps.convert.decode_polyline(route)
                return [(point['lat'], point['lng']) for point in points]
            else:
                return None
        except Exception as e:
            print("Error getting route:", e)
            return None
```

AntAlgorithm.py

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

class AntColonyOptimizationEV:
    def __init__(self, coordinates, distances, num_ants, num_iterations, decay, alpha=1,
beta=1, battery_capacity=100, energy_consumption_per_unit=1, charging_point=None):
        self.coordinates = coordinates
        self.distances = distances
        self.num_ants = num_ants
        self.num_iterations = num_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta
        self.num_nodes = len(coordinates)
        self.pheromones = np.ones((self.num_nodes, self.num_nodes)) / self.num_nodes
        self.battery_capacity = battery_capacity
        self.energy_consumption_per_unit = energy_consumption_per_unit
        self.charging_point = charging_point
        self.min_energy_for_return = None

    def _calculate_average_energy_consumption(self):
        total_energy_consumption = 0
        total_steps = 0
        for i in range(self.num_nodes):
            for j in range(self.num_nodes):
                total_energy_consumption += self.distances[i, j] *
```

```
self.energy_consumption_per_unit
                total_steps += 1
        return total_energy_consumption / total_steps

    def _set_min_energy_for_return(self):
        average_energy_consumption = self._calculate_average_energy_consumption()
        self.min_energy_for_return = 0.8 * average_energy_consumption

    def _construct_solution(self):
        solutions = []
        for ant in range(self.num_ants):
            solution = []
            visited = np.zeros(self.num_nodes, dtype=bool)
            current_node = np.random.randint(0, self.num_nodes)
            solution.append(current_node)
            visited[current_node] = True
            current_energy = self.battery_capacity

            while len(solution) < self.num_nodes:
                visibility = 1 / self.distances[current_node]
                next_node = self._choose_next_node(self.pheromones[current_node],
visibility, visited, current_energy)
                travel_cost = self.distances[current_node, next_node] *
self.energy_consumption_per_unit

                if current_energy <= 0 and self.charging_point is not None:
                    charging_cost = self.distances[current_node, self.charging_point] *
self.energy_consumption_per_unit
                    current_energy += self.battery_capacity - charging_cost
                    current_node = self.charging_point
                    solution.append(current_node)
                else:
                    if current_energy - travel_cost >= 0:
                        solution.append(next_node)
                        visited[next_node] = True
                        current_node = next_node
                        current_energy -= travel_cost
                    else:
                        solution.append(next_node)
```

```
                    visited[next_node] = True
                    current_node = next_node
                    current_energy = 0

            solutions.append(solution)
        return solutions

    def _choose_next_node(self, pheromone, visibility, visited, current_energy):
        prob = (pheromone ** self.alpha) * (visibility ** self.beta)
        prob[visited] = 0
        total_prob = np.sum(prob)
        if total_prob == 0:
            return np.random.choice(np.where(visited == False)[0])
        prob = prob / total_prob
        return np.random.choice(range(self.num_nodes), p=prob)

    def _update_pheromones(self, solutions):
        self.pheromones *= self.decay
        for solution in solutions:
            distance = self._calculate_total_distance(solution)
            for i in range(len(solution) - 1):
                self.pheromones[solution[i], solution[i + 1]] += 1.0 / distance
                self.pheromones[solution[i + 1], solution[i]] += 1.0 / distance

    def _calculate_total_distance(self, solution):
        distance = 0
        for i in range(len(solution) - 1):
            distance += self.distances[solution[i], solution[i + 1]]
        if len(solution) > 1:
            distance += self.distances[solution[-1], solution[0]]
        return distance

    def solve(self):
        if self.min_energy_for_return is None:
            self._set_min_energy_for_return()
        best_solution = None
        best_distance = float('inf')
        for iteration in range(self.num_iterations):
            solutions = self._construct_solution()
```

```
            valid_solutions = [sol for sol in solutions if len(sol) == self.num_nodes]
            if not valid_solutions:
                continue
            self._update_pheromones(valid_solutions)
            for solution in valid_solutions:
                distance = self._calculate_total_distance(solution)
                if distance < best_distance:
                    best_distance = distance
                    best_solution = solution
        return best_solution, best_distance
```

app.py

```
import os
from flask import Flask, render_template, request, jsonify, redirect, url_for
from flask_login import LoginManager, UserMixin, login_user, login_required, logout_user,
current_user
from AntAlgorithm import AntColonyOptimizationEV
from GoogleMapsAPI import GoogleMapsAPI

app = Flask(__name__)
app.secret_key = os.urandom(24)

login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'

API_KEY = ''
google_maps_api = GoogleMapsAPI(API_KEY)

# Mock user database
users = {'admin': {'password': 'admin'}}

class User(UserMixin):
    def __init__(self, username):
        self.id = username

@login_manager.user_loader
def load_user(user_id):
    return User(user_id)
```

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        if username in users and users[username]['password'] == password:
            user = User(username)
            login_user(user)
            return redirect(url_for('index'))
        else:
            return 'Invalid credentials'
    return render_template('login.html')

@app.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('login'))

@app.route('/')
@login_required
def index():
    return render_template('index.html')

@app.route('/add_point', methods=['POST'])
@login_required
def add_point():
    address = request.json.get('address')
    coordinates = google_maps_api.get_coordinates(address)
    if coordinates:
        return jsonify({'address': address, 'coordinates': coordinates}), 200
    else:
        return jsonify({'error': 'Invalid address'}), 400

@app.route('/optimize_route', methods=['POST'])
@login_required
def optimize_route():
    points = request.json.get('points')
```

```
    addresses = [point['address'] for point in points]
    coordinates = [tuple(point['coordinates']) for point in points]

    if len(coordinates) < 2:
        return jsonify({'error': 'Not enough points for optimization'}), 400

    distance_matrix = google_maps_api.get_distance_matrix(coordinates, coordinates)
    if distance_matrix is None:
        return jsonify({'error': 'Error getting distance matrix'}), 500

    num_ants = 10
    num_iterations = 100
    decay = 0.5

    aco = AntColonyOptimizationEV(coordinates, distance_matrix, num_ants, num_iterations,
decay)
    best_solution, best_cost = aco.solve()  # Expect only two values

    routes = []
    for i in range(len(best_solution) - 1):
        route = google_maps_api.get_route(coordinates[best_solution[i]],
coordinates[best_solution[i + 1]])
        if route:
            routes.extend(route)

    final_route = google_maps_api.get_route(coordinates[best_solution[-1]],
coordinates[best_solution[0]])
    if final_route:
        routes.extend(final_route)

    formatted_routes = [{'lat': step[0], 'lng': step[1]} for step in routes]

    return jsonify({'routes': formatted_routes, 'addresses': addresses}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

СПЕЦІАЛЬНА ЧАСТИНА З ОХОРОНИ ПРАЦІ

до кваліфікаційної роботи бакалавра

на тему:

ЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ МАРШРУТІВ ЕЛЕКТРОТРАНСПОРТУ НА ОСНОВІ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

Спеціальність 122 «Комп'ютерні науки»

122 – КРБ – 402.21810209

Виконав студент 4-го курсу, групи 401
_____ П. М. Петренко
«17» червня 2024 р.

Консультант: канд. техн. наук, доцент
_____А. О. Алексєєва
«17» червня 2024 р.

Миколаїв – 2024

ЗМІСТ

ВСТУП

Покращення умов праці є важливим завданням для забезпечення безпеки, здоров’я та ефективності працівників у будь-якому виробничому середовищі. Умови праці впливають на фізичне та психічне становище працівників, їхню продуктивність та загальний комфорт на робочому місці. З цієї причини постійне вдосконалення умов праці стає пріоритетним завданням для будь-якої організації.

Метою цього дослідження є оцінка ефективності заходів, спрямованих на покращення умов праці в обраному виробничому приміщенні. Це дозволить з’ясувати, наскільки успішними були заходи, спрямовані на покращення умов праці, та визначити можливість подальшого вдосконалення робочого середовища для забезпечення максимальної комфортності та безпеки працівників.

У даному дослідженні буде проведено аналіз факторів умов праці до та після впровадження заходів, розраховано інтегральну оцінку умов праці для обох випадків та визначено відсоток покращення умов праці після заходів. Основними цілями є оцінка ефективності проведених заходів та визначення можливих шляхів подальшого вдосконалення умов праці для забезпечення здоров’я та ефективності працівників.

1 ІНТЕГРАЛЬНА ОЦІНКА УМОВ ПРАЦІ ПРИМІЩЕННЯ

У даному розділі виконана інтегральна оцінка умов праці приміщення, в якому розробляється застосунок для планування маршрутів електротранспорту на основі інтелектуальних технологій.

1.1 Характеристики виробничого приміщення

Застосунок для вирішення задачі TSP розроблявся в домашніх умовах. Інтегральна оцінка умов праці буде стосуватись саме кімнати.

Будинок розташований в селі Миколаївське. Має такі параметри:

- довжина: 5 м;
- ширина: 3 м;
- висота: 2 м.

Загальна площа приміщення складає: 15 $м^2$.

В приміщенні зроблений ремонт по євростандартам, наявне металопластикове вікно з прозорим двокамерним склопакетом. Підлогове покриття - комерційний лінолеум. Вікна знаходяться з південної сторони.

У кімнаті розміщено 1 робоче місце, яке обладнане:

- персональним комп’ютером;
- маршрутизатором Wi-Fi;
- мережевим подовжувачем;
- другим монітором;
- лазерним принтером;
- настільною лампою;
- акустичною системою.

Біля робочого місця стоїть ємність з водою. По обидва боки знаходяться розетки. Біля дверей знаходиться зручний диван для відпочинку та шафа для зберігання речей.

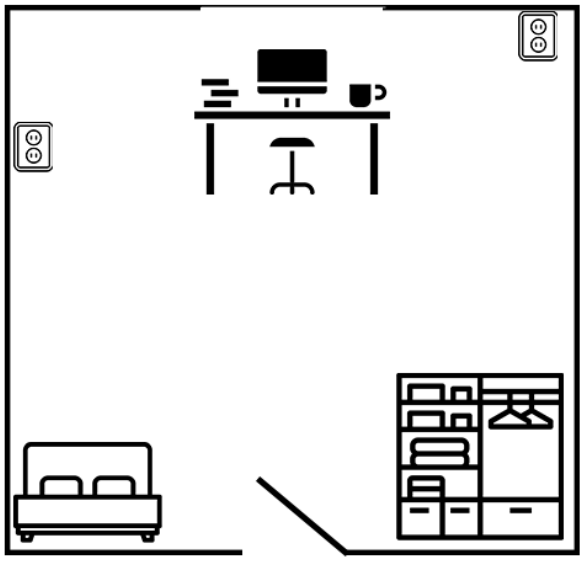

Рисунок 1.1 – план приміщення робочої кімнати у житловому будинку

Напруга джерела живлення техніки – 220 В. У приміщенні за правилами техніки безпеки знаходиться вогнегасник.

Кімната оснащена системою опалення та кондиціонування повітря. У вечірній час досить комфортно працювати, не відчувається нестачі світла.

Вимоги до електробезпеки в обраному приміщенні відповідає вимогам НПАОП 0.00-1.28-10.

1.2 Інтегральна оцінка умов праці в обраному виробничому приміщенні

Для інтегральної оцінки умов праці в обраному приміщенні треба застосувати вхідні дані, які подані в таблиці 1 та здійснити оцінку кожного з факторів трудового процесу за критеріями бальної оцінки , що наведено в додатку до даного розділу.

Таблиця 1.1 – Фактори умов праці в офісному приміщенні

№ з/п	**Фактор умов праці на робочому місці**	**Значення показника**	**Тривалість дії фактору, хв.**
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, ℃: - теплий період - холодний період	 23 -	 300 -
2	Відносна вологість повітря на РМ, %	65	480
3	Швидкість руху повітря на РМ, м/с	0,3	420
4	Освітленість на РМ, лк	140	360
5	Мінімальний розмір об’єкта розпізнавання, мм	0,5	360
6	Виробничий шум, дБА	50	420
7	Інтенсивність теплового випромінювання, Вт/м²	160	420

Продовження таблиці 1.1

8	Токсична речовина, озон, кратність перевищення ГДК	1,4	480
9	Виробничий пил (паперовий та ін.), кратність перевищення ГДК	0,5	360
10	Робоче місце (РМ), поза та переміщення у просторі	Робоче місце стаціонарне, поза не вільна, до 25 % часу зміни у нахиленому положенні до 30°	360
11	Кількість важливих об’єктів спостереження	1	420
12	Тривалість зосередженого спостереження, % часу зміни	50	360
13	Тривалість повторюваних операцій, с	50	360
14	Змінність роботи	Ранкова зміна	480
15	Тривалість безперервної роботи за добу, годин	6	360
16	Режим праці та відпочинку	Обґрунтований, без включення музики та гімнастики	480
17	Нервово-емоційне навантаження	Прості дії за заданим планом з можливістю корегування	420
18	Кількість рухів пальців на годину	400	360

У таблиці 1.2 наведені параметри, необхідні для інтегральної оцінки умов праці:

Таблиця 1.2 – Параметри, необхідні для розрахунку інтегральної бальної оцінки умов праці на робочому місці

№ з/п	**Фактор умов праці на робочому місці**	**Значення показника**	**Тривалість дії фактору, хв.**
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, °С: - теплий період - холодний період	 23 -	 300 -
2	Відносна вологість повітря на РМ, %	65	480
3	Швидкість руху повітря на РМ, м/с	0,3	420
4	Освітленість на РМ, лк	140	360
5	Мінімальний розмір об’єкта розпізнавання, мм	0,5	360
6	Виробничий шум, дБА	50	420
7	Інтенсивність теплового випромінювання, Вт/м²	160	420
8	Токсична речовина, озон, кратність перевищення ГДК	1,4	480
9	Виробничий пил (паперовий та ін.), кратність перевищення ГДК	0,5	360
10	Робоче місце (РМ), поза та переміщення у просторі	Робоче місце стаціонарне, поза не вільна, до 25 % часу зміни у нахиленому положенні до 30°	360
11	Кількість важливих об’єктів спостереження	1	420
12	Тривалість зосередженого спостереження, % часу зміни	24	360

Закінчення таблиці 1.2

13	Тривалість повторюваних операцій, с	50	360
14	Змінність роботи	Ранкова зміна	480
15	Тривалість безперервної роботи за добу, годин	6	360
16	Режим праці та відпочинку	Обґрунтований, з включення музики та гімнастики	480
17	Нервово-емоційне навантаження	Прості дії за заданим планом з можливістю корегування	420
18	Кількість рухів пальців на годину	300	360

де $x_{н_i}$ – нормативне значення i – того фактору умов праці (прийняті значення відповідають оптимальному (допустимому) класу умов праці згідно з Гігієнічною класифікацією [3]);

$x_{аб_i}$ – дійсне значення i – того фактору умов праці (відповідно до даних табл. 4.1);

x_{x_i} – оцінка i – того фактору умов праці (відповідно до даних додатку А даного розділу роботи), балів;

t_i – тривалість дії i – того фактору умов праці (відповідно до даних табл. 3.1), хв.;

t_{num_i} – відносна тривалість дії i – того фактору умов праці (за прийнятої тривалості робочої зміни $t_p = 480$ хв.), хв., тобто:

$$t_{num_i} = \frac{t_i}{t_p} = \frac{t_i}{480}; \quad (1.1)$$

x_{ϕ_i} – фактична оцінка питомої ваги *i* – того фактору умов праці, балів, а саме:

$$x_{\phi_i} = x_{x_i} t_{пит_i} = x_{x_i} \frac{t_i}{480}. \quad (1.2)$$

За даними таблиці 2 визначаємо елемент умов праці, який одержав у балах найбільшу оцінку x_{max}.

Найбільшу кількість балів отримав елемент *x2*, який пов'язаний з відносною вологістю повітря на робочому місці, %, тобто x_{max} = *x2* = 3. Даний елемент вважається визначаючим.

Щоб оцінити реальні умови праці в приміщенні, скористаємося наведеними даними та порівняємо їх із нормативними значеннями. Ми врахуємо кожний фактор, його фактичне значення, тривалість дії та його оцінку за нормативами. Використаємо надану вами таблицю з нормативними значеннями, оцінками та питомими вагами факторів.

Початок розрахунку:

1. температура повітря на робочому місці (теплий період):
   - норматив: 23…25;
   - фактичне: 23 (в межах нормативу);
   - оцінка: 1;
   - тривалість дії: 300 хв
   - питома тривалість: $t_{\text{питі}} = \frac{300}{480} = 0.625$;
   - питома вага: $x_{\text{фі}} = 1 \times 0.625 = 0.625$;
2. відносна вологість повітря:
   - норматив: 40-60%;
   - фактичне: 65% (вище нормативу);
   - оцінка: 3;
   - тривалість дії: 480 хв;

- питома тривалість: $t_{\text{питі}} = 1$;
- питома вага: $x_{\text{фі}} = 3 \times 1 = 3$;

3. швидкість руху повітря:
   - норматив: <0.2 м/с;
   - фактичне: 0.3 м/с (вище нормативу);
   - оцінка: 2;
   - тривалість дії: 420 хв;
   - питома тривалість: $t_{\text{питі}} = \frac{420}{480} = 0.875$;
   - питома вага: $x_{\text{фі}} = 2 \times 0.875 = 1.75$;
4. освітленість:
   - норматив: 200 лк
   - фактичне: 140 лк (нижче нормативу)
   - оцінка: 3
   - тривалість дії: 360 хв
   - питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$
   - питома вага: $x_{\text{фі}} = 3 \times 0.75 = 2.25$
5. мінімальний розмір об’єкта розпізнавання:
   - норматив: >1 мм
   - фактичне: 0.5 мм (нижче нормативу)
   - оцінка: 2
   - тривалість дії: 360 хв
   - питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$
   - питома вага: $x_{\text{фі}} = 2 \times 0.75 = 1.5$
6. виробничий шум:
   - норматив: 50 дба
   - фактичне: 50 дба (в межах нормативу)

- оцінка: 1
- тривалість дії: 420 хв
- питома тривалість: $t_{\text{питі}} = \frac{420}{480} = 0.875$
- питома вага: $x_{\text{фі}} = 1 \times 0.875 = 0.875$

7. інтенсивність теплового випромінювання:
   - норматив: ≤140 вт/м²
   - фактичне: 160 вт/м² (вище нормативу);
   - оцінка: 2;
   - тривалість дії: 420 хв;
   - питома тривалість: $t_{\text{питі}} = \frac{420}{480} = 0.875$;
   - питома вага: $x_{\text{фі}} = 2 \times 0.875 = 1.75$;
8. токсична речовина (озон):
   - норматив: ≤1;
   - фактичне: 1.4 (вище нормативу);
   - оцінка: 3;
   - тривалість дії: 480 хв;
   - питома тривалість: $t_{\text{питі}} = 1$;
   - питома вага: $x_{\text{фі}} = 3 \times 1 = 3$;
9. виробничий пил (паперовий та ін.):
   - норматив: ≤1;
   - фактичне: 0.5 (в межах нормативу);
   - оцінка: 1;
   - тривалість дії: 360 хв;
   - питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
   - питома вага: $x_{\text{фі}} = 1 \times 0.75 = 0.75$;
10. робоче місце (рм), поза та переміщення у просторі:

- норматив: рм стаціонарне, маса переміщення до 5 кг;
- фактичне: рм стаціонарне, поза не вільна, до 25 % часу у нахиленому положенні до 30° (в межах нормативу);
- оцінка: 3;
- тривалість дії: 360 хв;
- питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
- питома вага: $x_{\text{фі}} = 3 \times 0.75 = 2.25$;

11. кількість важливих об’єктів спостереження
    - норматив: < 5;
    - фактичне: 1;
    - оцінка: 1;
    - тривалість дії: 420 хв;
    - питома тривалість: $t_{\text{питі}} = \frac{420}{480} = 0.875$;
    - питома вага: $x_{\text{фі}} = 1 \times 0{,}875 = 0{,}875$;
12. тривалість зосередженого спостереження, % часу зміни;
    - норматив:< 25;
    - фактичне: 24%;
    - оцінка: 1;
    - тривалість дії: 360 хв;
    - питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
    - питома вага: $x_{\text{фі}} = 1 \times 0.75 = 0.75$;
13. тривалість повторюваних операцій, с
    - норматив: > 100;
    - фактичне: 50 с;
    - оцінка: 1;
    - тривалість дії: 360 хв;

- питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
- питома вага: $x_{\text{фі}} = 1 \times 0.75 = 0.75$;

14. змінність роботи:
    - норматив: ранкова;
    - фактичне: ранкова зміна;
    - оцінка: 1;
    - тривалість дії: 480 хв;
    - питома тривалість: $t_{\text{питі}} = 1$;
    - питома вага: $x_{\text{фі}} = 1$;
15. тривалість безперервної роботи за добу, годин:
    - норматив: < 8;
    - фактичне: 6 годин;
    - оцінка: 1;
    - тривалість дії: 360 хв;
    - питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
    - питома вага: $x_{\text{фі}} = 1 \times 0.75 = 0.75$;
16. режим праці та відпочинку:
    - норматив: обґрунтований, з включенням музики та гімнастики;
    - фактичне: обґрунтований, з включенням музики та гімнастики;
    - оцінка: припустимо 1;
    - тривалість дії: 480 хв;
    - питома тривалість: $t_{\text{питі}} = 1$;
    - питома вага: $x_{\text{фі}} = 1$;
17. нервово-емоційне навантаження:
    - норматив: прості дії за заданим планом;
    - фактичне: прості дії за заданим планом з можливістю корегування;

- оцінка: 1;
- тривалість дії: 420 хв;
- питома тривалість: $t_{\text{питі}} = \frac{420}{480} = 0.875$;
- питома вага: $x_{\text{фі}} = 1 \times 0{,}875 = 0{,}875$;

18. кількість рухів пальців на годину:

- норматив: <360;
- фактичне: 300;
- оцінка: 1;
- тривалість дії: 360 хв;
- питома тривалість: $t_{\text{питі}} = \frac{360}{480} = 0.75$;
- питома вага: $x_{\text{фі}} = 1 \times 0.75 = 0.75$.

Підсумковий розрахунок:

$$I = 0.625 + 3 + 1.75 + 2.25 + 1.5 + 0.875 + 1.75 + 3 + 0.75 + 2.25 + 0.875 + 0.75 + 0.75 + 1 + 0.75 + 1 + 0.875 + 0.75 = 24.5$$

Отримане значення інтегральної оцінки умов праці було проаналізовано і порівняно зі значеннями, наведеними в додатку Б даного розділу дипломної роботи бакалавра. Значення інтегральної оцінки відноситься до категорії.

Згідно даними додатку Б умови праці на визначеному робочому місці відносяться до II категорії.

1.3 Оцінка ефективності заходів щодо покращення умов праці

Пропонується до всіх факторів умов праці, бальна оцінка яких перевищує значення $x_{xi} = 2$, вжити заходи (надати рекомендації) з метою досягнення кожним із розглянутих елементів умов праці саме зазначеного вище значення ($x_{xi} = 2$).

За даними таблиці 2 визначаємо інтегральний показник важкості праці за формулою, що використовується, коли умови праці оцінюються балами «1» або

«2»:

$$I_{n_2} = 19{,}7 \cdot \bar{x} - 1{,}6 \cdot \bar{x}^2, \tag{1.3}$$

де

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}. \tag{1.4}$$

Після впровадження заходів, підсумуємо нові питомі ваги згідно таблиці 2:

Інтегральна оцінка до і після заходів

До заходів:

$I_n = 0.625 + 3 + 1.75 + 2.25 + 1.5 + 1.125 + 1.75 + 2 + 1.5 + 2.25 + 0.875 + 0.75 + 0.75 + 1 + 0.75 + 1 + 0.875 + 0.75 = 26.875$

Після заходів:

За даними таблиці 2 визначаємо інтегральний показник важкості праці за формулою, що використовується, коли умови праці оцінюються балами «1» або «2»:

$$I_{n_2} = 19{,}7 \cdot \bar{x} - 1{,}6 \cdot \bar{x}^2, \tag{1.5}$$

де

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}. \tag{1.6}$$

Тоді для даних умов праці згідно таблиці 2 маємо:

$$\underline{x} = \sum_{i=1}^{n} \frac{x_i}{n} = \frac{0.625 + 2 + 1.75 + 2 + 1.5 + 1.125 + 1.75 + 2 + 1.5 + 2 + 0.875 + 0.75 + 0.75 + 1 + 0.75 + 1 + 0.875 + 0.75}{18} = 1.28$$

$$I_{n_2} = 19.7 \cdot \underline{x} - 1.6 \cdot \underline{x}^2 = 19.7 \cdot 1.28 - 1.6 \cdot 1.28^2 = 31.5 - 4.1 = 22{,}6$$

Відповідно до даних, наведених у додатку Б, отримане значення інтегрального показника I_{n_2}=22,6 балів, відповідає II категорії умов праці, а саме

це такі роботи, що виконуються в умовах, які відповідають гранично допустимим концентраціям (ГДК) і рівням (ГДР) санітарно-гігієнічних елементів, а також допустимим рівням психофізіологічних факторів.

Інтегральний показник важкості праці дозволяє визначити вплив умов праці на працездатність людини у такій послідовності:

1. Ступінь втоми працівника на визначеному робочому місці *B*, у. о.

$$B = \frac{I_n - 15{,}6}{0{,}64}, \tag{1.7}$$

де чисельні значення 15,6 і 0,64 – це коефіцієнти регресії.

Тоді:

– до впровадження комплексу заходів з охорони праці коефіцієнт втоми складає

$$\mathrm{B}_1 = \frac{I_n - 15.6}{0.64} = \frac{26.875 - 15.6}{0.64} = 17.6 \ ;$$

– після впровадження комплексу заходів з охорони праці коефіцієнт втоми складає $\mathrm{B}_2 = \frac{I_{n_2} - 15.6}{0.64} = \frac{22.6 - 15.6}{0.64} = 10.9.$

2. Рівень працездатності людини *П*, у. о.

$$П = 100 - B. \tag{1.8}$$

Тоді:

– до впровадження комплексу заходів з охорони праці рівень працездатності складає $\mathrm{П}_1 = 100 - \mathrm{B}_1 = 100 - 17.6 = 82.4;$

– після впровадження комплексу заходів з охорони праці рівень працездатності складає $\mathrm{П}_2 = 100 - \mathrm{B}_2 = 100 - 10.9 = 89.1;$

3. Зміна продуктивності праці $\Delta П$, %.

$$\Delta\mathrm{П} = 0{,}2 \cdot \left(\frac{\mathrm{П}_2}{\mathrm{П}_1} - 1\right) \cdot 100 = 0{,}2 \cdot \left(\frac{89.1}{82.4} - 1\right) \cdot 100 = 1{,}62\%.$$

ВИСНОВКИ

Згідно з даними додатків А та Б, умови праці на визначеному робочому місці класифікуються як II категорія через відхилення в нормах відносної вологості повітря, освітленості, токсичних речовин, та неправильному положенні під час роботи.

Для покращення умов праці при зазначених проблемах необхідно перевірити рівень вологості за допомогою гігрометра та переконатися, що він знаходиться в нормі. Якщо останній занадто низький або високий, розглянути встановлення зволожувача або вентиляційної системи. Встановити регулярне провітрювання приміщення для забезпечення свіжого повітря.

Переконатися, що приміщення освітлене належним чином. Використовувати природне світло, де це можливо, та додаткові джерела освітлення при необхідності. Організовувати робочі місця так, щоб уникнути блиску на моніторах та інших пристроях.

Ідентифікувати та оцінити джерела токсичних речовин у вашому робочому середовищі. Зменшити експозицію до токсичних речовин шляхом встановлення ефективної вентиляційної системи або використання засобів індивідуального захисту.

Регулярно вставати зі столу та розтягуватися, щоб уникнути монотонної позиції. Забезпечити ергономічні меблі та обладнання, щоб підтримувати правильне положення тіла під час роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу // Охорона праці. 2001. № 12. С. 12-20.

2. Шевчук М.М., Куценко О.Д. Основи охорони праці : підручник. Л. : Вид-во ЛНУ, 2003. 392 с.

3. Климко Г.Г., Мартиненко І.О. Основи охорони праці : навч. посібник. К. : Кондор, 2005. 376 с.

4. Кучеренко В.І., Нагорний Є.І. Охорона праці в галузі : навч. посібник. К. : Каравела, 2006. –272 с.

5. Нестеренко М.А., Орлов Ю.А. Безпека життєдіяльності : підручник. К. : Вікар, 2003. 384 с.

6. Плотніков О.П. Охорона праці : навч. посібник. Х. : Вид-во НФаУ, 2004. 220 с.

7. Проектування систем вентиляції та кондиціювання повітря / В.С. Бойко, В.П. Ємельянов, В.О. Дворецький та ін. К. : Вища школа, 2006. 343 с.

8. Скакун І.П., Сокуренко В.М. Основи охорони праці : підручник. К. : Вікар, 2005. 320 с.

Додаток А

Критерії бальної оцінки умов праці

№ п\п	Фактор умов праці на робочому місці	Оцінка, бали					
		1	2	3	4	5	6
1	Температура повітря на робочому місці (РМ) у виробничому приміщенні, °С: - теплий період - холодний період	 23…25 21…23	 26…28 18…20	 29…32 15…17	 33…35 12…14	 35…37 Нижче +12	 >37 -
2	Відносна вологість повітря на РМ, %	40…50	55…60	61…75	76…85	Понад 85	-
3	Швидкість руху повітря на РМ, м/с	Менше 0,2	0,2…0,5	0,6…0,7	0,8…1,2	1,3…1,7	Понад 1,7
4	Освітленість на РМ, лк	≥ 300	240…300	160…230	100…150	60…90	30…50
5	Мінімальний розмір об’єкта розпізнавання, мм	> 1,0	1…0,3	< 0,3	0,005…0,3	< 0,05	-
6	Виробничий шум, перевищення ГДР, дБА	< 1	Рівно ГДР	1…5	6…10	> 10	> 10 з вібрацією
7	Інтенсивність теплового випромінювання, Вт/м²	≤ 140	141..1000	1001…1500	1501…2000	2001…2500	>2500
8	Токсична речовина, озон, кратність перевищення ГДК	-	≤ 1	1…2,5	2,6…4,0	4,1…6	> 6,0
9	Виробничий пил (паперовий), кратність перевищення ГДК	-	≤ 1	1…5	6…10	11…30	> 30

Закінчення додатку А

10	Робоче місце (РМ), поза та переміщення у просторі	РМ стаціонарне,поза вільна, маса переміщене вантажу ≤ 5 кг	РМ стаціонарне,поза вільна, маса переміщене вантажу > 5 кг	Робоче місце стаціонарне, поза не вільна, до 25 % часу зміни у нахиленому положенні до 30°	РМ стаціонарне,поза вимушена –до 50 % робочої зміни	РМ стаціонарне, поза вимушена, незручна – більше 50 % робочої зміни	РМ стаціонарне, поза вимушена, незручна, нахили під кутом до 60 град більше 300 разів за робочу змін
11	Кількість важливих об’єктів спостереження	Менше 5	5…10	11…25	Понад 25	-	-
12	Тривалість зосередженого спостереження, % часу зміни	Менше 25	25…50	51…75	76…85	86…90	Понад 90
13	Тривалість повторюваних операцій, с	Понад 100	31…100	20…30	10…19	5…9	1…4
14	Змінність роботи	Ранкова зміна	Дві зміни	Три зміни	Нерегулярні зміни	-	-
15	Тривалість безперервної роботи за добу, годин	-	< 8	< 12	> 12	-	-
16	Режим праці та відпочинку	Обґрунто-ваний, з включенням музики та гімнастики	Обґрунто-ваний, без включення музики та гімнастики	Відсутність обґрунтованого режиму праці та відпочинку	-	-	-
17	Нервово-емоційне навантаження	Прості дії за індивідуальним планом	Прості дії за заданим планом з можливістю корегування	Складні дії за заданим планом з можливістю корегування	Складні дії за заданим планом при дефіциті часу	Відповідальність за безпеку людей	Індивідуальний ризик
18	Кількість рухів пальців на годину	< 360	360…720	721…1080	1081…3000	> 3000	-

Додаток Б

Залежність категорії умов праці від величини інтегральної бальної оцінки

Діапазон інтегральної бальної оцінки	**Категорія умов праці**	**Характер роботи**
До 18	I	Роботи, що виконуються в оптимальних умовах
19…33	II	Роботи, що виконуються в умовах, які відповідають гранично допустимим концентраціям (ГДК) і рівням (ГДР) санітарно-гігієнічних елементів, а також допустимим рівням психофізіологічних факторів
34…45	III	Роботи, що відхиляються від ГДК і ГДР та допустимих рівнів психофізіологічних факторів
45,7…53,9	VI	Робота у несприятливих умовах праці
54…59	V	Роботи, що виконуються в екстремальних умовах
Понад 59	VI	Роботи, що виконуються в екстремальних умовах