Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

ГЮІК.502528.006 ПЗ
(позначення документа)

«Дослідження методів пошуку оптимальних шляхів руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів» .
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-19-1*
Спеціальність
122 – Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма
Системне проектування
(повна назва освітньої програми)
Костюкевич В.В.
(прізвище, ініціали)
Керівник *Решетнік В.М.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ *Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2020 р.

Атестаційна робота не містить відомостей заборонених до відкритого опублікування.

Атестаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведений «18» грудня 2020 р.

Керівник атестаційної роботи доц. Решетнік В.М.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Освітня програма *Системне проектування*
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри

_____
(підпис)
«___» _____2020 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Костюкевичу Владиславу Вікторовичу
(прізвище, ім'я по батькові)

1. Тема роботи *«Дослідження методів пошуку оптимальних шляхів руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів»*
затверджена наказом по університету від «2» 11 2020 р. № 1516Ст
2. Термін подання студентом роботи 22.12.2020
3. Вхідні дані роботи теоретичні відомості про існуючі алгоритми пошуку шляху та способи порівняння результатів дослідження

4. Зміст пояснювальної записки (перелік питань, що потрібно опрацювати)
1. Аналіз предметної області
2. Дослідження алгоритмів пошуку шляху
3. Порівняння результатів моделювання алгоритмів
4. Інтеграція компоненту вибору алгоритму пошуку оптимального шляху в інформаційну систему організації повітряної доставки корисних вантажів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів

5.1 Концептуальна діаграма компоненту системи (1 аркуш формату А4). 5.2 Декомпозиція діаграми компоненту системи (1 аркуш формату А4). 5.3. Декомпозиція функції взаємодії користувача через інтерфейс (1 аркуш формату А4) 5.4 Декомпозиція функції взаємодії безпілотного літального апарату з системою (1 аркуш формату А4) 5.5 Декомпозиція функції обслуговування програмної системи (1 аркуш формату А4) 5.6 Логічна схема бази даних системи (1 аркуш формату А4).5.7. Фізична схема бази даних системи (1 аркуш формату А4).

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на атестаційну роботу	02.11.20	
2	Аналіз завдання, підбір літератури	02.11.20-06.11.20	
3	Аналіз літератури з досліджуваної проблеми	06.11.20-10.11.20	
4	Аналіз технічних засобів для реалізації	10.11.20-12.11.20	
5	Дослідження алгоритмів	12.11.20-22.11.20	
6	Аналіз результатів дослідження	22.11.20-5.12.20	
7	Оформлення пояснювальної записки	5.12.20-19.12.20	
8	Перевірка на плагіат	20.12.20	
9	Представлення на рецензування	20.12.20	
10	Представлення атестаційної роботи в ДЕК	22.12.20	

Дата видачі завдання 2 11 2020р.

Студент _____
(підпис)

Керівник роботи _____ Решетнік В.М.
(підпис) (посада, прізвище ініціали)

РЕФЕРАТ

Атестаційна робота: 74 стор., 23 рис., 13 табл., 25 джерел інформації, 3 додатка.

АЛГОРИТМ ПОШУКУ ШЛЯХУ, ТРЬОХВИМІРНИЙ ПРОСТІР, ДОСЛІДЖЕННЯ, ПОШУК ОПТИМАЛЬНОГО ЗНАЧЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА

Об’єктом дослідження є інформаційна система організації повітряної доставки корисних вантажів.

Предметом дослідження є алгоритми пошуку оптимального шляху в тривимірному просторі для повітряної доставки корисних вантажів.

Мета дослідження – отримати результати дослідження алгоритмів пошуку шляху руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів та запропонувати вибір кращого рішення.

Методи дослідження – аналіз літератури за темою дослідження, математичне моделювання алгоритмів пошуку шляху, порівняння результатів дослідження.

Результат роботи – запропонований прийнятний кращий алгоритм пошуку шляху руху в трьохвимірному просторі для повітряної доставки корисних вантажів.

Галузь застосування – організація повітряної доставки корисних вантажів.

ABSTRACT

Attestation work: 74 p., 23 pic., 13 tables, 25 sources, 3 applications.

METHODS OF OPTIMAL MOVEMENT WAYS, THREE-DIMENSIONAL SPACE, RESEARCH, SEARCH FOR OPTIMAL VALUE, INFORMATION SYSTEM

The object of the research – information system of organizing air delivery of useful payloads.

The subject of the research – methods of optimal movement ways in three-dimensional space for air delivery of useful payloads.

The purpose of the work – to receive results of research of methods of optimal movement ways in three-dimensional space for information systems of the organization of air delivery of payloads and to offer a choice of the best decision.

Methods of working – analysis of the literature on the research topic, mathematical modeling of the methods of optimal movement ways, comparison of research results.

The results – the acceptable best method of optimal movement ways in three-dimensional space for air delivery of payloads is offered.

Scope – organization of air delivery of payloads.

ЗМІСТ

ВСТУП 6
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1 Опис предметної області 7
1.2 Опис сучасних проблем 8
1.3 Постановка задачі 9
1.4 Аналіз існуючих методів пошуку шляху в тривимірному просторі 11
2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ПОШУКУ ШЛЯХУ 18
2.1 Огляд існуючої літератури на тему дослідження 18
2.2 Алгоритм A* 18
2.3 Алгоритм D* 22
2.4 Алгоритм Theta* 26
2.5 Алгоритм Данцига 31
2.6 Алгоритм Флойда 34
2.7 Алгоритм Дейкстри 37
2.8 Алгоритм Лі 41
2.9 Вибір алгоритмів для порівняння результатів 43
3 ПОРІВНЯННЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ АЛГОРИТМІВ 45
3.1 Результати виконання алгоритмів 45
3.2 Порівняння отриманих результатів 50
4 ІНТЕГРАЦІЯ КОМПОНЕНТУ ВИБОРУ АЛГОРИТМУ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ В ІНФОРМАЦІЙНУ СИСТЕМУ ОРГАНІЗАЦІЇ ПОВІТРЯНОЇ ДОСТАВКИ КОРИСНИХ ВАНТАЖІВ 59
4.1 Визначення системних вимог до компоненту системи забезпечення доставки 59
4.2 Визначення функціональних вимог до компоненту системи забезпечення доставки 59
4.3 Логічне та фізичне моделювання даних 67
ВИСНОВКИ 72
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 73
ДОДАТОК А Графічні матеріали 75
ДОДАТОК Б Текст програми 83
Додаток В 101

ВСТУП

За останні декілька років інформаційні технології почали ставати однією з найбільш важливих галузей для життя людини. Частиною цього впливу є зростання попиту на різноманітні товари, що в свою чергу змусило підприємців вдосконалювати свої технології не тільки у виробництві товарів, а також у способах розповсюдження цих товарів серед людей. Головними проблемами у знаходженні ефективних методів транспортування є звичайно бюджет, який обмежує доступні ресурси, та кількість людей, яких можна влаштувати на роботу.

Метою дослідження цієї атестаційної роботи є огляд альтернативних методів транспортування, зокрема пошук найбільш прийнятного алгоритму пошуку шляху у тривимірному просторі для забезпечення повітряної доставки корисних товарів. Було обрано розгляд саме повітряних способів доставки, тому що вони дозволяють розв’язати проблему заторів під час доставки наземним транспортом, а їх використання разом з алгоритмами автоматизації пошуку шляху дозволяє тримати більшу кількість техніки на одного працівника. Також деякі повітряні апарати дешевше та легше замінити у разі несправності ніж наземний транспорт, який в свою чергу потребує людину для керування.

У якості методу дослідження проведено порівняння декількох основних алгоритмів пошуку шляху у тривимірному просторі використовуючи комп’ютерну модель повітряного літального апарату у межах міста. Така модель відображає симуляцію автоматичної поведінки повітряного літального апарату від початку процесу доставки до моменту його повернення на початкове місце.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

За останні 20 років вигляд українського ринку та підприємств, які його заповнюють, дуже сильно змінився, в той час як способи вироблення та продажу товарів майже ті самі. Якщо обрати будь-яке підприємство корисних товарів того часу, то можна побачити загальний шлях товарів від виробника сировини до покупця готового товару. Уся сировина автомобілями перевозиться на підприємство, обробляється та за допомогою різноманітних верстатів перетворюється в кінцевий продукт, який в свою чергу розповсюджують по магазинам та продають на місці.

На даний момент найбільш ефективним способом доставки сировини на підприємство є використання вантажних автомобілів [1]. Вони мають достатньо велику вантажопідйомність та об’єм вантажу, який можуть перевезти за один раз. Такий спосіб не є найбільш швидким та потребує виключної уваги водія, не забуваючи про графік цього водія, який він зобов’язаний підтримувати. Чим більша відстань транспортування, тим більше стресу у водія, тому що йому потрібний час для відпочинку, що впливає на швидкість перевезення. Незважаючи на швидкість, такий спосіб є досить надійним, хоч і ця надійність у більшості залежить від професіоналізму водія. Також об’єм вантажних автомобілів дозволяє настільки ефективно, наскільки можливо, спланувати маршрут таким чином, щоб один автомобіль міг забезпечити сировиною одразу декілька різних підприємств та їх підрозділів за одну поїздку.

Після виготовлення товару його необхідно розповсюдити по усім необхідним точкам продажу. Цей процес також проводиться за допомогою вантажних автомобілів, хоч і займає набагато менше часу ніж доставка сировини. Надалі увесь

процес передачі товарів у руки клієнтів лягає на плечі магазинів та, власне, самих клієнтів.

Якщо більша частина життєвого процесу товару важливіша для самого підприємства, то процес покупки та отримання товару важливий для самих покупців. Майже кожна людина завжди прагне розподілити свій час таким чином, щоб мати більше часу для корисних для неї справ, при цьому не забуваючи про рутинні дії, такі як похід у магазин за продуктами, чи пошук потрібного товару шляхом відвідування усіх доступних магазинів неподалік. У випадку коли не вдається знайти необхідну річ неподалік, людина змушена шукати буквально на іншому кінці міста. Саме тому на сьогоднішній день все більш популярними стають інтернет-магазини – веб-сайти, які пропонують безліч товарів з можливістю доставки не тільки в межах міста, а навіть із-за кордону [2]. Але більшість таких магазинів пропонує лише декілька способів доставки такі як доставка на ближчий поштовий філіал, чи доставка додому кур’єром.

1.2 Опис сучасних проблем

Більшість сучасних методів доставки достатньо зручні, авжеж у різних покупців різні потреби та можливості, але не завжди ефективні або швидкі для самого сервісу доставки.

Якщо розглянути шлях товарів від сировини до клієнта спочатку, при чому зосередившись на недоліках цього процесу, можна знайти деякі способи покращення методів доставки, що використовуються.

Вантажні автомобілі використовуються вже більше сторіччя, тому важко сказати, що вони непотрібні чи застарілі. Навпаки, чим більше часу пройшло, тим далі розвинулись технології, які покращили старий концепт майже в усіх можливих напрямах. Ідея вантажних автомобілів проста та ефективна: вмістити якомога більше вантажу в один автомобіль та найняти водія, який безпечно скерує цей

автомобіль до точки прибуття. Пройшло більше 100 років з моменту конструювання першого вантажного автомобілю, їх вигляд, швидкість, вантажопідйомність, керування та багато інших аспектів сильно змінилися. Незважаючи на це головна ідея лишилась та сама. Навіть у наш час досі використовують наземний транспорт та водія: головні компоненти, які є центром великих вантажоперевезень.

Тому у наземних перевезеннях є дві проблеми: якщо обраний шлях буде несподівано заблокований [3], то у кращому випадку автомобіль повинен шукати об'їзд, а в гіршому – необхідно чекати доки перешкода буде ліквідована; якщо у водія виникнуть проблеми зі здоров'ям чи виникне ситуація, що не дозволить йому сісти за кермо, то потрібно швидко шукати заміну цьому водію задля підтримки робочого графіку.

Обидві наведені проблеми можна розв'язати використовуючи безпілотні літальні апарати: будь-яку перешкоду можна облетіти за мінімальний витрачений час та відпадає необхідність у водії, що не тільки дозволить проводити неперервні доставки, а також дозволить збільшити кількість проведених доставок одночасно.

1.3 Постановка задачі

Метою виконання атестаційної роботи є отримання результатів дослідження методів пошуку оптимальних шляхів руху в трьохвимірному просторі та розробка рекомендацій з їх вибору для інформаційних систем організації повітряної доставки корисних вантажів.

Вхідними даними до методів пошуку шляху є початкові координати ($S(x_s,y_s,z_s)$), кінцеві координати ($E(x_e,y_e,z_e)$), вартість переходу між вузлами (g), відстань між вузлами (h) та відстань пройденого шляху (f) (див. рис. 1.1).

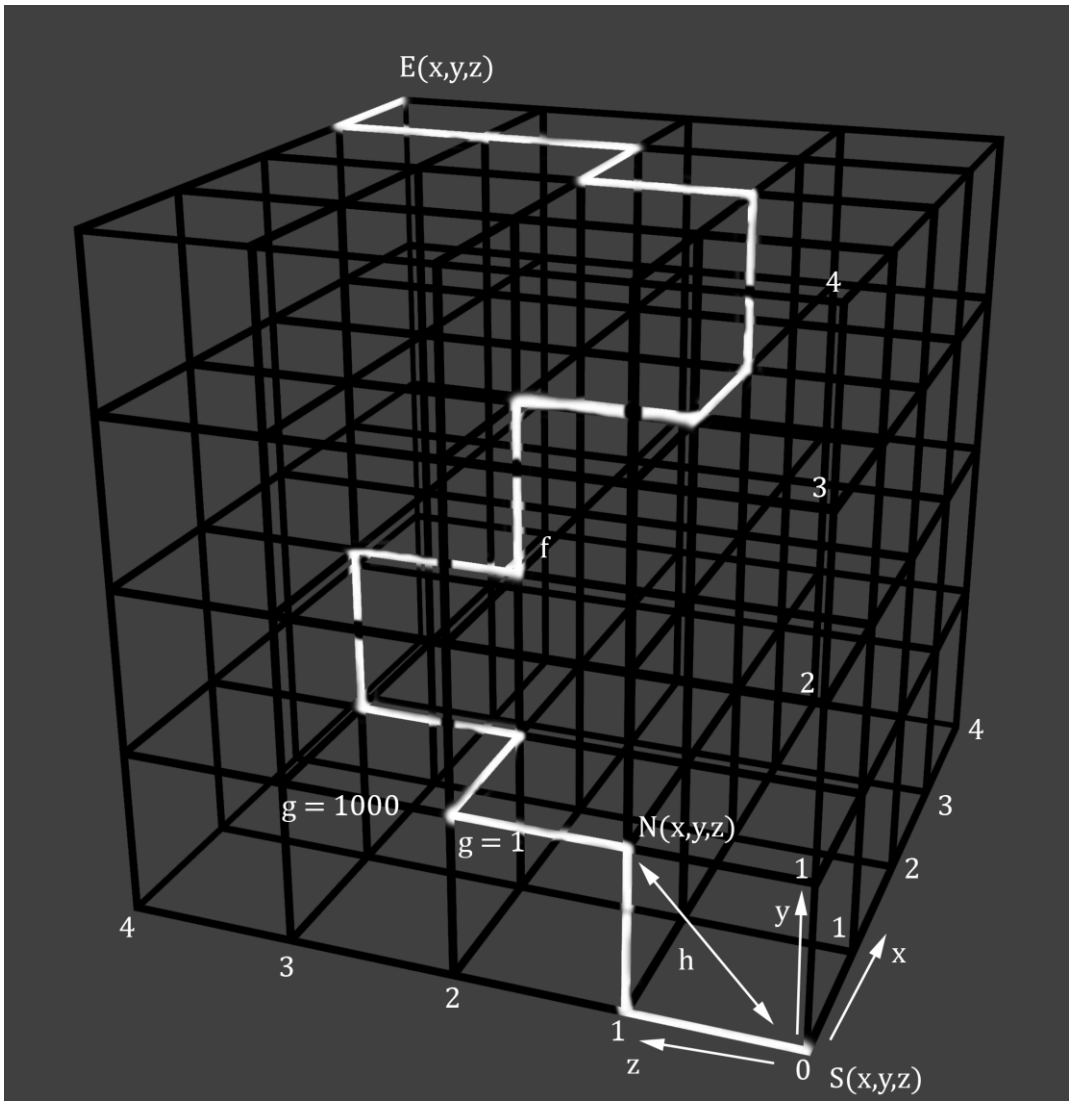

Рисунок 1.1 - Вхідні дані до методів пошуку шляху

Критеріями вибору методів є час повного проходження шляху (t_a, сек), кількість вузлів у шляху (n_p), кількість опрацьованих вузлів (n_a).

Заради ефективного дослідження результатів алгоритмів пошуку оптимального шляху накладаються деякі обмеження:

– припускається, що незаблокований шлях між початковою та кінцевою точкою завжди існує;

– мінімально можлива кількість точок у трьохвимірній сітці дорівнює 4;

– присутня кількість перешкод має знаходитись в діапазоні від 10% до 60% від загальної кількості вузлів;

– припускається, що всі алгоритми використовуються на безпілотних літальних апаратах одного типу;

– припускається, що корисний вантаж, який транспортується безпілотним літальним апаратом, не перевищує деяку максимальну вантажопідйомність та просторові габарити;

– під час руху за розрахованим шляхом, безпілотний літальний апарат працює безвідмовно та без технічних несправностей.

Для досягнення поставленої мети сформовано наступні задачі:

– проаналізувати існуючі методи розв’язування обраної задачі;

– дослідити обрані методи розв’язування задачі;

– отримати результати порівняння методів пошуку шляху;

– розробити рекомендації щодо вибору оптимальних шляхів руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів.

1.4 Аналіз існуючих методів пошуку шляху в тривимірному просторі

Головною метою конструювання комп’ютерів, або як їх спочатку називали електронно-обчислювальні машини, була автоматизація процесів, виконуваних людиною. Спочатку це були базові на той момент процеси, такі як арифметичні операції над числами. Після кожного успішно виконаного комп’ютером завдання програмісти створювали більш складні завдання, щоб побачити, на що здатні ці машини. В той самий час інженери не стояли на одному місці та проектували нові покращені моделі комп’ютерів.

З часом почали програмувати різноманітні складні алгоритми, одними з яких є алгоритми пошуку шляху. Найпростішими на той час були задачі з переставлення фігурки з однієї клітинки на іншу в квадратному полі, складеному з менших квадратів.

Далі було розроблено багато таких двовимірних алгоритмів, кожен з яких має свої переваги та недоліки.

З розвитком трьохвимірних технології [4] для проектування та створення комп'ютерних ігор з'явилась потреба у трьохвимірних алгоритмах пошуку шляху. Спочатку такі алгоритми були створенні шляхом модифікації існуючих двовимірних алгоритмів.

Один з найвідоміших алгоритмів є алгоритм пошуку шляху A* [5] (див. рис. 1.2). Цей алгоритм крок за кроком переглядає всі шляхи, що ведуть від початкової вершини в кінцеву, поки не знайде мінімальний. Як і всі алгоритми пошуку з інформацією про поле, він переглядає спочатку ті маршрути, які здається, що ведуть до мети. Від звичайного алгоритму, який є алгоритмом пошуку по першому найкращому збігу, його відрізняє те, що при виборі вершини він враховує, крім іншого, весь пройдений до неї шлях. У трьохвимірному просторі основний процес пошуку шляху не дуже відрізняється, лише додається координата третьої просторової координати.

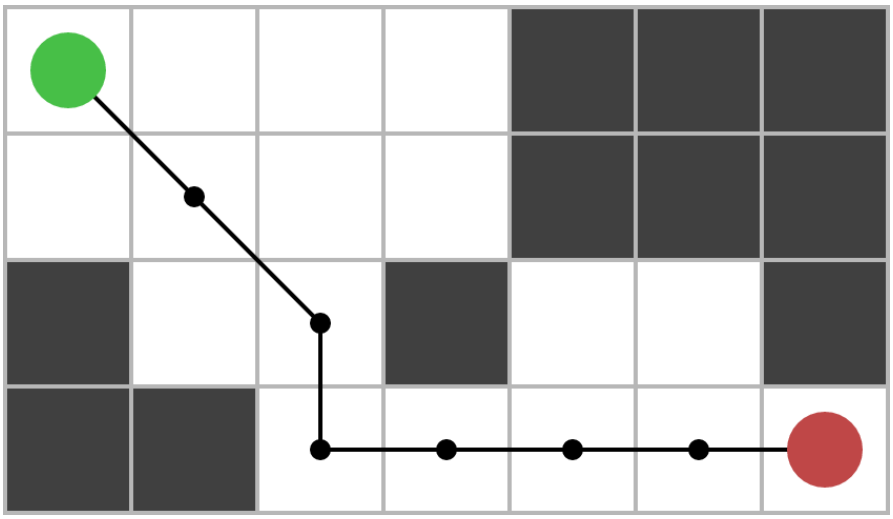

Рисунок 1.2 – Приклад алгоритму пошуку шляху A*

Цей алгоритм працює, використовуючи два списки: відкритий і закритий. Мета відкритого списку - утримувати потенційні найкращі вузли шляху, які ще не розглянуті, починаючи з початкового вузла. Якщо відкритий список стає порожнім,

тоді можливого шляху немає. Закритий список починається порожнім і містить усі вузли, які вже були розглянуті.

Основний цикл алгоритму обирає вузол із відкритого списку з найнижчою розрахованою вартістю до кінцевого вузла. Якщо вибраний вузол не є кінцевим, він поміщає всі допустимі сусідні вузли у відкритий список і повторює процес.

Важливою частиною алгоритму є те, що всі створені вузли зберігають посилання на попередній елемент. Це означає, що можна повернутися до початкового вузла з будь-якого вузла, створеного алгоритмом.

Алгоритм пошуку D* (див. рис 1.3) вирішує проблеми планування шляху на основі припущення, включаючи планування з припущенням про вільний простір, де об’єкт повинен переходити до заданих координат цілі на невідомій місцевості [6]. Він робить припущення щодо невідомої частини місцевості і знаходить найкоротший шлях від її поточних координат до координат цілі за цими припущеннями. Потім об’єкт йде шляхом. Коли він спостерігає нову інформацію на карті, він додає інформацію на свою карту і, якщо потрібно, перепланує новий найкоротший шлях від своїх поточних координат до заданих координат цілі. Він повторює процес, поки не досягне цільових координат або не визначить, що координати цілі досягти неможливо. При об'їзді невідомої місцевості нові перешкоди можуть часто виявлятися, тому це перепланування повинно бути швидким.

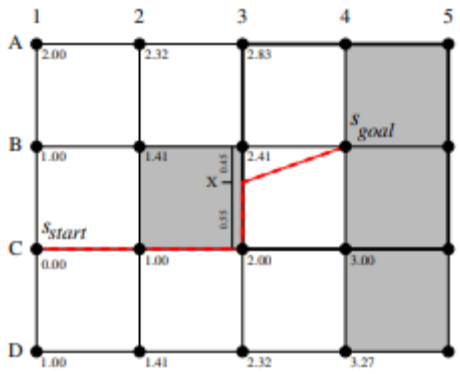

Рисунок 1.3 – Приклад алгоритму пошуку шляху D*

Алгоритм D* використовує декілька станів вузлів, які відображають поточний статус вузла в списку вузлів. Відкритий стан показує вузли які доступні для переміщення. Закритий стан відповідно показує вузли, що недоступні для переміщення. Також у цьому алгоритмі використовується стан нових вузлів, які ще не потрапляли у список відкритих вузлів. Два останні стани зображають вартість вузла відповідно до його попереднього стану: підвищення чи зниження вартості.

Алгоритм Theta* (див. рис. 1.4) є версією A*, який поширює інформацію по краях сітки, не обмежуючи шляхи до країв сітки [7]. Він поєднує ідеї A* на графіках видимості та A* на сітках. Ключова різниця між Theta* та A* на сітках полягає в тому, що в алгоритмі Theta* наступною вершиною може бути будь-яка вершина, тоді як в алгоритмі A* наступна вершина повинен бути сусідом попередньої вершини.

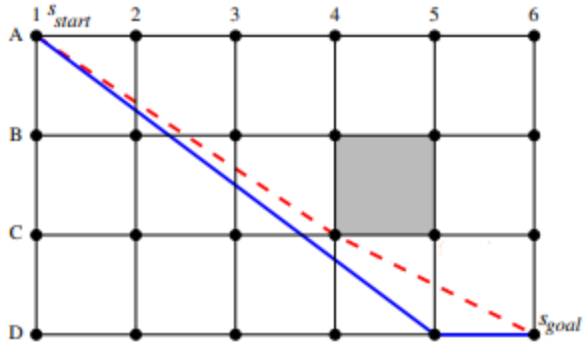

Рисунок 1.4 – Приклад алгоритму пошуку шляху Theta*

Алгоритм пошуку Theta* хоч і дуже схожий на свого попередника A*, але має важливу різницю: під час виконання розрахунків він будує два шляхи замість одного, що дозволяє скоротити відстань шляху між перешкодами. Перший шлях – це звичайний шлях від вершини до вершини вузлів, який проходить через усі вершини, сусідні одна з одною, які і формують шлях від початкової вершини до кінцевої. Другий шлях – це шлях між вершинами, який пов’язує вершини у полі зору одна одної, навіть якщо вони не є сусідніми на сітці. Такий шлях дозволяє

згладити кути шляху, огинаючи тільки перешкоди незважаючи на прямокутну форму сітки.

Алгоритм Данцига (див. рис. 1.5) дозволяє знаходити найкоротший шлях до усіх вершин графа [8]. Даний алгоритм використовує декілька принципів, що дозволяють йому ефективно знаходити необхідний шлях: будь-який короткий шлях є або дугою між початковою та кінцевою вершиною, або коротшим шляхом у підграфі від початкової вершини, до вершини, яка має дугу до кінцевої вершини; якщо існують цикли від'ємної вартості, то існує шлях з від'ємною вартістю; найкоротший шлях між двома будь-якими точками у графі або такий же як у підграфі, або послідовність найкоротших шляхів у графі.

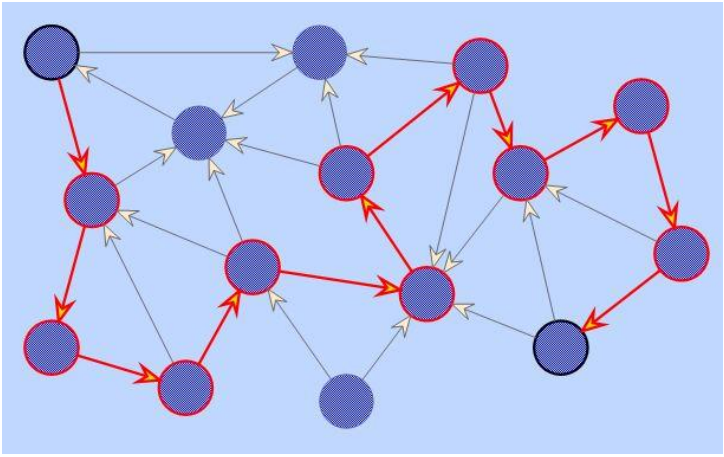

Рисунок 1.5 – Приклад пошуку шляху за алгоритмом Данцига

Алгоритм Флойда (див. рис. 1.6) використовується, якщо необхідно знайти найкоротші шляхи між усіма парами вершин [9]. Для цього спочатку усі шляхи позначаються нескінченністю, після чого підраховуються усі шляхи, які мають різну кількість проміжних вершин. Далі знаходяться мінімальні з розрахованих шляхів, на основі яких будується коротший шлях через комбінацію усіх необхідних пар вершин.

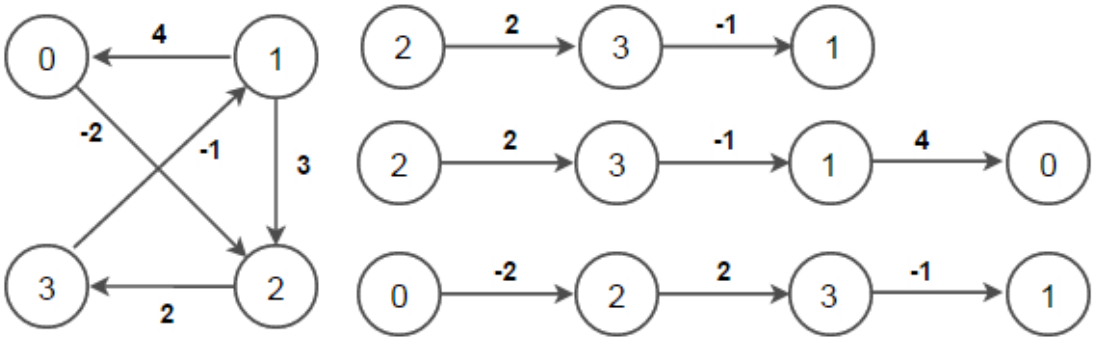

Рисунок 1.6 – Приклад пошуку найкоротших шляхів за алгоритмом Флойда

Алгоритм Дейкстри (див. рис. 1.7) є одним з простіших алгоритмів пошуку шляху, але не потребує велику кількість ресурсів [10]. Головним принципом алгоритму є перебір доступних вершин для формування шляху. Тому у даному алгоритмі присутні списки перевірених вершин, доступних вершин та неперевірених вершин. На початку створення шляху алгоритм обирає доступну вершину с найменшою вартістю та помічає її як перевірену. Такий спосіб дозволяє ніколи не повертатися до перевірених вершин. Також перевіряються сусідні до перевіреного вузли, та помічаються як доступні, якщо ніколи не були перевірені до цього. Отже, завжди існує найдешевша вершина із неперевірених, при чому повторно вершини не обробляються.

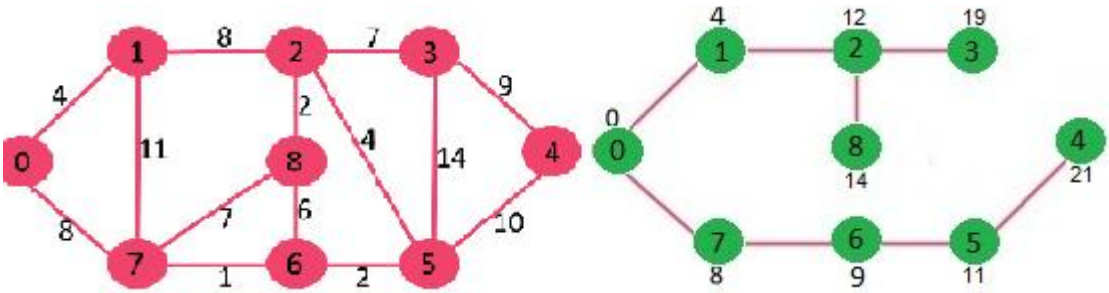

Рисунок 1.7 – Приклад пошуку шляху за алгоритмом Дейкстри

Алгоритм Лі (див. рис. 1.8) є варіацією алгоритму пошуку шляху в ширину [11]. Цей алгоритм дозволяє обходити навколишні вузли сітки шляхом пошуку

найближчих до початкової точки неперевірених сусідів. Головною особливістю алгоритму Лі є хвильовий пошук цих сусідніх вузлів. Алгоритм Лі використовує ітеративний спосіб знаходження шляху, тобто на кожній ітерації виникає нова хвиля пошуку. Кожен перевірений вузол помічається номером ітерації, на якій він був знайдений. У результаті виконання алгоритму виходить набір вузлів з позначеннями ітерацій, що дозволяє легко відновити шляху від цільового вузла до початкового шляхом поступового зменшення номера ітерації.

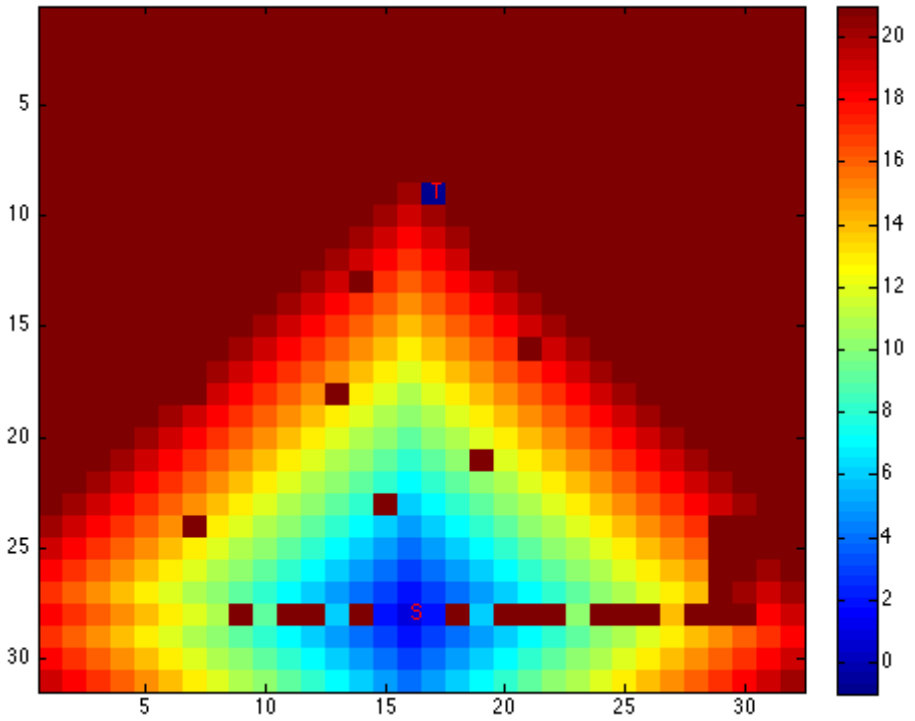

Рисунок 1.8 – Приклад пошуку шляху за алгоритмом Лі

З цього короткого огляду відомих алгоритмів пошуку шляху можна зробити висновок, що усі однаково доступні для адаптації у тривимірному просторі. Але, щоб дізнатися наскільки ці адаптації будуть ефективні порівняно одна з одною, необхідно дослідити їх роботу на тривимірній сітці вузлів.

2 ДОСЛІДЖЕННЯ АЛГОРИТМІВ ПОШУКУ ШЛЯХУ

2.1 Огляд існуючої літератури на тему дослідження

Тема алгоритмів пошуку оптимального шляху почала користуватися популярністю з моменту конструювання перших комп'ютерів, здатних виконувати покрокові завдання. Пройшло багато років, можливості обчислювальних ресурсів збільшились в десятки разів, але алгоритми пошуку шляху все ще актуальні. За останні декілька років було написано безліч наукових праць, які розглядають алгоритми пошуку шляху в іграх [12], у реалізаціях штучного інтелекту [13], а також при знаходженні нових оптимальних способів транспортування вантажу [14] та навіть для захисту важливих підприємств від проникнення ворогів [15].

Більшість алгоритмів ще відомі з минулого сторіччя, наприклад такі книги як «Вступ до алгоритмів» [16] та «Швидкі алгоритми для коротких шляхів» [17], але навіть подібні книги отримують нові видання з оновленою актуальністю інформації.

Тому при проведенні аналізу літератури для дослідження знайшлась велика кількість корисної та доступної інформації.

2.2 Алгоритм A*

Як і більшість алгоритмів пошуку шляху, даний алгоритм використовує вузли, які знаходяться на сітці таких вузлів, а також мають декілька характеристик, що відрізняють їх один від одного [18].

Вузол в цьому алгоритмі має чотири різні характеристики:

– координати вузла;

– вартість переходу від стартового вузла до поточного вузла;

– відстань між поточним вузлом та кінцевим вузлом;

– додана вартість усього шляху, що проходить через поточний вузол.

Координати вузла представляють собою набір чисел, які відповідають за положення центру вузла відповідно до кожної з трьох осей координат:

$$N(x_n, y_n, z_n) \tag{2.1}$$

де *x, y, z* – координати на кожній з осей.

Вартість переходу від стартового вузла до поточного вузла (параметр g) розраховується за наступною формулою:

$$g(n) = g(np) + cost(np, n), \tag{2.2}$$

де n – поточний вузол;

np – попередній вузол;

$cost(np,n)$ – вартість переходу від попереднього вузла до поточного.

Відстань між поточним вузлом та кінцевим вузлом (параметр h) може розраховуватись за декількома метриками, такі як манхеттенська відстань за формулою (2.3) чи відстань Евкліда за формулою (2.4), які можна розрахувати за наступними формулами:

$$h(n) = |n.x - goal.x| + |n.y - goal.y| + |n.z - goal.z|, \tag{2.3}$$

$$h(n) = \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2 + (n.z - goal.z)^2}, \tag{2.4}$$

де n – поточний вузол;

$goal$ – кінцевий вузол;

x,y,z – трьохвимірні координати вузлів.

Наприкінці, для знаходження вартості усього шляху, що проходить через поточний вузол (параметр f), використовується наступна формула:

$$f(n) = g(n) + h(n), \quad (2.5)$$

де n – поточний вузол;

$g(n)$ - вартість переходу від стартового вузла до поточного вузла;

$h(n)$ - відстань між поточним вузлом та кінцевим вузлом.

Виходячи з характеристик вузла є можливість почати виконання алгоритму (див. рис. 2.1).

Крок 1: створюються два списки – відкритий та закритий. Відкритий список зберігає усі вузлі, які доступні для переміщення, закритий список зберігає усі вузли недоступні для переміщення. Спочатку закритий список не тримає в собі жодних елементів. У відкритий список додається початковий вузол.

Крок 2: розраховується довжина шляху f від початкового елементу до кінцевого за формулою (2.5). Вартість початкового вузла встановлюється рівною нулю, тому що об’єкт не може повернутися до початкової позиції після початку руху.

Крок 3: відкривається цикл, який обробляє усі елементи з відкритого списку до моменту, коли даний список стане порожнім. Першим елементом перевірки в циклі стає елемент з відкритого списку з найменшою вартістю переходу. Під час першої ітерації циклу таким елементом є стартовий вузол, тому що інші елементи не були перевірені.

Крок 4: відбувається перевірка, яка вирішує чи був досягнутий кінцевий вузол за час роботи циклу. Така перевірка порівнює поточний вузол у циклі та кінцевий вузол. Якщо вузли співпадають, то виконується функція побудови повного шляху та алгоритм припиняє свою роботу. Функція побудови повного шляху зберігає кожний пройдений вузол від кінцевого до початкового, що дозволяє відновити увесь пройдений шлях.

Крок 5: якщо кінцевий вузол не досягнутий, то поточний вузол, що перевіряється, видаляється зі списку відкритих вузлів та додається до списку закритих вузлів. Така операція дозволяє алгоритму завжди рухатися вперед не перевіряючи повторно вже пройдені вузли.

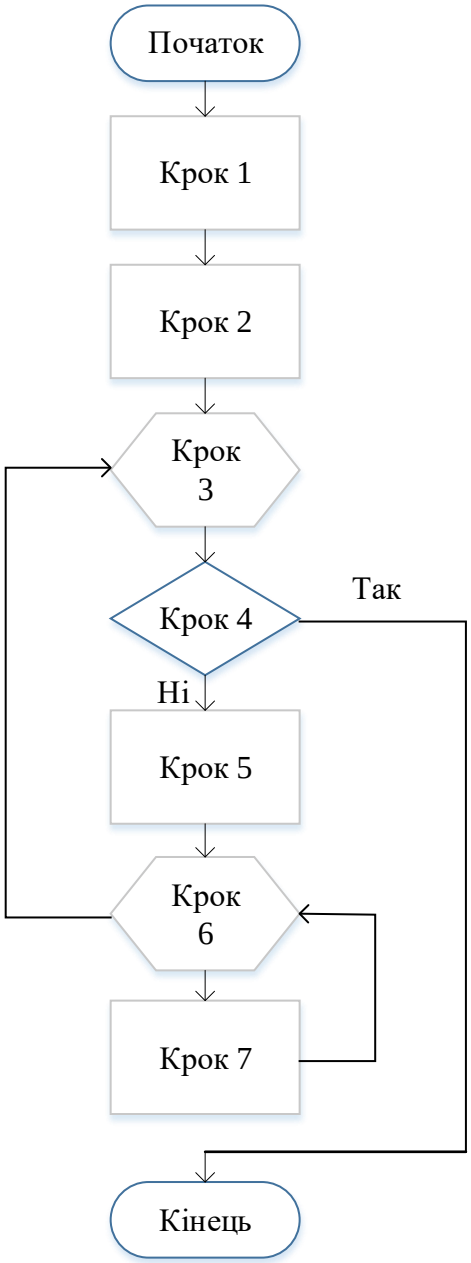

Рисунок 2.1 – Алгоритм пошуку шляху A*

Крок 6: починається цикл перевірки усіх сусідніх вузлів до поточного вузла. Якщо сусідній вузол не зберігається у закритому списку, то можна почати його перевірку. Для початку розраховується вартість пройденого шляху від поточного сусіднього вузла до кінцевого за формулою (2.5).

Крок 7: якщо сусід не зберігається у відкритому списку, то він додається до цього списку. Якщо поточний сусід вже зберігаються у відкритому списку, то відбувається порівняння поточного сусіду із іншим сусідом з відкритого списку.

Обирається сусід з найдешевшою вартістю переходу g від початкового вузла до поточного.

Після перевірки усіх вузлів, сусідніх до того, що перевіряється, обирається наступний вузол для перевірки, як вказано під час кроку 3. Алгоритм продовжує свою роботу до моменту знаходження шляху від початкового до кінцевого елементу.

2.3 Алгоритм D*

Алгоритм називається D*, оскільки він нагадує A*, за винятком того, що він динамічний у тому сенсі, що дугові витрати можуть змінитись під час переміщення об’єкту за сформованим шляхом. За умови, що перехід між вузлами належним чином пов'язаний з перебудовою оптимального шляху, то результат гарантовано буде оптимальним [19].

Для формалізації даних введемо наступні позначення та визначення. Проблемний простір можна сформулювати як набір поглядів, що позначають вузли, з'єднані напрямленими дугами, кожна з яких має відповідну вартість. Об’єкт починає рух в певному вузлі і рухається за дугами (з урахуванням вартості переходу) до інших вузлів, поки не досягне цільового вузла, позначеного E. Кожен вузол N, крім кінцевого, має зворотний вказівник на наступний вузол M, позначений як

$$b\,(N) = M \tag{2.6}$$

Алгоритм D* використовує зворотні вказівники для представлення шляхів до кінцевого вузла. Вартість переходу за дугою із вузла M до вузла N є додатним числом, заданим функцією вартості дуги

$$c(N, M) \tag{2.7}$$

Якщо M не має дуги до N, то $c(N, M)$ не визначено. Два вузли N і M є сусідами в просторі, якщо визначено $c(N, M)$ або $c(M, N)$.

Як і алгоритм A*, алгоритм D* підтримує відкритий список вузлів. Відкритий список використовується для розповсюдження інформації про зміни до функції вартості дуги та для обчислення витрат шляху до вузлів у просторі. Кожен вузол N має відповідний стан $t(N)$. який вказує на новий вузол, якщо N ніколи не був у списку відкритих вузлів, відкритий вузол, якщо N присутній у списку відкритих вузлів, та закритий вузол, якщо N більше немає у списку відкритих вузлів. Для кожного вузла N алгоритм D * зберігає оцінку суми витрат шляху від N до E, задані функцією вартості шляху:

$$h\ (E, N) \tag{2.8}$$

За належних умов, ця оцінка еквівалентна оптимальній вартості шляху від вузла N до E. Для кожного вузла N у списку відкритих вузлів присутня ключова функція

$$k\ (E, N) \tag{2.9}$$

Ця функція визначається рівною мінімуму $h(E, N)$ до наступних модифікацій та всі значення беруться ті, що прийняті $h(E, N)$, оскільки N було внесено до списку відкритих вузлів. Ключова функція класифікує вузли N у списку відкритих вузлів на один з двох типів: стан підвищеної вартості, якщо $k(E,N) < h(E, N)$, і стан нижчої вартості, якщо $k(E, N) = h(E, N)$. Алгоритм D* використовує стан підвищеної вартості на відкритому списку для розповсюдження інформації про збільшення вартості шляху, а стан нижчої вартості – для поширення інформації про скорочення вартості шляху. Поширення інформації відбувається шляхом багаторазового вилучення вузлів зі списку відкритих вузлів. Кожен раз, коли вузол вилучається зі

списку, він розширюється для передачі змін вартості сусідам. Цих сусідів, у свою чергу, ставлять у список відкритих вузлів, щоб продовжити процес.

Вузли у списку відкритих вузлів відсортовані за значенням ключової функції. Параметр k_{min} визначено як

$$min\ (k\ (N)), \tag{2.10}$$

для всіх N таких, що зберігаються у відкритому списку.

Параметр k_{min} представляє важливу межу в алгоритмі D*: якщо шлях коштує менше або дорівнює k_{min}, то він є оптимальним, а якщо вартість перевищує k_{min}, то такий шлях може бути неоптимальним. Параметр k_{old}, визначається рівним k_{min} до останнього вилучення вузла зі списку відкритих вузлів. Якщо жоден вузол не було видалено, k_{old} не визначено.

Впорядкування вузлів визначається як послідовність, якщо

$$b\ (N_i + 1)\ =\ N_i, \tag{2.11}$$

для всіх i таких, що $1 < i < N$.

Таким чином, послідовність визначає шлях зворотних вказівників від X_n, до X_1. Алгоритм D* будує та підтримує монотонну послідовність $\{E, N\}$, що представляє зменшення поточних або нижчих витрат шляху. для кожного вузла N, який є або був у списку відкритих вузлів.

Виходячи з представлених характеристик є можливість почати виконання алгоритму (див. рис.2.2).

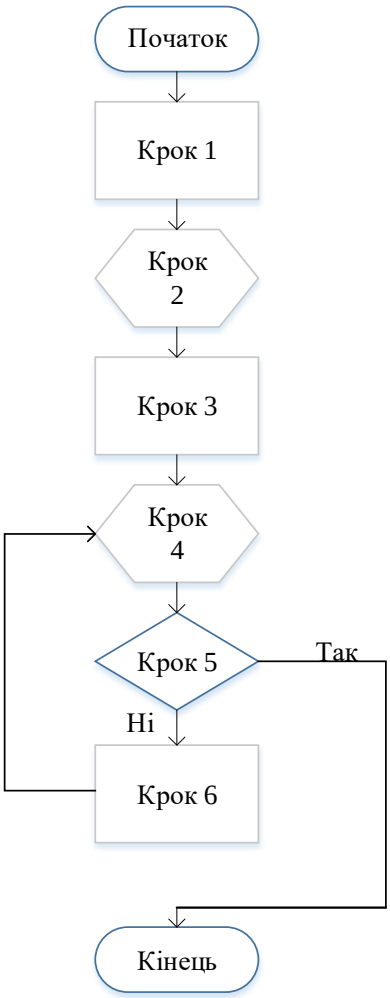

Рисунок 2.2 – Алгоритм пошуку шляху D*

Крок 1: спочатку усі вузли у сітці помічаються новими, що дозволяє починати обробку кожного вузла як доступного. Вартість кінцевого вузла встановлюється на 0 та сам кінцевий вузол додається у відкритий список.

Крок 2: починається цикл, який створює перший можливий шлях від початкового вузла до кінцевого. Для цього використовується функція пошуку шляху, яка починає з найближчого вузла з мінімальним значенням ключовою функції.

Крок 3: перевіряється кожен сусід поточного вузла, з яких обирається наступний прийнятний вузол.

Крок 4: будується шлях із оброблених прйнятних вузлів, які пов’язуються між собою зворотними вказівниками. Такий підхід дозволяє завжди повернутися до будь-якого попереднього вузла, якщо шлях зустріне необхідну перешкоду.

Крок 5: після побудови першого доцільного шляху починається рух об’єкта. Під час руху працює цикл, який постійно перевіряє поточну вартість руху через сусідів. Якщо під час цієї перевірки встановиться, що з’явилась якась перешкода, або кращий напрямок руху, то початковий сформований шлях буде скоректовано відповідною функцією.

Крок 6: для фіксації зміни вартості шляху перераховується початкова вартість руху між вузлами $c(N, M)$, що дозволяє завжди тримати останню достовірну інформацію про вартість переходу між вузлами.

Наведені кроки 5 та 6 повторюються до того моменту, доки об’єкт не досягне кінцевого вузла.

2.4 Алгоритм Theta*

Алгоритм Theta* правильний (тобто знаходить лише шляхи від початкової вершини до вершини цілі, які не мають повного блокування шляху) і повний (тобто знаходить шлях від початкової вершини до вершини цілі, якщо така існує).

Якщо існує не заблокований шлях між двома вершинами, то існує і не заблокований шлях за сіткою між однаковими двома вершинами. Не заблокований шлях між двома вершинами існує, якщо не заблокований шлях $[s_0,..., s_n]$ існує між однаковими двома вершинами. Розглянемо будь-який відрізок $s_k s_{k+1}$ даного шляху. Якщо сегмент шляху горизонтальний або вертикальний, тоді розглянемо не заблокований шлях сітки від вершини s_k до вершина s_{k+1}, яка збігається з відрізком шляху. Також розглянемо послідовність $(b_0, ..., b_m)$ не заблокованих вузлів, усередині яких проходить сегмент шляху. Будь-які дві послідовні клітинки b_j і b_{j+1} ділять принаймні одну вершину s_{j+1}, оскільки вузли або діляться ребром, або

розташовані по діагоналі. (Якщо вони мають більше однієї вершини, обирається будь-яка з них.) Розглянемо шлях сітки $[s_0 = s_k, s_1, \ldots, s_{m+1} = s_{k+1}]$. Цей шлях сітки від вершини s_k до вершини s_{k+1} не заблокований, оскільки будь-які дві послідовні вершини на ньому є кутами однієї і тієї ж не заблокованої комірки і, таким чином, є видимими сусідами. При повторенні цієї процедури для кожного відрізка шляху та об'єднанні отриманого результату отримаємо не заблокований шлях на сітки від вершини s_0 до вершини s_n. (Якщо кілька послідовних вершин на шляху сітки ідентичні, тоді всі, крім однієї, можна видалити зі шляху) [20].

У будь-який момент під час виконання алгоритму Theta*, слідуючи за попередніми пов’язаними вузлами шляху з будь-якої вершини у відкритих або закритих списках до початкової вершини виникає не заблокований шлях від початкової вершини до цієї вершини в зворотному напрямку. Це правило виконується і таким чином будь-яка попередня вершина у відкритому або закритому списках також знаходиться у відкритому або закритому списках. Це твердження справедливо на початку тому, що початкова вершина є єдиною вершиною в відкритому або закритому списках. Але твердження продовжує виконуватись, коли будь-яка вершина змінює свою попередню вершину або змінює належність до відкритого або закритого списків. Як тільки вершина стає членом відкритого або закритого списків, вона продовжує бути таким членом. Вершина може стати членом у відкритому або закритому списках лише тоді, коли алгоритм Theta* оновлює деяку вершину s і оновлює значення вартості шляху g та попередній елемент не оновленого видимого сусіда s. Таким чином, вершина s знаходиться у закритому списку, а її попередній елемент – у відкритому або закритому списках відповідно до припущення. Таким чином, слідуючи за попередніми елементами з вершини s (або її попередніх елементів) до початкової вершини отримуємо не заблокований шлях від початкової вершини до вершини s (або її попередньої відповідно) в зворотному напрямку відповідно. Якщо алгоритм Theta* оновлює вершину s відповідно до одного шляху, тоді твердження продовжує виконуватись тому, що вершина s і її сусід на шляху видимі один одному і відрізок шляху між цими вершинами таким чином розблокується. Якщо алгоритм Theta* оновлює вершину s відповідно до

іншого шляху, тоді твердження продовжує виконуватися, оскільки алгоритм Theta* явно перевіряє, що шлях відрізку від попередньої вершини до вершини s не заблоковано. Немає інших способів, за допомогою яких попередня вершина може змінюватися.

Алгоритм Theta* припиняє роботу і надає не заблокований шлях із початкової вершина до вершини цілі, якщо такий шлях існує. В іншому випадку алгоритм Theta* завершує роботу та повідомляє про не існуючий не заблокований шлях. Наступні властивості використовують той факт, що алгоритм Theta* завершується, якщо відкритий список порожній або оновлює вершину цілі. Початкова вершина спочатку знаходиться в відкритому списку. Будь-яка інша вершина спочатку не є ні у відкритому, ні у закритому списку. Вершину, що не у відкритому списку можна вставляти у обидва відкритий та закритий списки. Вершину з відкритого списку можна видалити з відкритого списку та вставити її до закритого списку. Вершина в закритому списку залишається в закритому списку.

Якщо алгоритм Theta* припиняється, то він оновлює одну вершину у відкритому списку протягом кожної ітерації. У процесі він видаляє вершину з відкритого списку, а потім ніколи не може вставити її знову у відкритий список. Оскільки число вершин є кінцевим, відкритий список з часом стає порожнім і алгоритм Theta* повинен завершити роботу, якщо він не завершився раніше.

Якщо алгоритм Theta* припиняється, оскільки його відкритий список порожній, тоді немає існуючого не заблокованого шляху від початкової вершини до вершини цілі. Припустимо, що існує не заблокований шлях від початкової вершини до вершини цілі. Тоді алгоритм Theta* не припиняється, оскільки його відкритий список порожній. Таким чином, припустимо також, що алгоритм Theta* завершується, оскільки його відкритий список порожній. Тоді, існує не заблокований шлях на сітці $[s_0 = s_{start}, \ldots, s_n = s_{goal}]$ від початкової вершини до кінцевої вершини. Обрана вершину s_i так, щоб бути першою вершиною на шляху сітки, якої немає у закритому списку, коли алгоритм Theta* припиняється. Вершина цілі відсутня у закритому списку коли алгоритм Theta* припиняється, оскільки алгоритм Theta* в іншому випадку припинив би свою роботу оновивши вершину

цілі. Отже, вершина s_i існує. Вершина s_i не є початковою вершиною з самого початку, ця вершина в іншому випадку була б у відкритому списку, і алгоритм Theta * не міг би закінчитися, оскільки його відкритий список порожній. Таким чином, вершина s_i має попередника на шляху сітки. Цей попередник є у закритому списку, коли алгоритм Theta* закінчується з вершини s_i – перша вершина сітки на шляху, якої немає у закритому списку, коли алгоритм Theta* завершується. Коли алгоритм Theta* оновлює попередника, він додає вершину s_i до відкритого списку. Таким чином, вершина s_i все ще знаходиться у відкритому списку, коли алгоритм Theta* припиняється. Але тоді алгоритм Theta* не можна було б припинити, оскільки відкритий список порожній, що є суперечливістю.

Якщо алгоритм Theta* припиняється, оскільки оновлює вершину цілі, то формування шляху отримує не заблокований шлях від початкової вершини до вершини цілі, оскільки слідуючи за попередніми вершинами від вершини цілі до початкової вершини отримуємо не заблокований шлях із самої початкової вершини до цільової вершини в зворотному порядку.

Алгоритм Theta* не є оптимальним (тобто не гарантовано знайти справжні найкоротші шляхи), оскільки попередня вершина повинна бути або видимим сусідом вершини, або попередником видимого сусіда, що не завжди має місце для справжніх найкоротших шляхів.

Виходячи з представлених властивостей є можливість почати виконання алгоритму (див. рис. 2.3).

Крок 1: спочатку, як і в алгоритмі A*,створюються два списки – відкритий та закритий. Відкритий список зберігає усі вузлі, які доступні для переміщення, закритий список зберігає усі вузли недоступні для переміщення. Спочатку закритий список не тримає в собі жодних елементів. У відкритий список додається початковий вузол.

Крок 2: розраховується довжина шляху f від початкового елементу до кінцевого. Вартість початкового вузла встановлюється рівною нулю, тому що об’єкт не може повернутися до початкової позиції після початку руху.

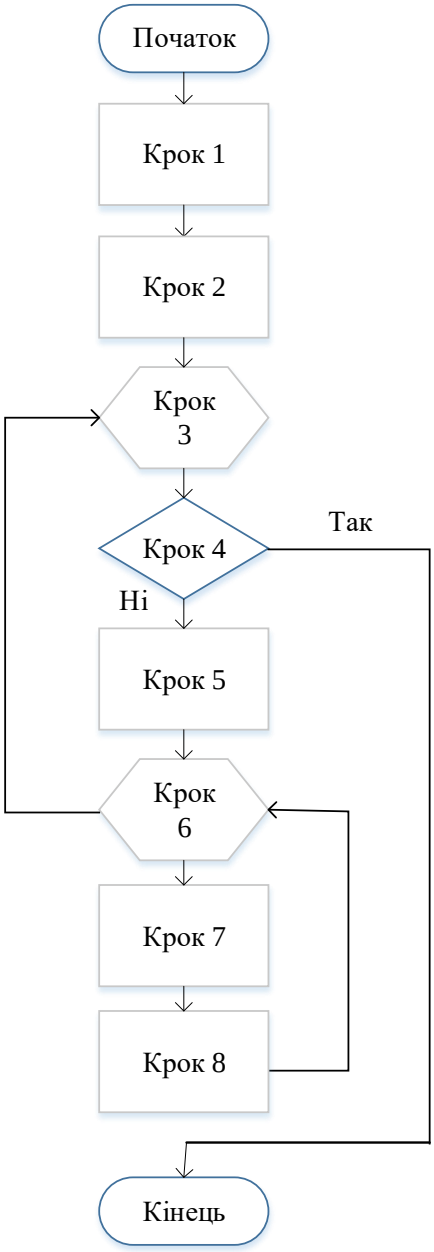

Рисунок 2.3 – Алгоритм пошуку шляху Theta*

Крок 3: відкривається цикл, який обробляє усі елементи з відкритого списку до моменту, коли даний список стане порожнім. Першим елементом перевірки в циклі стає елемент з відкритого списку з найменшою вартістю переходу. Під час першої ітерації циклу таким елементом є стартовий вузол, тому що інші елементи не були перевірені.

Крок 4: відбувається перевірка, яка вирішує чи був досягнутий кінцевий вузол за час роботи циклу. Така перевірка порівнює поточний вузол у циклі та кінцевий вузол. Якщо вузли співпадають, то виконується функція побудови повного шляху та

алгоритм припиняє свою роботу. Функція побудови повного шляху зберігає кожний пройдений вузол від кінцевого до початкового, що дозволяє відновити увесь пройдений шлях.

Крок 5: якщо кінцевий вузол не досягнутий, то поточний вузол, що перевіряється, видаляється зі списку відкритих вузлів та додається до списку закритих вузлів. Така операція дозволяє алгоритму завжди рухатися вперед не перевіряючи повторно вже пройдені вузли.

Крок 6: починається цикл перевірки усіх сусідніх вузлів до поточного вузла. Якщо сусідній вузол не зберігається у закритому списку, то можна почати його перевірку. Для початку розраховується вартість пройденого шляху від поточного сусіднього вузла до кінцевого.

Крок 7: якщо сусід не зберігається у відкритому списку, то він додається до цього списку. Якщо поточний сусід вже зберігаються у відкритому списку, то відбувається порівняння поточного сусіду із іншим сусідом з відкритого списку. Обирається сусід з найдешевшою вартістю переходу g від початкового вузла до поточного.

Крок 8: виконується перевірка видимості між попередником поточної вершини та обраним сусідом, якщо існує прямий безперешкодний шлях між даними вершинами, то поточна вершина прибирається зі шляху задля скорочення необхідного руху за шляхом.

Таким чином виконується перевірка кожного сусіда кожного наступного вузла та при наявності можливості скорочення шляху зменшується довжина шляху та необхідна кількість для проміжних вузлів.

2.5 Алгоритм Данцига

Для початку роботи алгоритму Данцига перераховуються вершини початкової сітки з цілими числами від 1 до N і позначаються довжиною найкоротшого шляху

від вершини *i* до вершини *j*, в якій дозволено використовувати перші вершини сітки як проміжні *m*. Також існує матриця, що складається зі значень d^m_{ij}, ця матриця має розмірність *MxM*. Виходячи з цих даних потрібно визначити матрицю, елемент *(i, j)* якої визначає довжину найкоротшого шляху від вершини *i* до вершини *j*. В алгоритмі Данцига кожна матриця визначається з подібної попередньої матриці, в разі чого формується фінальна матриця [21].

Алгоритм Данцига має в собі декілька ідей. По-перше, кожна нова обчислювана матриця містить на один рядок і на один стовпець більше, ніж її попередня матриця. Елементи матриці, які не включені в останній рядок і стовпець (кількість таких елементів дорівнює $(m-1)^2$). Що стосується інших елементів, які відносяться до останніх рядку та стовпцю фінальної матриці, то вони визначаються з урахуванням деяких деталей. Найкоротший шлях від вершини *i* до вершини *m* (або навпаки), в якому лише перші *m* вершин графіка можуть бути використані як проміжні, не може мати вершину *m* серед проміжних, оскільки будь-який контур в початковій сітці має невід'ємну довжину. Завдяки цьому такий найкоротший шлях від вершини *i* до вершини *m* повинен мати в якості першої частини найкоротший шлях від вершини *i* до якоїсь вершини *j*, яка може бути використана як проміжна, а друга частина – найкоротша дуга, що веде від вершини *j* до вершини *m* (розглядаються лише такі вершини *j*, для яких існує хоча б одна дуга, що веде від *j* до *m*). Аналогічно, найкоротший шлях від вершини *m* до вершини *i*, в якому дозволено використовувати лише перші *m* вершин графа як проміжні, повинні мати свою першу частину найкоротшою дугою, що веде від вершини *m* до якоїсь вершини *j*, а друга частина – найкоротший шлях від вершини *j* до. вершина *i*, що дозволяє використовувати лише *(m-1)* перших вершин як проміжні (авжеж, розглядаються лише ті вершини *j*, для яких існує принаймні одна дуга, що веде від *m* до *j*). Важливо зазначити, що величини останнього кутового елементу матриці слід встановлювати рівними нулю.

Обчислювальна складність алгоритму – це кількість елементарних операцій, які виконує алгоритм для вирішення конкретної задачі. Складність залежить не тільки від розміру вхідних даних, а й від самих даних. Очевидно, що чим

складніший обчислювальний алгоритм, тим більше часу та обчислювальних ресурсів йому знадобиться для виконання.

Розрізняють часову та просторову складність. Перша визначає час, необхідний для вирішення заданої задачі за допомогою цього алгоритму, а друга визначає обсяг необхідних ресурсів (пам’яті) за тих самих умов.

Кожен обчислювальний алгоритм можна класифікувати за одним із двох класів складності. У більшості алгоритмів обчислювальні витрати лінійно зростають із збільшенням розмірності. Наприклад, час, необхідний для виконання операції, прямо пропорційний розміру оброблюваних даних. Якщо його подвоїти, тоді і витрати часу також подвояться. У деяких алгоритмів є проблеми з масштабуванням, їх складність експоненційно залежить від розмірності даних. Тому вони, як правило, неефективні при роботі з великими наборами даних. Прикладом є проблема пошуку шляху на матриці, час, витрачений на який зростає в геометричній прогресії із збільшенням розмірності матриці.

Можливо легко підрахувати обчислювальну складність алгоритму Данцига. Розрахунок кожного нового елемента матриці D^m включає 2 операції (додавання та порівняння). Кожна матриця містить MxM елементів, тому на кожній ітерації необхідно виконувати $2N^2$ операцій. Всього в процесі алгоритму обчислюється M таблиць, тобто обчислювальна складність алгоритму Данцига становить $2N^3$ операцій.

Беручи до уваги наведені властивості, можливо почати виконання алгоритму Данцига (див. рис. 2.4).

Крок 1: усім вершинам початкової сітки призначають порядковий номер від 1 до N, де N – кількість вершин.

Крок 2: формується початкова матриця розмірністю NxN, яка заповнюється значеннями найкоротшої вартості переходу між усіма вершинами. При відсутності такого переходу, вартість встановлюється як невизначена.

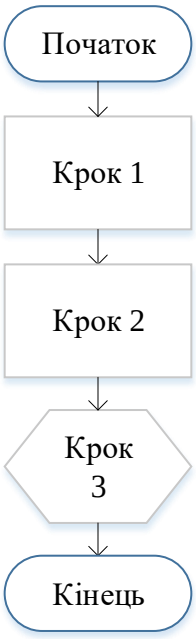

Рисунок 2.4 – Алгоритм пошуку шляху Данцига

Крок 3: далі формується наступна матриця розмірністю *1x1*, елементи якої визначаються виходячи з початкової матриці. Надалі працює цикл, у якому повторюється ця процедура для кожної матриці наступної розмірності.

Алгоритм продовжує розраховувати усі можливі матриці доти, доки остання матриця не буде повністю заповнена. Такий результат алгоритму дозволяє отримати усі найкоротші шляхи, доступні на сітці.

2.6 Алгоритм Флойда

Ключовою ідеєю алгоритму Флойда є розподіл процесу пошуку найкоротшого шляху між будь-якими двома вершинами на декілька фаз, що будуть повторюватись з невеликими змінами [9].

Нумеруються вершини, починаючи від *1* до *n*. Вводиться матриця відстаней *d*[][]. До *k*-ї фази ($k = 1, ..., n$), *d*[i][j] для будь-яких вершин *i* та *j* зберігає довжину

найкоротшого шляху між вершиною *i* та вершиною *j*, яка містить лише вершини *{1, 2 , ..., k - 1}* як внутрішні вершини шляху.

Перед *k*-ю фазою значення *d*[i][j] дорівнює довжині найкоротшого шляху від вершини *i* до вершини *j*, якщо цьому шляху дозволено входити лише у вершину з числами, меншими за *k* (початок і кінець шляху не обмежуються цією властивістю).

Ця властивість відповідає першій фазі. Для $k = 0$ ми можемо заповнити матрицю $d[i][j] = w_{ij}$, якщо між *i* та *j* існує ребро з вагою w_{ij} та $d[i][j] = \infty$, якщо ребро не існує. На практиці ∞ буде деяким великим значенням, що є вимогою до алгоритму.

Припустимо, що почалась фаза *k*, і необхідно обчислити матрицю *d*[][] так, щоб вона відповідала вимогам для *(k + 1)* -ї фази. Ми повинні зафіксувати відстані для деяких пар вершин *(i, j)*. Є два принципово різні випадки:

– найкоротший шлях від вершини *i* до вершини *j* з внутрішніми вершинами з множини *{1, 2, ..., k}* збігається з найкоротшим шляхом з внутрішніми вершинами з множини *{1, 2, ..., k - 1}*. У цьому випадку *d*[i][j] не зміниться під час переходу;

– найкоротший шлях із внутрішніми вершинами від *{1, 2, ..., k}* коротший. Це означає, що новий, коротший шлях проходить через вершину *k*. Також існує можливість розділити найкоротший шлях між *i* та *j* на два шляхи: шлях між *i* та *k* та шлях між *k* та *j*. Обидва ці шляхи використовують лише внутрішні вершини *{1, 2, ..., k - 1}* і є найкоротшими такими шляхами в цьому відношенні. Тому проводиться обчислювання довжини цих шляхів раніше, і тепер можливо обчислити довжину найкоротшого шляху між *i* та *j* як

$$d[i][k] + d[k][j] \tag{2.12}$$

Поєднуючи ці два випадки, виявляється, що є можливість перерахувати довжину всіх пар *(i, j)* у *k*-й фазі наступним чином:

$$d_{new}[i][j] = min\,(d[i][j], d[i][k] + d[k][j]) \tag{2.13}$$

Таким чином, вся робота, яка потрібна в *k*-й фазі, полягає в ітерації всіх пар вершин і перерахунку довжини найкоротшого шляху між ними. В результаті після *n*-ої фази значення *d*[i][j] в матриці відстані є довжиною найкоротшого шляху між *i* та *j*, або дорівнює ∞, якщо шлях між вершинами *i* та *j* не існує.

Для цього алгоритму не потрібно створювати окрему матрицю відстані d_{new}[][] для тимчасового зберігання найкоротших шляхів *k*-ї фази, тобто всі зміни можна вносити безпосередньо в матрицю *d*[][] в будь-який фазі. Насправді в будь-якій *k*-й фазі максимально покращується відстань будь-якого шляху в матриці відстані, отже, неможливо погіршити довжину найкоротшого шляху для будь-якої пари вершин, які повинні бути розраховані в *(k + 1)*-тій фазі або пізніше.

На основі наведених властивостей, можливо почати виконання алгоритму Флойда (див. рис. 2.5).

Крок 1: усім вершинам початкової сітки призначають порядковий номер від *1* до *N*, де *N* – кількість вершин.

Крок 2: створюється одна матриця для зберігання найкоротших шляхів між усіма вузлами сітки.

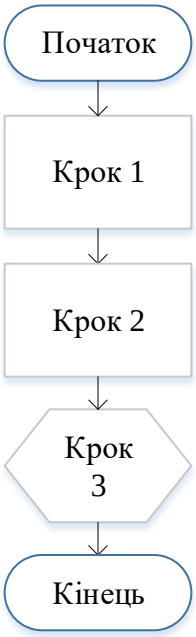

Рисунок 2.5 – Алгоритм пошуку шляху Флойда

Крок 3: починається цикл, під час виконання якого у кожній ітерації створюється новий шлях для кожної пари вузлів. Також кожен новий шлях порівнюється з попереднім, що дозволяє завжди зберігати найкоротший шлях між двома вузлами.

Алгоритм припиняє роботу після підрахунку усіх можливих варіантів найкоротших шляхів та у результаті виходить матриця, що містить тільки по одному найкоротшому шляху для кожної пари вузлів.

2.7 Алгоритм Дейкстри

Для використання алгоритму Дейкстри необхідно створити масив *d[]*, де для кожної вершини v ми зберігаємо поточну довжину найкоротшого шляху від s до v у *d[v]*. Спочатку $d[s] = 0$, а для всіх інших вершин ця довжина дорівнює нескінченності. У реалізації достатньо велике число (яке гарантовано перевищує будь-яку можливу довжину шляху) вибирається як нескінченність [10].

Крім того, створюється масив бінарних значень *u[]*, який зберігає для кожної вершини v, чи є вона позначена. Спочатку всі вершини не позначені:

Алгоритм Дейкстри працює для n ітерацій. На кожній ітерації вершина v вибирається як непомічена вершина, яка має найменше значення *d[v]*. На першій ітерації буде вибрано початкову вершину s.

Виділена вершина v позначена. Далі з вершини виконуються розрахунки: розглядаються всі ребра сітки між вершинами (v, to), і для кожної вершини алгоритм намагається покращити значення *d[to]*. Якщо відома довжина поточного переходу, тоді розрахунок проводиться за наступною формулою:

$$d\,[to] \;=\; min\,(d\,[to], d\,[v] \;+\; len), \qquad (2.14)$$

де *len* – довжина поточного переходу.

Після розгляду всіх таких переходів поточна ітерація закінчується. Нарешті, після n ітерацій усі вершини будуть позначені, а алгоритм завершується. Стверджується, що знайдені значення $d[v]$ – це довжини найкоротших шляхів від s до всіх вершин v.

Якщо деякі вершини недосяжні з початкової вершини s, значення $d[v]$ для них залишаться нескінченними. Останні кілька ітерацій алгоритму виберуть ці вершини, але корисної роботи для них не буде зроблено. Отже, алгоритм можна зупинити, як тільки вибрана вершина має нескінченну відстань до неї.

Зазвичай потрібно знати не тільки довжину найкоротших шляхів, але й самі короткі шляхи. Потрібно віршувати, як зберегти достатньо інформації для відновлення найкоротшого шляху від s до будь-якої вершини. Можна створити масив попередніх вершин $p[]$, в якому для кожної вершини $v \neq s$, p[v] є передостанньою вершиною за найкоротшим шляхом від s до v. Тут використовується той факт, що якщо пройти найкоротший шлях до будь-якої вершини v і видалити v з цього шляху, отримаємо шлях, що закінчується на вершину p[v], і цей шлях буде найкоротшим для вершини p[v]. Цей масив попередніх вершин можна використовувати для відновлення найкоротшого шляху до будь-якої вершини. Починаючи з v, багаторазово беручи попередника поточної вершини, поки не буде отримана початкова вершина s, можливо отримати необхідний найкоротший шлях з вершинами, перерахованими в зворотному порядку.

Побудова цього масиву попередників дуже проста: для кожного успішного розрахунку довжини поточного переходу, тобто коли для деякої вибраної вершини v відбувається покращення відстані до кінцевої вершини, проводиться оновлення попередньої вершини до кінцевої на вершину v.

Основним твердженням, на якому базується правильність алгоритму Дейкстри, є наступне: після того, як будь-яка вершина v стане позначеною, поточна відстань до неї d[v] є найкоротшою і більше не буде змінюватися.

Для першої ітерації це твердження очевидно: єдиною позначеною вершиною є s, а відстань до d[s] $= 0$ справді є довжиною найкоротшого шляху до s. Тепер

припустимо, що це твердження справедливо для всіх попередніх ітерацій, тобто для всіх вже позначених вершин це не порушується після завершення поточної ітерації. Нехай v – вершина, вибрана в поточній ітерації, тобто v – вершина, позначена алгоритмом. Тепер потрібно розрахувати d[v], щоб дізнатися чи дорівнює вона довжині найкоротшого шляху до неї l[v].

Розглянемо найкоротший шлях P до вершини v. Цей шлях можна розділити на дві частини: $P1$, яка складається лише з позначених вузлів (принаймні початкова вершина s є частиною $P1$), і решта шляху $P2$ (вона може включати позначену вершину, але вона завжди починається з непозначеної вершини). Позначимо першу вершину шляху $P2$ як p, а останню вершину шляху $P1$ як q.

Спочатку перевіряється твердження для вершини p: на одній з попередніх ітерацій ми вибрали вершину q і здійснили пошук шляхів від неї. Оскільки (внаслідок вибору вершини p) найкоротший шлях до p – це найкоротший шлях до q плюс перехід (p,q), пошук шляху від q встановлює значення d[p] на довжину найкоротшого шляху l[q].

Оскільки вартості переходів невід'ємні, довжина найкоротшого шляху l[p] (що дорівнює d[p]) не перевищує довжини l[v] найкоротшого шляху до вершини v. Враховуючи, що l[v] $\leq d$[v] (оскільки алгоритм Дейкстри не міг знайти коротший шлях, ніж найкоротший з можливих), ми отримуємо нерівність:

$$d\,[p] \;=\; l\,[p] \;\leq l\,[v] \;\leq d\,[v] \tag{2.15}$$

З іншого боку, оскільки обидві вершини p і v не позначені, а поточна ітерація обрала вершину v, а не p, ми отримуємо ще одну нерівність:

$$d\,[p] \;\geq d\,[v] \tag{2.16}$$

З цих двох нерівностей робимо висновок, що d[p] $= d$[v], а потім із раніше знайдених рівнянь отримуємо:

$$d[v] = l[v] \tag{2.17}$$

Після наведення усіх необхідних даних для алгоритму (див. рис. 2.6), можливо почати його виконання.

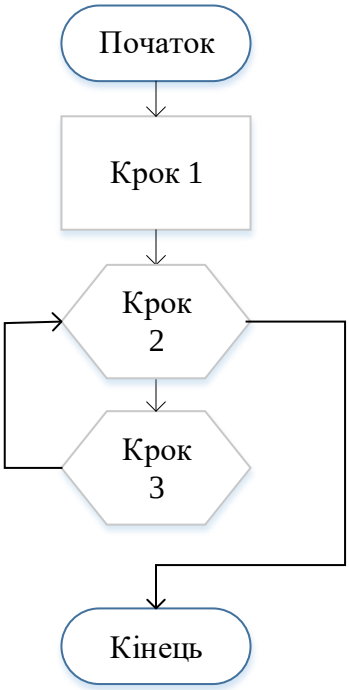

Рисунок 2.6 – Алгоритм пошуку шляху Дейкстри

Крок 1: створюється матриця, яка зберігатиме вартості усіх шляхів між вузлами. Вартість початкового вузла встановлюється нулем. Вартості всіх інших вузлів встановлюються числом, що представляє собою нескінченно велике число.

Крок 2: починається цикл, у якому спочатку обирається вузол з найменшою вартістю шляху серед помічених вузлів. Якщо вартість обраного вузлу дорівнює нескінченно великому числу, то алгоритм припиняє роботу.

Крок 3: якщо алгоритм продовжує роботу, то виконується розрахунок нової поточної довжини переходу, якщо це можливо. При порівнянні результату зберігається найменша відстань з двох відомих.

Алгоритм продовжує роботу, доки не залишиться непомічених вершин. Це буде означати, що усі можливі шляхи перевірено і розраховано оптимальний шлях переміщення з початкового вузла до кінцевого.

2.8 Алгоритм Лі

Алгоритм Лі приймає на вхід сітку вузлів та початкову вершину s. Алгоритм поширюється на сітці хвилями: на нульовому кроці відомий лише початковий вузол s. На кожному кроці перевіряються усі сусіди вузлів із минулого кроку. За одну ітерацію алгоритму хвиля поширюється на відстань одного вузла [11].

Також, алгоритм можна сформулювати наступним чином: створюється список q, який буде містити вершини, що підлягають обробці, при цьому кожна вершина зберігає номер ітерації, на якій була оброблена.

Початковий вузол s зберігається до списку та встановлюється як нульовий. Потім, виконується цикл, доки список не порожній, і в кожній ітерації обирається вершина з передньої частини списку. Переглядаються усі переходи, що можливі з цієї вершини, і якщо деякі з цих переходів ведуть до вершини, яка ще не перевірялась, то вона додається до списку.

У кожній ітерації усі наступні сусіди, що перевіряються, зберігають номер ітерації, на якій вони були оброблені, тобто на 1 більший, ніж їх попередня вершина. Отже кожна вершина має номер, що дозволяє відновити знайдений шлях способом віднімання одиниці від номеру, що зберігається у вершині.

Як результат, коли список порожній, масив перевірених вершин містить усі вершини, до яких можна дістатися з початкової вершини s, причому кожна вершина досягається найкоротшим шляхом. Також можливо розрахувати довжини найкоротших шляхів (для цього просто потрібно зберегти масив довжин шляхів), а також зберегти інформацію для відновлення всіх цих найкоротших шляхів (для

цього необхідно підтримувати масив попередніх вузлів, який зберігає для кожної вершини вершину, з якої до неї можливо перейти).

Після наведення усіх необхідних даних для алгоритму (див. рис. 2.7), можливо почати його виконання.

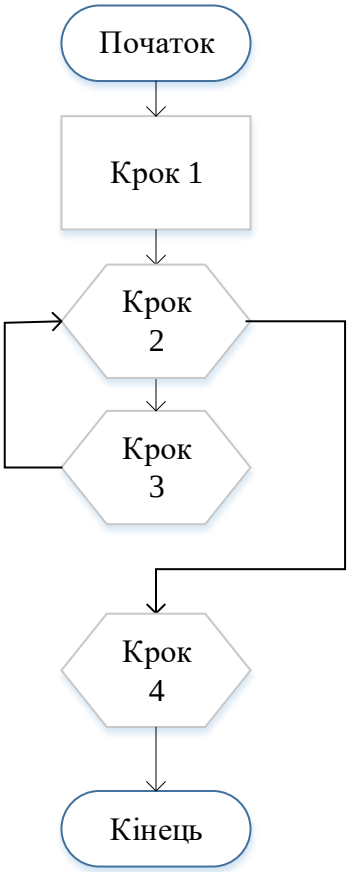

Рисунок 2.7 – Алгоритм пошуку шляху Лі

Крок 1: початкова вершина додається до списку вершин, представлених для оброблення. Також у масиві станів вузлів помічається, що цей вузол перевірений.

Крок 2: починається цикл, який перевіряє усі вершини зі списку, доки список на стане порожнім. Після обрання вузла для обробки, цей вузол видаляється зі списку.

Крок 3: починається цикл для обраного вузла, який перевіряє усіх його сусідів. Для кожного сусіда зберігається значення, що відповідає тому, як далеко він знаходиться від початкового вузла. Такий підхід хвилями розповсюджує номер ітерації між усіма сусідами вузлів, перевірених на попередній ітерації.

Крок 4: після перевірки усіх вузлів починається цикл, який дозволяє відновити знайдений шлях від початкової вершини.

Алгоритм завершується після повного конструювання знайденого шляху.

2.9 Вибір алгоритмів для порівняння результатів

Усі наведені алгоритми мають свої переваги та недоліки у загалі, але не всі алгоритми підходять для виконання поставленої мети. Після розгляду багатьох можливих принципів роботи алгоритмів, можливо обрати тільки ті, які відповідатимуть технічним вимогам.

Алгоритм A* дозволяє знайти найкоротший шлях не використовуючи багато обчислювальних ресурсів. Хоча він і потребує повну сітку з перешкодами для обчислення, таку сітку можливо змоделювати виходячи з карти місцевості між початковою точкою та кінцевою. Тому цей алгоритм можливо розглянути для використання у безпілотних польотах.

Алгоритм D* працює майже як і алгоритм A*, але з однією великою перевагою: йому не потрібно знати про усі перешкоди на шляху до цілі. Цей алгоритм може сканувати навколишнє середовище на предмет перешкод та динамічно оновлювати шлях, що дозволяє обминати усі перешкоди, навіть якщо їх не було на момент початку переміщення. Тому цей алгоритм підходить для використання у безпілотних польотах.

Алгоритм Theta* схожий на A*, але окрім цього дозволяє скорочувати шлях, якщо вузли сітки між собою не мають перешкод. Такий підхід корисний, якщо є максимально точні дані про перешкоди на шляху на початок переміщення. Для планування безпілотних польотів цей спосіб не є достатньо зручним, тому що не завжди можна отримати точну карту перешкод, що може спровокувати скорочення шляху в тому місці, де це не є можливим.

Алгоритм Данцига не підходить для безпілотних польотів, тому що потребує велику кількість обчислень. Він потребує окрему матрицю з даними для кожного кроку прорахунку шляху, що не є ефективним при роботі з великими об’ємами сітки.

Алгоритм Флойда не є ефективним, тому що він для кожної вершини перераховує вартості переміщення між вузлами. Такий підхід є достатньо точним, але потребує зберігати велику кількість даних для обчислення.

Алгоритм Дейкстри насамперед є алгоритмом пошуку шляхів до усіх існуючих вузлів. При виконанні доставки завжди відомий кінцевий вузол шляху, тому обробка усіх можливих шляхів на сітці є зайвим використанням обчислювальних ресурсів.

Алгоритм Лі можливо використовувати при безпілотних польотах, тому що поширення інформації про сусідів у вигляді декількох хвиль дозволяє перевіряти наявність перешкод на шляху. Такий спосіб дозволяє знаходити навколишні перешкоди за допомогою радіохвильового обладнання, що дає можливість перепланування шляху під час переміщення.

Отже, розглянувши недоліки та переваги наведених алгоритмів було обрано реалізувати, отримати та порівняти дані наступних алгоритмів: A*, D* та алгоритм Лі.

3 ПОРІВНЯННЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ АЛГОРИТМІВ

3.1 Результати виконання алгоритмів

З метою порівняльного аналізу та подальшої обробки проведено математичне моделювання запропонованих алгоритмів, розроблено програмний застосунок з використаням об’єктно-оорієнтованого підходу та мови програмування С++ (Додаток Б). За допомогою розробленого застосунку оцінені основні характеристики алгоритмів: час виконання, довжина шляху, кількість оброблених вузлів.

Головною метою кожного алгоритму є пошук найкоротшого шляху від початкової точки руху до кінцевої точки. Кожен з алгоритмів має декілька однакових характеристик, які визначають ефективність їх роботи. Основною характеристикою є t_a – час виконання алгоритму, який вимірюється в секундах. Іншою характеристикою можна вважати довжину шляху, або кількість вузлів, пройдених обраним шляхом n_p, вимірюється в одиницях, що відображають кількість вузлів. Останньою характеристикою, яку обрано для перевірки є кількість опрацьованих алгоритмом вузлів n_a, що також вимірюється в одиницях вузлів [22].

Було проведено перевірку обраних алгоритмів на сітках різного розміру та з різним відсотком перешкод. Такий підхід дозволить перевірити різноманітні умови, що можуть виникнуть під час реальної роботи алгоритмів на безпілотних літальних апаратах.

Результати виконання алгоритму пошуку шляху A* наведені у таблиці 3.1. Сформована таблиця відображає час виконання алгоритму за різних умов.

Таблиця 3.1 – Час виконання алгоритму A*

Час виконання		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	0.344	0.328	0.312	1.141
	25x25x25	1.015	1.172	1.468	2.218
	50x50x50	2.547	3.14	2.765	10.625
	100x100x100	7.421	7.562	10.937	42.217

У наступній таблиці 3.2 відображені результати про кількість вузлів у сформованого алгоритмом A* шляху.

Таблиця 3.2 – Довжина шляху в алгоритмі A*

Довжина шляху		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	10	10	10	18
	25x25x25	26	27	33	45
	50x50x50	52	56	56	76
	100x100x100	104	108	125	174

Далі у таблиці 3.3 наведені результати про загальну кількість оброблених вузлів алгоритмом A*.

Таблиця 3.3 – Кількість оброблених вузлів в алгоритмі A*

Кількість оброблених вузлів		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	11	11	11	42
	25x25x25	27	32	42	68
	50x50x50	55	67	66	230
	100x100x100	113	121	176	615

За отриманими результатами можливо зробити декілька важливих висновків про алгоритм A*:

– час пошуку оптимального шляху зростає пропорційно збільшенню кількості перешкод та розміру сітки;

– довжина оптимального шляху зростає пропорційно збільшенню кількості перешкод та розміру сітки;

– кількість оброблених вузлів небагато зростає пропорційно збільшенню кількості перешкод та розміру сітки, але відчутне збільшення оброблених вузлів виникає на сітці з великою кількістю перешкод.

Виходячи з попередніх висновків можливо побачити, що алгоритм A* має гарну масштабованість, тобто його ефективність не погіршується зі збільшенням даних, що необхідно обробити, але різко зменшується при великих кількостях перешкод.

Наступним відображені результати виконання алгоритму D*. На відміну від інших представлених алгоритмів, алгоритм D* має можливість перераховувати прийнятний шлях під час руху об’єкта. Тому для отримання реалістичних даних для алгоритму D* впроваджено генерацію непередбачених перешкод. Наведені характеристики відповідають різним розмірам сітки та відсотку присутніх перешкод. Сформована таблиця 3.4 відображає час виконання алгоритму за різних умов.

Таблиця 3.4 – Час виконання алгоритму D*

Час виконання		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	0.765	0.719	0.625	0.594
	25x25x25	4.577	4.968	5.516	5.25
	50x50x50	55.201	24.14	23.858	36.937
	100x100x100	115.306	256.79	141.746	1135.522

У наступній таблиці 3.5 відображені результати про кількість вузлів у сформованого алгоритмом D* шляху.

Таблиця 3.5 – Довжина шляху в алгоритмі D*

Довжина шляху		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	10	10	10	11
	25x25x25	26	27	33	34
	50x50x50	51	56	63	74
	100x100x100	104	108	125	178

Далі у таблиці 3.6 наведені результати про загальну кількість оброблених вузлів алгоритмом D*.

Таблиця 3.6 – Кількість оброблених вузлів в алгоритмі D*

Кількість оброблених вузлів		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	22	22	22	21
	25x25x25	52	58	74	85
	50x50x50	381	138	145	229
	100x100x100	212	722	311	1674

Виходячи з отриманих результатів, можливо зробити декілька висновків про алгоритм D*:

– час виконання сильно зростає при збільшенні розміру сітки;

– довжина шляху збільшується пропорційно збільшенню розміру сітки та кількості перешкод;

– кількість оброблених вузлів збільшується пропорційно збільшенню розміру сітки та кількості перешкод.

Враховуючи наведені висновки, можна відмітити, що ефективність алгоритму D* загалом на задовільному рівні, але, не зважаючи на оптимальність початкового шляху, можливе зменшення ефективності заради можливості перерахунку доцільного шляху відповідно до стану перешкод, який може змінюватися.

Насамкінець, наведені результати виконання алгоритму Лі. Наведені характеристики відповідають різним розмірам сітки та відсотку присутніх перешкод. Сформована таблиця 3.7 відображає час виконання алгоритму за різних умов.

Таблиця 3.7 – Час виконання алгоритму Лі

Час виконання		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	0.958	0.953	0.945	1.374
	25x25x25	2.66	2.77	2.856	3.201
	50x50x50	5.776	5.83	6.026	6.595
	100x100x100	13.887	13.889	13.691	14.051

У наступній таблиці 3.8 відображені результати про кількість вузлів у сформованого алгоритмом Лі шляху.

Таблиця 3.8 – Довжина шляху в алгоритмі Лі

Довжина шляху		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	10	10	10	14
	25x25x25	26	27	28	31
	50x50x50	52	53	55	62
	100x100x100	102	105	109	120

Далі у таблиці 3.9 наведені результати про загальну кількість оброблених вузлів алгоритмом Лі.

Таблиця 3.9 – Кількість оброблених вузлів в алгоритмі Лі

Кількість оброблених вузлів		Кількість перешкод			
		10%	20%	40%	60%
Розмір сітки	10x10x10	662	586	424	377
	25x25x25	14056	12513	9398	6236
	50x50x50	112314	99899	74791	50037
	100x100x100	900216	800334	600143	400376

Після отримання результатів виконання алгоритму Лі, можливо зробити декілька висновків:

– час виконання збільшується пропорційно до збільшення розміру сітки, але не сильно залежить від кількості перешкод;

– довжина оптимального шляху збільшується пропорційно до збільшення розміру сітки, але не сильно залежить від кількості перешкод;

– кількість оброблених вузлів дуже сильно збільшується пропорційно збільшенню розміру сітка, але зменшується зі збільшенням кількості перешкод.

Виходячи з отриманих висновків можливо визначити закономірність у роботі алгоритму Лі. Його ефективність залишається високою незважаючи на розмір сітки чи кількість перешкод. Але якщо обчислювальних ресурсів при використанні цього алгоритму буде замало, то його ефективність може знизитись, тому що цей алгоритму оброблює усі вузли сітки, які не є перешкодами.

3.2 Порівняння отриманих результатів

В результаті проведення різноманітних досліджень зазвичай отримується об’ємний набір даних, що спочатку представляє собою різноманітні параметри, відповідні до різних методів або об’єктів дослідження. Подібна інформація у її початковому стані несе лише статистичний характер, тобто рішення, проведені на

основі таких даних, будуть стосуватись лише об’єктів дослідження, яким відповідають ці дані. Але на цьому вивчення результатів не припиняється, після розгляду усієї доступної інформації може виникнути потреба у вирішенні, на якому об’єкті дослідження слід зосередитись. Для цього необхідно виконати порівняння отриманих результатів від кожного дослідженого об’єкта [23].

З метою полегшення вибору були розроблені різноманітні методи прийняття рішень. У сучасній теорії прийняття рішень вважається, що варіанти рішень характеризуються різними показниками своєї привабливості для того, хто приймає рішення. Ці показники називаються характеристиками якості. Усі вони служать критеріями вибору рішення. У переважній більшості реальних проблем існує безліч критеріїв. Ці критерії можуть бути незалежними або залежними.

Припустимо, що дві порівняні альтернативи мають різні оцінки для першої група критеріїв і однакові для другої групи. У теорії прийняття рішень критерії вважаються залежними від уподобань того, хто приймає рішення при порівнянні альтернатив, якщо вони змінюються залежно від значень однакових оцінок для другої групи критеріїв. На складність завдань прийняття рішень впливає також кількість критеріїв. При невеликій кількості критеріїв (скажімо, двох) проблема порівняння двох альтернатив досить проста, значення критеріїв можуть бути безпосередньо порівняні і буде обрана краща альтернатива. Завдяки великій кількості критеріїв завдання стає об’ємним для того, хто приймає рішення. Враховуючи велику кількість критеріїв, вони зазвичай можуть об’єднуватися в групи, що мають конкретне семантичне значення. Ці групи критеріїв, як правило, незалежні. Виявлення структури при багатьох критеріях робить процес прийняття рішень більш ефективним.

Використання критеріїв для вибору рішення для оцінки альтернатив вимагає визначення градацій значень критеріїв: найкращих, найгірших та проміжних оцінок. Іншими словами, існують шкали оцінок на основі критеріїв. При прийнятті рішень існують шкали безперервних та дискретних оцінок, шкали кількісних та якісних оцінок.

Процес прийняття рішень — це не одноразовий акт, а часто досить тривалий процес. Зазвичай в ньому виділяють три етапи: пошук інформації та постановка проблеми, побудова набору альтернатив та вибір найкращої альтернативи. На першому етапі збирається вся інформація, наявна на момент прийняття рішення: будуються фактичні дані, висновки експертів, математичні моделі, проводяться соціологічні опитування, визначаються погляди на проблему з боку активних груп, що впливають на її вирішення, формуються критерії вибору рішення тощо. Другий етап пов'язаний з визначенням того, що можливо, і що неможливо зробити в поточній ситуації, тобто визначити реалізацію варіантів рішення. І вже третій етап включає порівняння альтернатив та вибір найкращого рішення.

З перерахованих трьох етапів процесу прийняття рішень найбільша увага приділяється останньому етапу. Шляхи проходження етапів залежать не тільки від змісту проблеми прийняття рішення, а й від досвіду, звичок, особистого стилю особи, що приймає рішення, та його оточення.

Вибір невеликої кількості альтернатив із часто невизначеної кількості можливих варіантів дій на другому етапі вимагає всебічного аналізу цих варіантів. У той же час, аналіз великої кількості варіантів може бути досить грубим, але повинен, якщо це можливо, включати всі такі варіанти. На третьому етапі, навпаки, потрібно ретельно проаналізувати та порівняти лише невелику кількість альтернатив, які вже були сформульовані в явній формі. Такий погляд на процес прийняття рішень використовується при розробці методів підтримки прийняття рішень.

Існує велика кількість методів прийняття рішень, однак для спрощення проведення порівняння результатів буде використано один з найпростіших методів. Задачі максимізації та мінімізації значення дозволяють отримати необхідну інформацію про результати дослідження шляхом порівняння даних за наявними критеріями. Результат розв’язання подібних задач зазвичай дозволяє отримати оптимальні рішення виходячи з отриманих даних.

Метою дослідження, що проводиться, є знаходження алгоритмів пошуку оптимальних шляхів. Оскільки присутні декілька різних критеріїв, за якими можна вважати алгоритм кращим за інші, то розглянемо кожен з цих критеріїв окремо, як

основний, при чому інші критерії не впливатимуть на результат вибору за основним критерієм. Отримані раніше результати містять інформацію про роботу алгоритмів на сітках різного розміру при різному відсотку перешкод. Для зменшення даних, необхідних для подальшого порівняння, обрано порівнювати результати на сітках, розміром 25x25x25 та 50x50x50 вузлів при будь-яких рівнях перешкод *(o)*.

Першим розглянемо критерій часу виконання алгоритму t_a. Для знаходження оптимального результату необхідно розв'язати наступну задачу:

$$k(t_a) \to min \tag{3.1}$$

Результати часу виконання кожного алгоритму наведені у таблиці 3.10.

Таблиця 3.10 – Порівняння часу виконання

Час виконання	Сітка розміром 25x25x25			
	10%	20%	40%	60%
Алгоритм A*	1.015	1.172	1.468	2.218
Алгоритм D*	4.577	4.968	5.516	5.25
Алгоритм Лі	2.66	2.77	2.856	3.201
Час виконання	Сітка розміром 50x50x50			
	10%	20%	40%	60%
Алгоритм A*	2.547	3.14	2.765	10.625
Алгоритм D*	55.201	24.14	23.858	36.937
Алгоритм Лі	5.776	5.83	6.026	6.595

Знайдемо мінімальний час виконання алгоритму на сітці, розміром 25x25x25, при мінімальній та максимальній кількості перешкод o:

$$\min_{o \to min} k_{25}(t_a) = 1.015 \tag{3.2}$$

$$\min_{o \to max} k_{25}(t_a) = 2.218 \tag{3.3}$$

Знайдемо мінімальний час виконання алгоритму на сітці, розміром 50x50x50, при мінімальній та максимальній кількості перешкод:

$$\min_{o \to min} k_{50}(t_a) = 2.547 \quad (3.4)$$

$$\min_{o \to max} k_{50}(t_a) = 6.595 \quad (3.5)$$

Отримані результати означають, що при необхідності низького часу пошуку оптимального шляху алгоритм A* є кращим варіантом при невеликих розмірах оброблюваних даних з невеликою кількістю перешкод. Але, якщо необхідно використовувати великі набори вузлів, з яких лише невелика кількість є доступними, то в такому разі краще використовувати алгоритм Лі.

Наступним розглянемо критерій довжини шляху, що знаходиться алгоритмом n_p. Для знаходження оптимального результату необхідно розв'язати наступну задачу:

$$k(n_p) \to min \quad (3.6)$$

Результати довжини оптимальних шляхів кожного алгоритму наведені у таблиці 3.11.

Таблиця 3.11 – Порівняння довжини шляхів

Довжина шляхів	Сітка розміром 25x25x25			
	10%	20%	40%	60%
Алгоритм A*	26	27	33	45
Алгоритм D*	26	27	33	34
Алгоритм Лі	26	27	28	31
Довжина шляхів	Сітка розміром 50x50x50			
	10%	20%	40%	60%
Алгоритм A*	52	56	56	76
Алгоритм D*	51	56	63	74
Алгоритм Лі	52	53	55	62

Знайдемо мінімальні довжини шляхів алгоритмів на сітці, розміром 25x25x25, при мінімальній та максимальній кількості перешкод:

$$\min_{o \to min} k_{25}(n_p) = 26 \quad (3.7)$$

$$\min_{o \to max} k_{25}(n_p) = 31 \quad (3.8)$$

Знайдемо мінімальний час виконання алгоритму на сітці, розміром 50x50x50, при мінімальній та максимальній кількості перешкод:

$$\min_{o \to min} k_{50}(n_p) = 51 \quad (3.9)$$

$$\min_{o \to max} k_{50}(n_p) = 62 \quad (3.10)$$

На основі отриманих результатів можна зазначити, що при низькій кількості перешкод усі перевірені алгоритми знайдуть найбільш прийнятний шлях. Але, якщо потрібно використовувати великі кількості вузлів за високими рівнями перешкод, то найкращим рішенням буде алгоритм Лі.

Нарешті розглянемо останній критерій кількості вузлів n_a, що оброблюються алгоритмом. Для знаходження оптимального результату необхідно розв'язати наступну задачу:

$$k(n_a) \to min \quad (3.11)$$

Результати кількості оброблених вузлів кожним алгоритмом наведені у таблиці 3.12.

Таблиця 3.12 – Порівняння кількості оброблених вузлів

Кількість оброблених вузлів	Сітка розміром 25x25x25			
	10%	20%	40%	60%
Алгоритм A*	27	32	42	68
Алгоритм D*	52	58	74	85
Алгоритм Лі	14056	12513	9398	6236
Кількість оброблених вузлів	Сітка розміром 50x50x50			
	10%	20%	40%	60%
Алгоритм A*	55	67	66	230
Алгоритм D*	381	138	145	229
Алгоритм Лі	112314	99899	74791	50037

Знайдемо мінімальну кількість оброблених алгоритмами вузлів на сітці, розміром 25x25x25, при мінімальній та максимальній кількості перешкод:

$$\min_{o \to min} k_{25}(n_a) = 27 \quad (3.12)$$

$$\min_{o \to max} k_{25}(n_a) = 68 \quad (3.13)$$

Знайдемо мінімальний час виконання алгоритму на сітці, розміром 50x50x50, при мінімальній та максимальній кількості перешкод:

$$\min_{o \to min} k_{50}(n_a) = 55 \quad (3.14)$$

$$\min_{o \to max} k_{50}(n_a) = 229 \quad (3.15)$$

З отриманих результатів можна встановити, що алгоритм A* є найбільш доцільним алгоритмом, якщо необхідний якомога менший об’єм використовуваних ресурсів.

3.3 Рекомендації з вибору оптимальних алгоритмів пошуку шляху у тривимірному просторі

На основі отриманих результатів дослідження можливо сформувати деякі рекомендації з вибору алгоритму пошуку оптимального шляху у трьохвимірному просторі. Беручи до уваги обмеження та припущення, сформовані для проведення дослідження, подібні адаптації алгоритмів можливо реалізувати для повітряної доставки корисних вантажів безпілотними літальними апаратами.

Головним критерієм при проведенні доставки є час виконання цієї доставки. У разі використання безпілотного літального апарата цей критерій припадає не тільки на швидкість руху цього апарата, але і на швидкість формування необхідного шляху переміщення від складу до адреси доставки. В залежності від бажаної деталізації сітки, яка представлятиме собою віртуальний макет міста, можливо обрати алгоритм A* при недалеких перельотах в межах найближчих до складу корисних вантажів вулиць міста. Незважаючи на це, рекомендується використовувати алгоритм Лі для досягнення максимальної деталізації карти перешкод та проведення доставок у межах усього міста.

Не менш важливим критерієм при доставці є довжина шляху, пройденого від складу до адреси замовника. Авжеж, це правдиво тому, що замовник завжди бажає отримати свій товар якомога швидше. Але довжина шляху також означає, що увесь цей час безпілотний літальний апарат та, власне, сам корисний вантаж, будуть знаходитись за межами можливостей охорони працівниками служби доставки. Навіть при постійному відстеженні безпілотного літального апарату, за допомогою його зв’язку с сервером інформаційної системи, потрібен деякий час на реагування працівниками у разі виникнення проблеми. Тому для цього критерію рекомендується використовувати алгоритм Лі, тому що його можливість швидко обробляти великі об’єми даних дозволяє сформувати більш деталізовану карту місцевості, що у свою чергу допомагає розрахувати найкращий шлях руху.

Наступний критерій з часом перестає бути дуже важливим, тому що нові технології збільшують можливості обчислювальних систем, при цьому зменшуючи їх фізичний розмір. Але нове не завжди доступніше і дешевше, тому критерій кількості оброблених алгоритмом вузлів дозволить зменшити витрати на апаратне забезпечення безпілотних літальних апаратів. Якщо головною метою служби повітряної доставки корисних вантажів є збільшення прибутку та зменшення витрат, то для не менш успішного, хоча й дешевшого рішення, рекомендується використовувати алгоритм A*.

Останнім критерієм, який важко підрахувати та оптимізувати, є динамічність формування шляху. Більшість розглянутих алгоритмів підтримують лише початковий розрахунок оптимальному шляху. Це означає, якщо на шляху безпілотного літального апарата з’явиться непередбачувана перешкода, то існує можливість повної зупинки руху. Для вирішення подібних проблем існують захисні пристрої для безпілотних літаючих апаратів, але якщо головним пріоритетом є зберігання обладнання в близькому до незайманого стані, то рекомендується використовувати алгоритм D*. Хоча він і трохи повільніший ніж інші алгоритми, але має корисну властивість огинання будь-яких непередбачуваних перешкод прямо під час виконання доставки.

Після ознайомлення з рекомендаціями до використання розглянутих алгоритмів, можливо перейти до прикладу проектування компоненту, що дозволить використовувати ці алгоритми в існуючій інформаційній системі.

4 ІНТЕГРАЦІЯ КОМПОНЕНТУ ВИБОРУ АЛГОРИТМУ ПОШУКУ ОПТИМАЛЬНОГО ШЛЯХУ В ІНФОРМАЦІЙНУ СИСТЕМУ ОРГАНІЗАЦІЇ ПОВІТРЯНОЇ ДОСТАВКИ КОРИСНИХ ВАНТАЖІВ

4.1 Визначення системних вимог до компоненту системи забезпечення доставки

Компонент системи забезпечення доставки потрібен для автоматизації роботи відділу доставки. Використовуючи цю систему персонал не повинен залишати приміщення організації та повинен мати усі необхідні дані задля виконання своїх обов’язків, а саме відправка безпілотних літальних апаратів на виконання доставки, їх прийом після повернення на склад, контроль виконання доставки. Виходячи з наведених проблем можна визначити системні вимоги до компоненту системи забезпечення доставки.

Системні вимоги до компоненту системи забезпечення доставки:

– програмне забезпечення безпілотних літальних апаратів повинно мати доступ до серверу системи у будь-який час;

– менеджер повинен мати змогу перевірити місцезнаходження безпілотних літальних апаратів та статус доставки зі свого робочого місця.

4.2 Визначення функціональних вимог до компоненту системи забезпечення доставки

Задля визначення функціональних вимог до компоненту системи забезпечення доставки побудована функціональна модель системи.[24]

На рис. 4.1 зображена контекстна діаграма, яка є вершиною деревовидної структури діаграм і являє собою загальний опис системи та її взаємодії із зовнішнім середовищем.

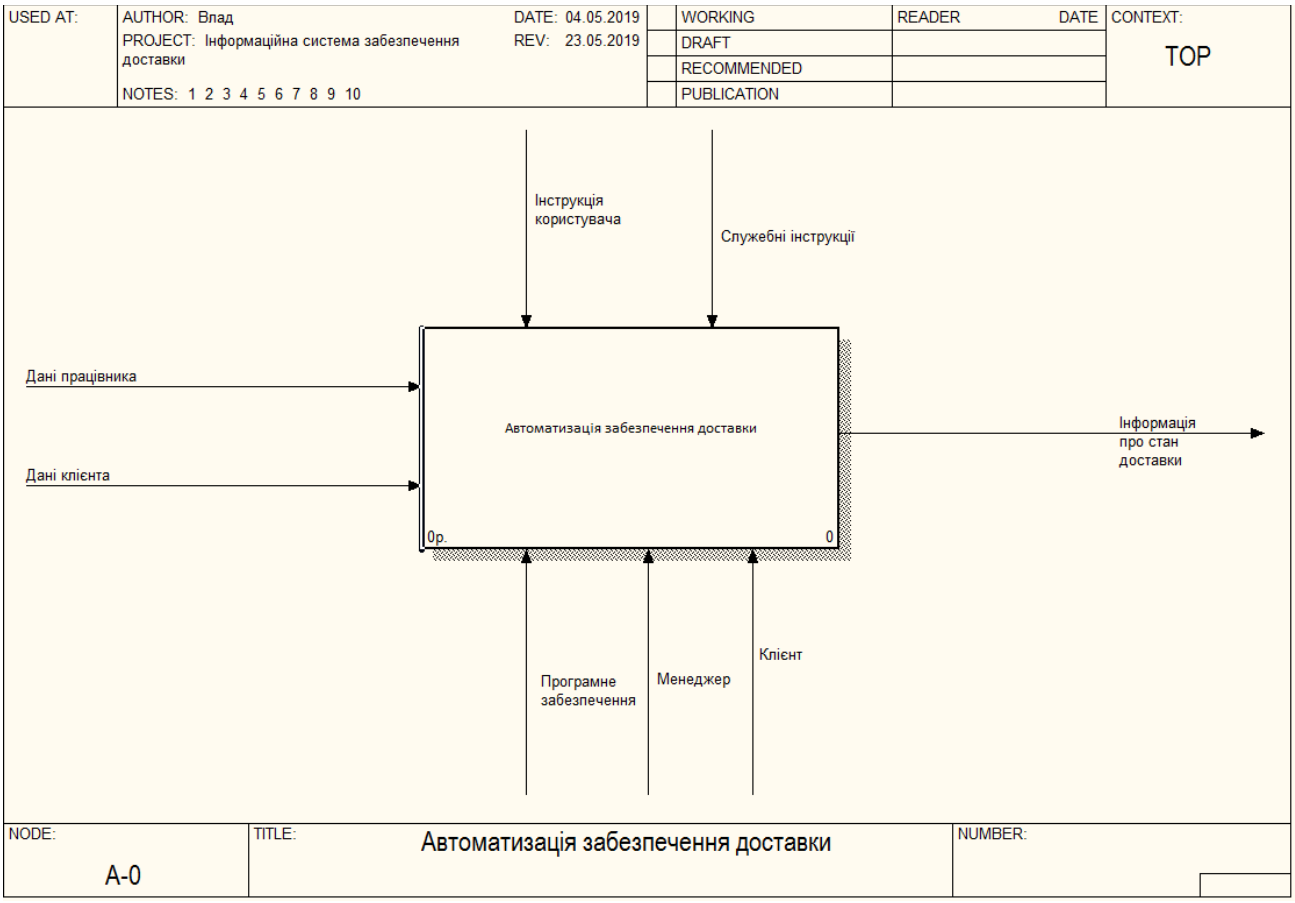

Рис. 4.1 – Контекстна діаграма

Основні інформаційні потоки:

а) вхідні потоки:

– дані клієнта;

– дані менеджера;

б) керуючі потоки:

– інструкція користувача;

– службові інструкції;

в) механізми:

– менеджер;

– програмне забезпечення;

г) вихідні потоки:

– інформація про стан доставки.

Об'єкт моделювання - інформаційна система забезпечення доставки.

Мета моделювання - описати процес використання та управління системою.

Система розглядається з точки зору менеджера системи.

Після опису системи в цілому проводиться розбиття її на великі фрагменти. Цей процес називається функціональною декомпозицією, а діаграми, які описують кожен фрагмент і взаємодію фрагментів, називаються діаграмами декомпозиції. Після декомпозиції контекстної діаграми проводиться декомпозиція кожного великого фрагмента системи на більш дрібні і так далі, до досягнення потрібного рівня точності опису. Так досягається відповідність моделі реальним процесам на будь-якому і кожному рівні моделі. Синтаксис опису системи в цілому і кожного її фрагмента однаковий у всієї моделі.

Після декомпозиції контекстної діаграми «Автоматизація забезпечення доставки» виникло три функції, зображені на рис. 4.2:

а) взаємодія з користувачем через інтерфейс;

б) взаємодія безпілотних літальних апаратів з системою;

в) обслуговування програмної системи.

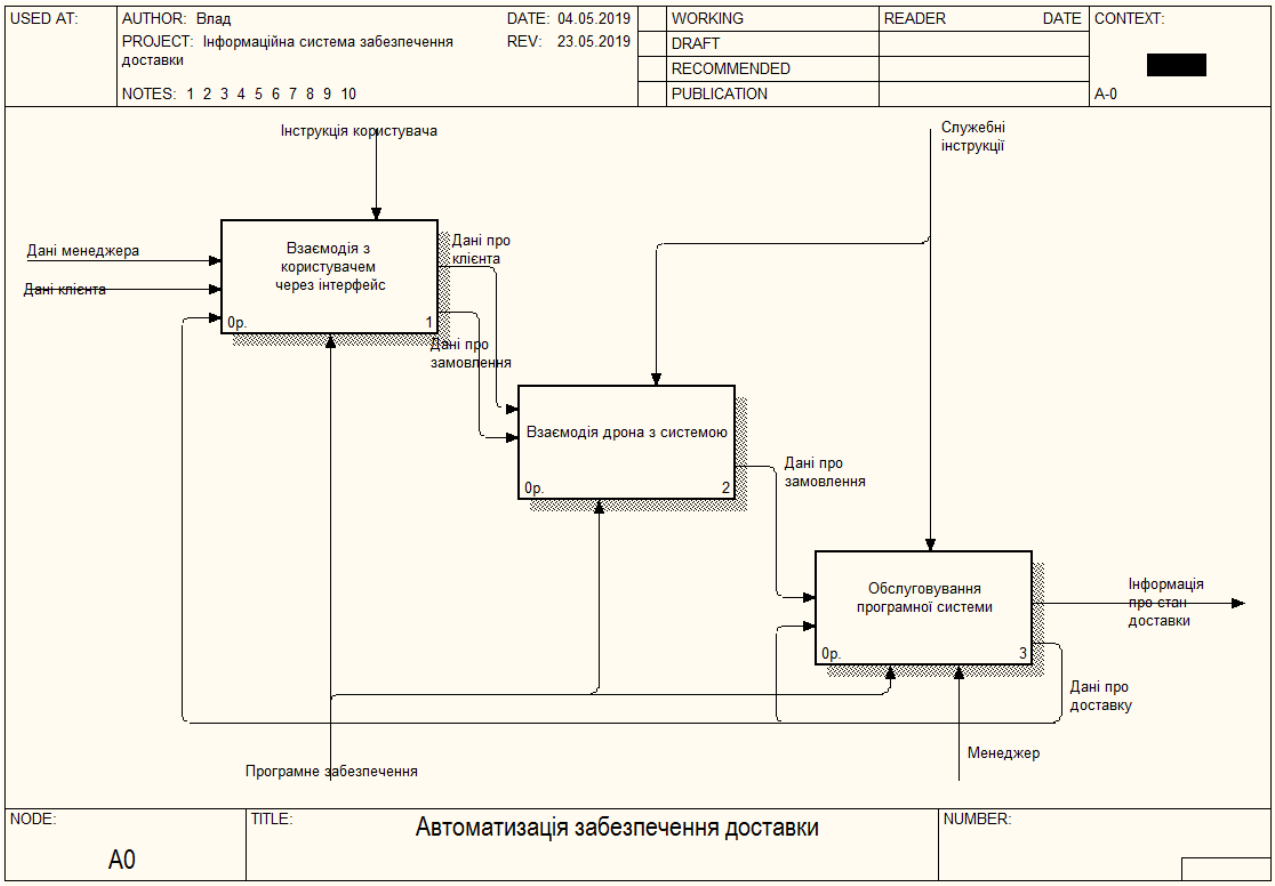

Рис. 4.2 – Діаграма декомпозиції

Взаємодія з користувачем через інтерфейс контролюється інструкцією користувача. На вхід надходять дані клієнта, дані менеджера і дані про доставку. Механізми представляють собою програмне забезпечення. На виході маємо дані про клієнта і дані про замовлення.

Взаємодія безпілотних літальних апаратів з системою контролюється службовими інструкціями. На вхід надходять дані про клієнта і дані про замовлення. Механізми - програмне забезпечення. На виході маємо дані про замовлення.

Обслуговування програмної системи контролюється службовими інструкціями. На вхід надходять дані про замовлення і дані про доставку. Механізми - програмне забезпечення і менеджер. На виході маємо інформацію про стан доставки.

На рис 4.3 зображена декомпозиція функції «взаємодія з користувачем через інтерфейс»:

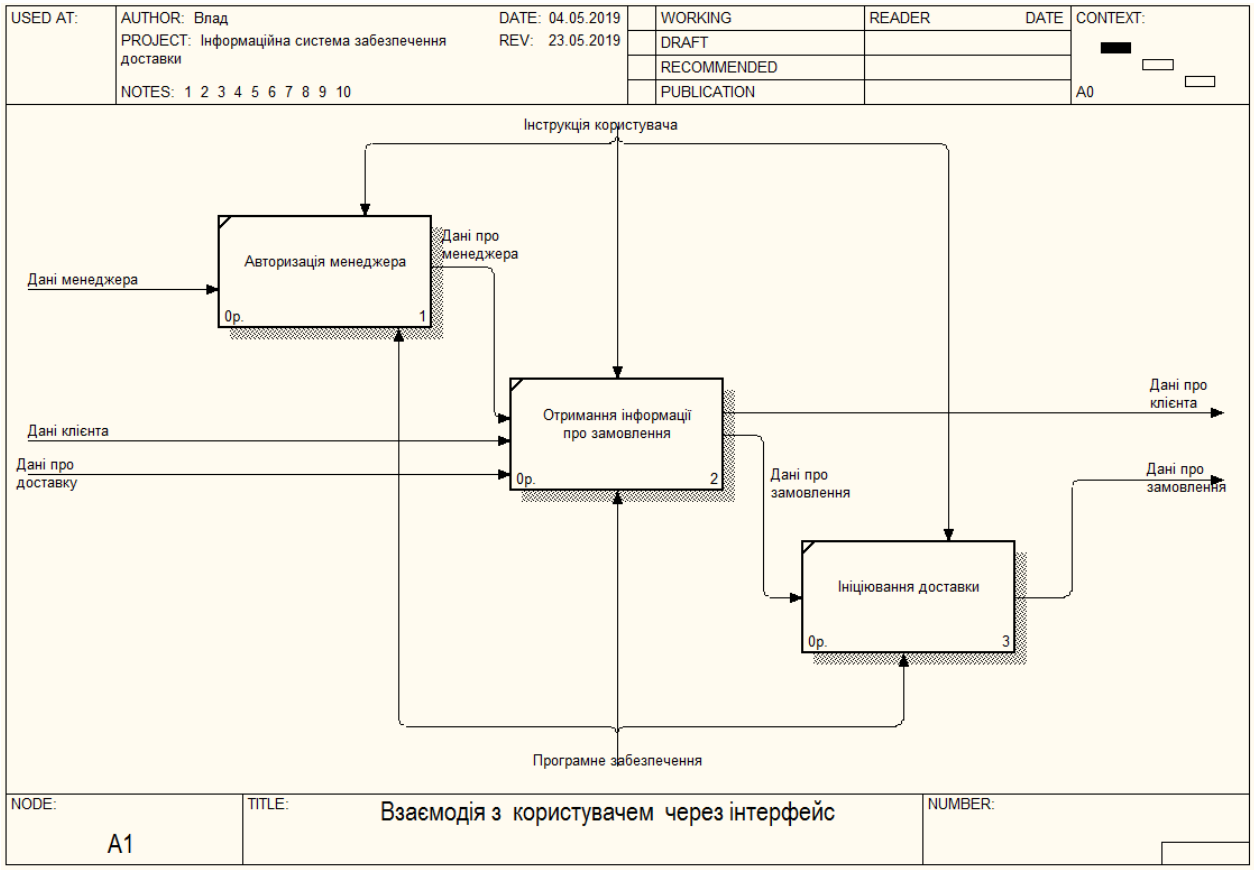

Рис. 4.3 – Діаграма A1

Авторизація менеджера контролюється інструкцією користувача. На вхід надходять дані менеджера. Механізми - програмне забезпечення. Вихід - дані про менеджера.

Отримання інформації про замовлення контролюється інструкцією користувача. На вході дані про клієнта, дані про менеджера і дані про доставку. Механізми – програмне забезпечення. На виході маємо дані про замовлення та дані про клієнта.

Ініціювання доставки контролюється інструкцією користувача. На вхід подаються дані про замовлення. Механізми – програмне забезпечення. На виході маємо дані про замовлення.

На рис 4.4 зображена декомпозиція функції «взаємодія безпілотних літальних апаратів з системою»:

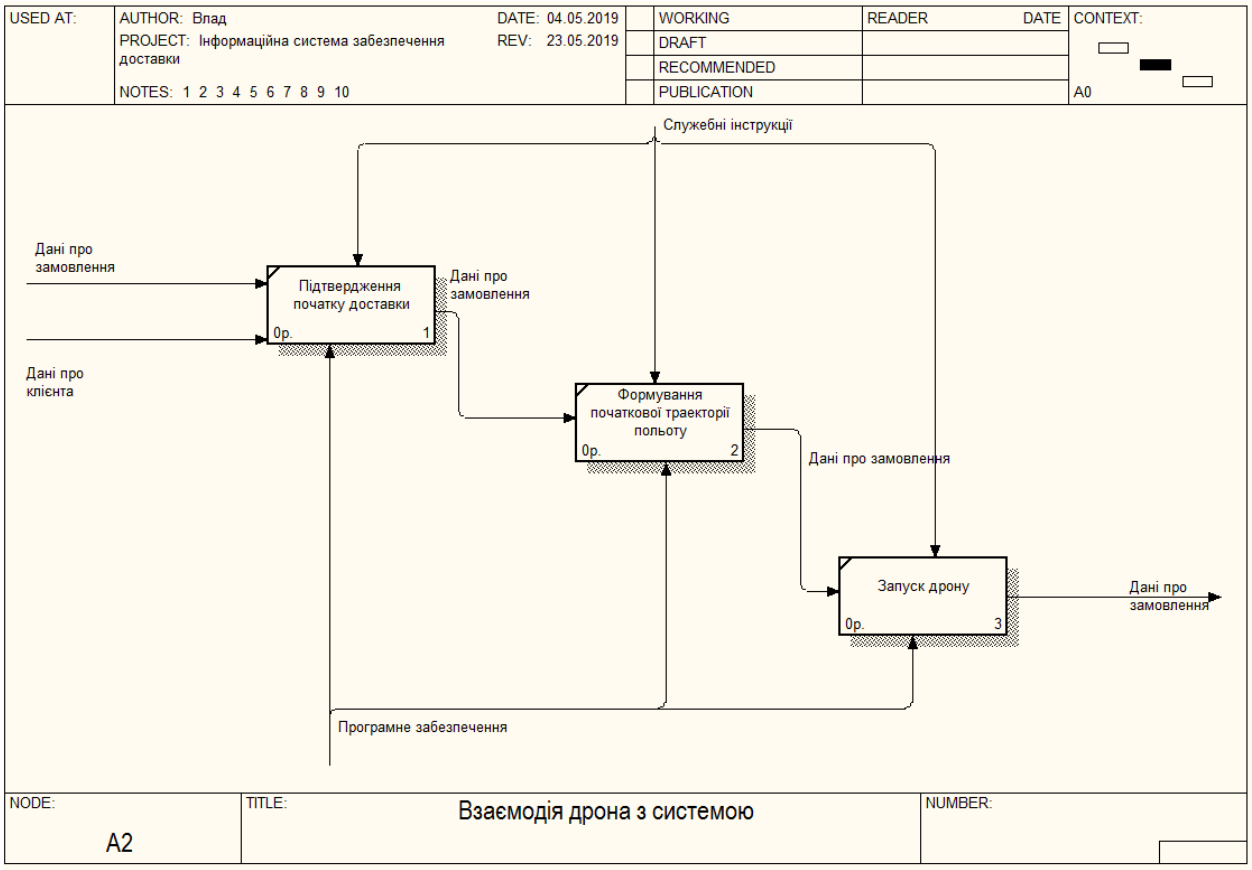

Рис. 4.4 – Діаграма A2

Підтвердження початку доставки контролюється службовими інструкціями. На вхід надходять дані про клієнта і дані про замовлення. Механізми – програмне забезпечення. На виході маємо дані про замовлення.

Формування початкової траєкторії польоту контролюється службовими інструкціями. На вхід подаються дані про замовлення. Механізм – програмне забезпечення. На виході маємо дані про замовлення.

Запуск безпілотних літальних апаратів контролюється службовими інструкціями. На вхід надходять дані про замовлення. Механізм - програмне забезпечення. На виході маємо дані про замовлення.

На рис 4.5 зображена декомпозиція функції «обслуговування програмної системи»:

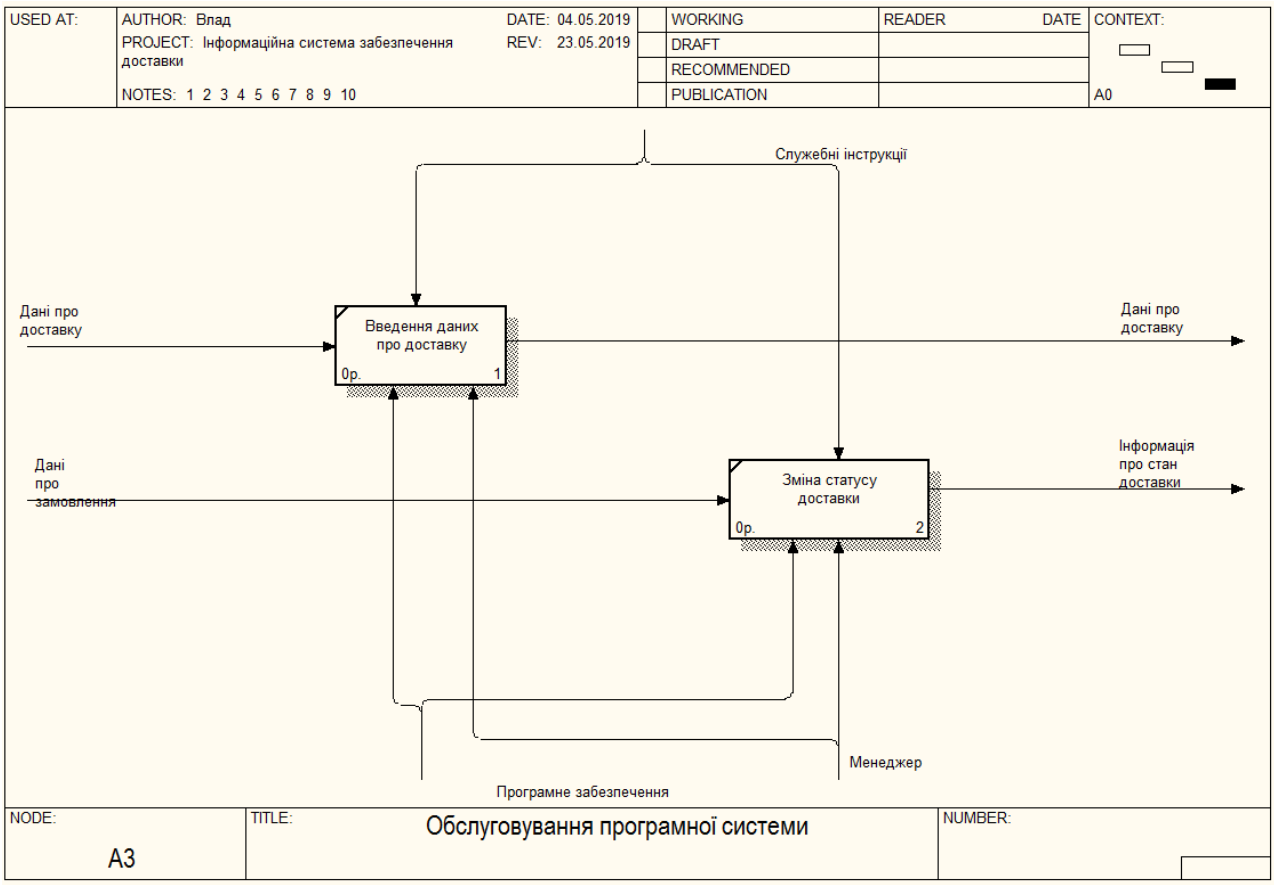

Рис. 4.5 – Діаграма A3

Введення даних про доставку контролюється службовими інструкціями. На вхід надходять дані про доставку. Механізми – програмне забезпечення і менеджер. На виході маємо введені дані про доставку.

Зміна статусу доставки контролюється службовими інструкціями. На вхід подаються дані про замовлення. Механізми – програмне забезпечення і менеджер. На виході маємо інформацію про стан доставки.

Таким чином, була створена функціональна модель інформаційної системи за методологією IDEF0, яка містить наочний опис основного функціоналу і може використовуватися для подальшої розробки системи. Модель описує процес функціонування системи з точки зору менеджера системи і визначає основні функціональні вимоги до проектованої системі.

Функціональні вимоги до інформаційної системи забезпечення доставки:

Користувачем інформаційної системи є менеджер системи;

Всього є два типи користувачів, у кожного з яких свої функції:

а) незареєстрований користувач;

б) менеджер системи.

Інформаційна система повинна забезпечувати можливість виконання функцій авторизації менеджера, отримання інформації про замовлення, ініціювання доставки та відстеження стану доставки.

Незареєстровані користувачі повинні мати такі функції: можливість авторизації в системі.

Інформаційна система повинна забезпечувати роботу основної категорії користувачів: менеджер системи.

Ідентифікація менеджера здійснюється за допомогою введення логіна і пароля, значення яких перевіряються на наявність спеціальної «ролі» в базі даних.

Функції системи:

а) авторизація користувачів:

– заповнення форми входу на відповідній сторінці;

– при коректному заповненні і успішній відправці форми входу повинен здійснюватися перехід в особистий кабінет менеджера, при неправильному введенні логіна або пароля з'являється повідомлення з підказкою і проханням ввести дані заново;

б) надання доступу до даних замовлень:

– менеджер повинен мати можливість ознайомитися з усіма доступними замовленнями;

– менеджер повинен мати можливість ініціювання доставки;

в) зміна статусу доставки:

– менеджер повинен мати можливість переглядати всі наявні доставки замовлень;

– менеджер повинен мати можливість підтвердити початок доставки та кінець доставки.

4.3 Логічне та фізичне моделювання даних

Початковою стадією проектування системи баз даних є побудова семантичної моделі предметної області, яка базується на уявленні користувачів про предметну область. Цю стадію прийнято називати концептуальним проектуванням системи, а її результат - концептуальною моделлю предметної області. Така модель створюється без орієнтації на якусь конкретну систему управління базами даних і модель даних.

Найпоширенішим засобом опису концептуальних схем предметної області є модель даних "сутність-зв'язок» (ER-модель). З її допомогою можна виділити ключові сутності і позначити зв'язки, які можуть встановлюватися між цими сутностями. ER-модель являє собою формальну конструкцію, яка сама по собі не встановлює ніяких графічних засобів її візуалізації. Як стандартна графічна нотація, за допомогою якої можна візуалізувати ER-модель, була запропонована діаграма сутність-зв'язок (ER-діаграма).

Логічне проектування - створення схеми бази даних на основі конкретної моделі даних. В даному випадку - для реляційної. Для реляційної моделі даних логічна модель - набір схем відносин, зазвичай із зазначенням первинних ключів, а також «зв'язків» між відносинами, що представляють собою зовнішні ключі.

Сутність - деякий окремий об'єкт або подія, що моделюється, має певний набір властивостей - атрибутів. Окремий елемент цієї множини називається "екземпляром сутності". Сутність може мати одну або декілька атрибутів, які однозначно ідентифікують кожен зразок сутності, і може мати будь-яку кількість зв'язків з іншими сутностями. Сутність зображується на ER-діаграмі у вигляді прямокутника, у верхній частині якого наводиться її назва, далі йде список атрибутів. Ключові атрибути відокремлені підкресленням.

Зв'язок - іменована асоціація між двома сутностями, при якій, як правило, екземпляр однієї сутності, названої батьківською сутністю, асоційований з довільною кількістю примірників іншої сутності, названої сутність-дитина, а кожен екземпляр сутності-дитини асоційований з одним екземпляром сутності-батька.[25]

Для системи забезпечення доставки створена логічна модель бази даних, зображена на рис. 4.6.

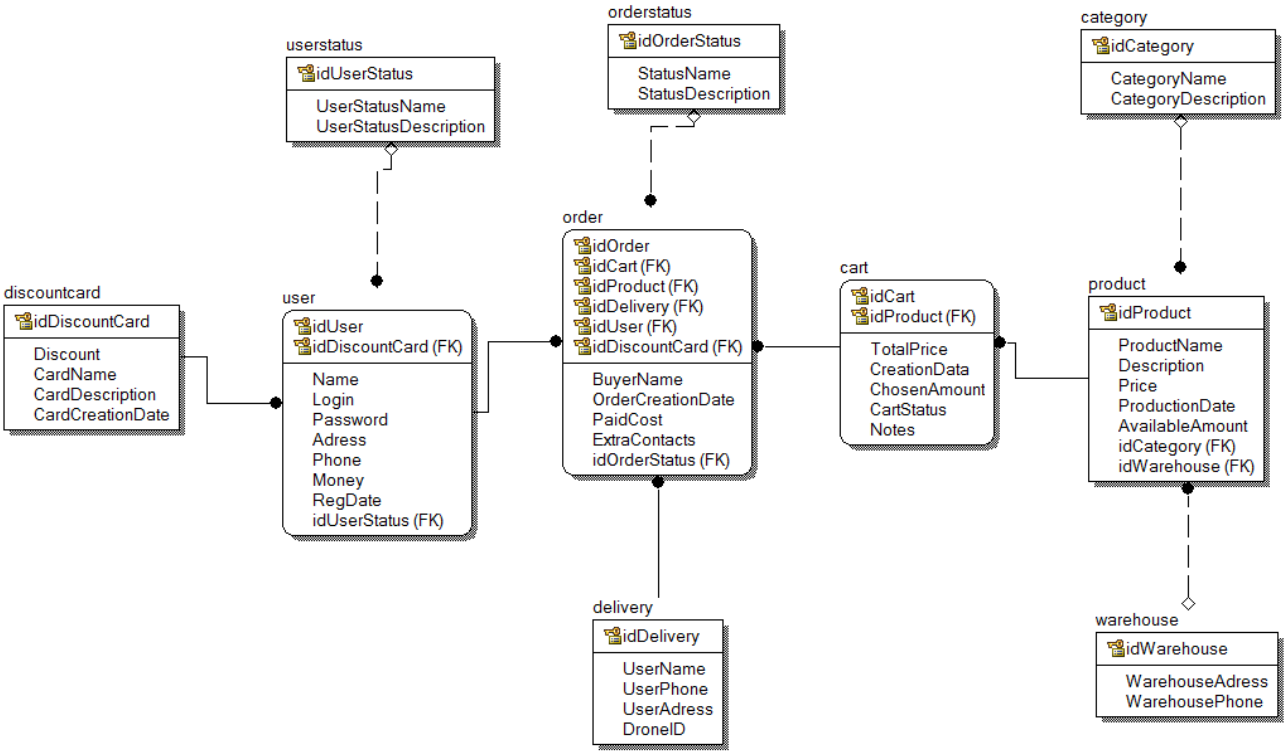

Рисунок 4.6 – Логічна модель даних

Логічна модель даних має наступні сутності:

- user – користувач;
- discountcard – карта знижок;
- userstatus – статус користувача;
- order – замовлення;
- orderstatus – статус замовлення;
- delivery – доставка;
- cart – кошик;
- product – товар;
- category – категорія;
- warehouse – склад.

Згідно логічної схеми бази даних було створено фізичну модель, яка визначає типи даних кожного атрибуту. Використовувались наступні типи даних:

- числовий;
- текстовий;
- дата.

Усі сутності, їх атрибути та типи зображені у таблиці 4.1.

Таблиця 4.1 – Перелік сутностей та їх атрибутів

№	Найменування сутності	Найменування атрибута	Тип даних	Призначення
1	user	idUser	Цілочисловий	Первинний ключ
		Name	Текстовий	
		Login	Текстовий	
		Password	Текстовий	
		Adress	Текстовий	
		Phone	Текстовий	
		Money	Числовий з рухомою крапкою	
		RegDate	Дата	
		idDiscountCard	Цілочисловий	Зовнішній ключ
		idUserStatus	Цілочисловий	Зовнішній ключ
2	userstatus	idUserStatus	Цілочисловий	Первинний ключ
		UserStatusName	Текстовий	
		UserStatusDescription	Текстовий	
3	discountcard	iDiscountCard	Цілочисловий	Первинний ключ
		Discount	Числовий з рухомою крапкою	
		CardName	Текстовий	
		CardDescription	Текстовий	
		CardCreationDate	Дата	
4	order	idOrder	Цілочисловий	Первинний ключ
		BuyerName	Цілочисловий	
		OrderCreationDate	Дата	
		PaidCost	Числовий з рухомою крапкою	
		ExtraContacts	Текстовий	
		idUser	Цілочисловий	Зовнішній ключ
		idDiscountCard	Цілочисловий	Зовнішній ключ
		idCart	Цілочисловий	Зовнішній ключ

Продовження таблиці 4.1

		idOrderStatus	Цілочисловий	Зовнішній ключ
		idDelivery	Цілочисловий	Зовнішній ключ
5	orderstatus	idOrderStatus	Цілочисловий	Первинний ключ
		StatusName	Текстовий	
		StatusDescription	Текстовий	
6	delivery	idDelivery	Цілочисловий	Первинний ключ
		UserName	Текстовий	
		UserPhone	Текстовий	
		UserAdress	Текстовий	
		DroneID	Цілочисловий	
7	cart	idCart	Цілочисловий	Первинний ключ
		TotalPrice	Числовий з рухомою крапкою	
		CreationDate	Дата	
		ChosenAmount	Цілочисловий	
		CartStatus	Текстовий	
		Notes	Текстовий	
		idProduct	Цілочисловий	Зовнішній ключ
8	product	idProduct	Цілочисловий	Первинний ключ
		ProductName	Текстовий	
		Description	Текстовий	
		Price	Числовий з рухомою крапкою	
		ProductionDate	Дата	
		AvailableAmount	Цілочисловий	
		idCategory	Цілочисловий	Зовнішній ключ
		idWarehouse	Цілочисловий	Зовнішній ключ
9	category	idCategory	Цілочисловий	Первинний ключ
		CategoryName	Текстовий	
		CategoryDescription	Текстовий	
10	warehouse	idWarehouse	Цілочисловий	Первинний ключ
		WarehouseAdress	Текстовий	
		WarehousePhone	Текстовий	

Розроблена фізична модель даних зображена на рис. 4.7

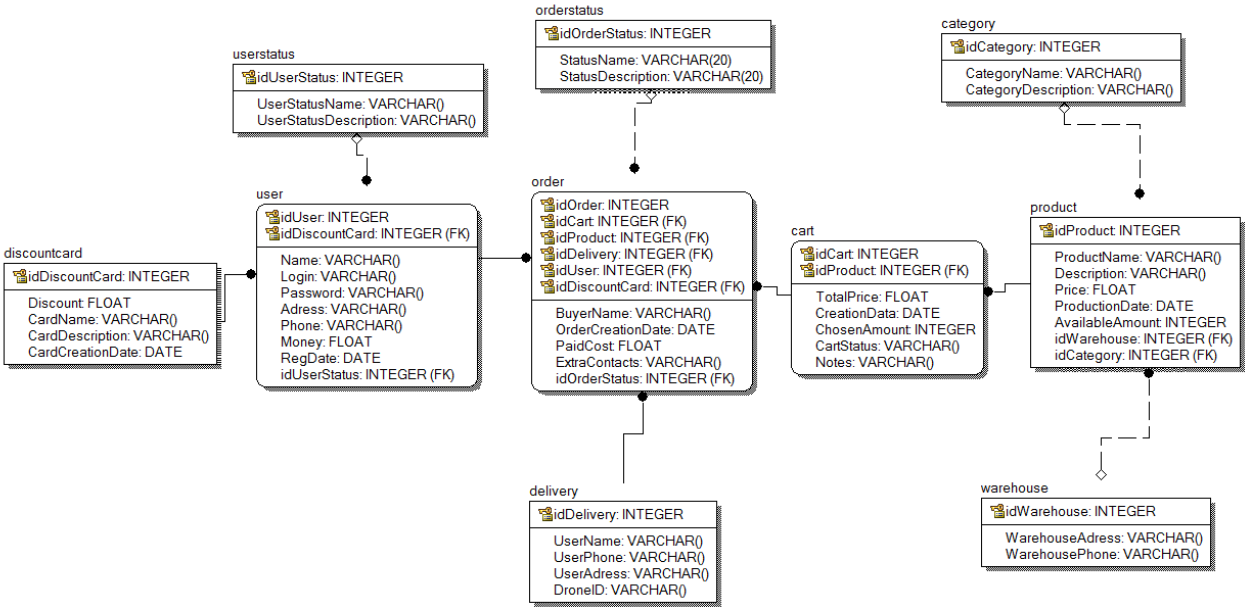

Рисунок 4.7 – Фізична модель бази даних

На основі сформованих вимог, можливо провести інтеграцію компоненту вибору алгоритму пошуку оптимального шляху для існуючої інформаційної системи організації повітряної доставки корисних вантажів (див. рис. 4.8)

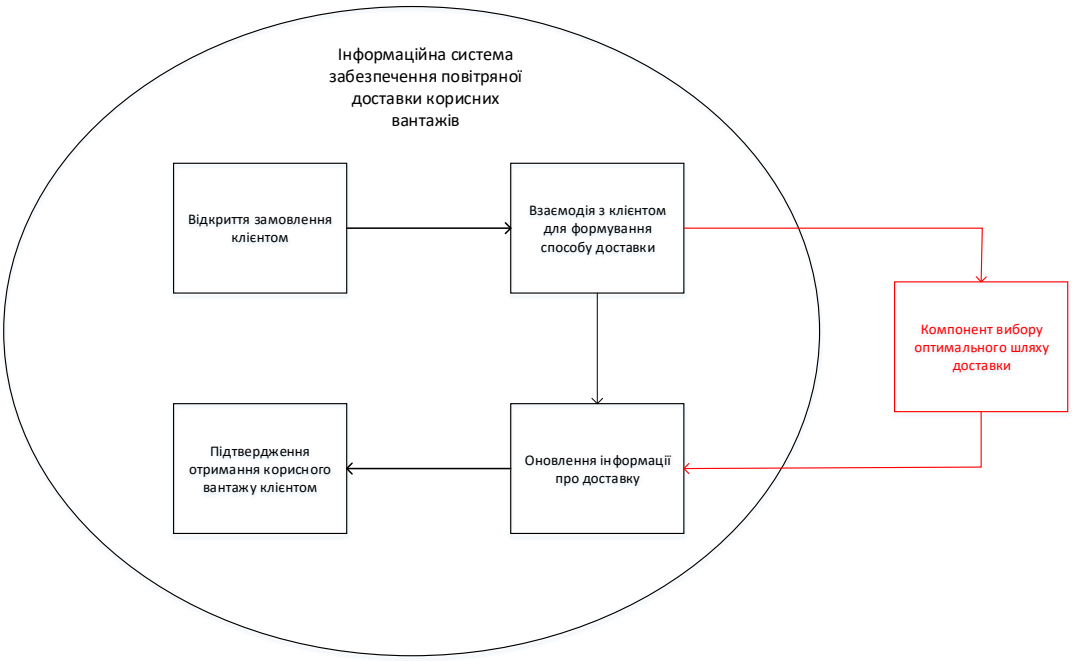

Рисунок 4.8 – Схема інтеграції компоненту в інформаційну систему

ВИСНОВКИ

Під час виконання атестаційної роботи було виконано ряд завдань, що спрямовувались на досягнення мети дослідження методів пошуку оптимальних шляхів руху в трьохвимірному просторі для інформаційних систем організації повітряної доставки корисних вантажів.

В результаті виконання роботи була вивчена предметна область дослідження та проаналізовані джерела інформації з теми дослідження. Це дозволило виявити проблемні задачі в предметній області.

Для вирішення проблеми були проаналізовані існуючи методи їх вирішення, що допомогло обрати методи розв’язання задач для проведення дослідження.

За результатами проведеного дослідження були виявленні недоліки та переваги обраних алгоритмів пошуку шляху у трьохвимірному просторі, які дозволили сформувати ряд рекомендації щодо застосування цих алгоритмів як частини існуючої інформаційної системи організації повітряної доставки.

Були сформовані вимоги, які у майбутньому дозволять розробити компонент автоматизації повітряної доставки та інтегрувати його в існуючу інформаційну систему.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. СПБ-ЭКСПЕРТ [Електронний ресурс] Режим доступу до ресурсу: https://xn----8sbbgpsnbf3agudte6o.xn--p1ai/stati/remont-gruzovyh-avtomobilej-vidy-i-ispolzovanie/

2. На часі [Електронний ресурс] - Режим доступу до ресурсу: https://nachasi.com/2019/10/24/glovo-20-city-ukraine/

3. Ліга.Бізнес [Електронний ресурс] Режим доступу до ресурсу: https://biz.liga.net/all/transport/novosti/kiev-zanyal-12-mesto-v-reytinge-gorodov-s-samymi-bolshimi-probkami

4. Novel 3D Media Technologies / Kondoz, Ahmet, Dagiuklas, Tasos, 2015 p. ISBN 978-1-4939-2026-6

5. A* pathfinding algorithm [Електронний ресурс] - Режим доступу до ресурсу: https://www.growingwiththeweb.com/2012/06/a-pathfinding-algorithm.html

6. Поиск пути в 3D-пространстве с учетом динамических объектов / Винокурова Світлана Марійський технічний університет.

7. Theta*: Any-Angle Path Planning on Grids / Kenny Daniel, Alex Nash, Sven Koenig, Ariel Felner University of Southern California, 2010p. – 47 с.

8. All shortest distances in a graph. An improvement to Dantzig's inductive algorithm / YvesTabourier, 1973 p. – 83 с.

9. Techie Delight [Електронний ресурс] Режим доступу до ресурсу: https://www.techiedelight.com/pairs-shortest-paths-floyd-warshall-algorithm/

10. Dijkstra, Dantzig, and Shortest Paths [Електронний ресурс] Режим доступу до ресурсу: https://stegua.github.io/blog/2012/09/19/dijkstra/

11. Maze Routing - Lee's Algorithm [Електронний ресурс] Режим доступу до ресурсу: https://amitrajitbose.github.io/blog/maze_routing_lee_algorithm/

12. Pathfinding Algorithms in Game Development / Abdul Rafiq, 2020 p. –12 с.

13. A comprehensive study on pathfinding techniques for robotics and video games / Zeyad Abd Algfoor, Mohd Shahrizal Sunar, Hoshang Kolivand, 2015 p. – 12 с.

14. Solution Strategy for One-to-One Pickup and Delivery Problem Using the Cyclic Transfer Approach/ R.Dupas, I. Grebennik, I. Litvinchev, T. Romanova, O. Chorna, 2020 p. – 9c.

15. A Comparative Study of the Algorithms for Path finding to Determine the Adversary Path in Physical Protection System of Nuclear Facilities / A. Марді, I. Савітрі , 2019 p. – 11 c.

16. Introduction to Algorithms (3rd ed.) / Thomas H. Cormen, 2009 p.

17. Fast algorithms for shortest paths / Alistair Moffat, University of Canterbury, 1985 p. – 178 c.

18. Pathfinding algorithms [Електронний ресурс] – Режим доступу до ресурсу: https://neo4j.com/docs/graph-algorithms/current/algorithms/pathfinding/

19. The D* Algorithm for Real-Time Planning of Optimal Traverses / Anthony Stentz – Carnegie Mellon University, 1994p. – 34 c.

20. A new weighted pathfinding algorithms to reduce the search time on grid maps / Zeyad Abd Algfoora, Mohd Shahrizal Sunara, Afnizanfaizal Abdullah, 2017 p. – 319 c.

21. Study of optimal pathfinding techniques [Електронний ресурс] – Режим доступу до ресурсу:https://www.longdom.org/open-access/study-of-optimal-path-finding-techniques-0976-4860-4-124-130.pdf

22. Методи прийняття рішень : навч. посіб. / О. Г. Наконечний, І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев ; Мін-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2016. – 132 с. : іл. – ISBN 978-966-659-212-8. – 5.45

23. Многокритериальные задачи принятия решений. / А. Лотов, И. Поспелова, – М.: МАКС Пресс, 2008. – 197 с.

24. IDEF0 and SADT: A Modeler's Guide by David A. Marca, Clement L. McGowan ISBN-13: 978-0977604432

25. Дубейковский В. И. Эффективное моделирование с CA ERwin Process Modeler (BPwin; AllFusion Process Modeler) / В. И. Дубейковский. – 2-е изд., испр. и доп. – М. : ДИАЛОГ-МИФИ, 2009. – 384 с. : ил. – ISBN 5-86404-216-1. – 88,90