МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____ Ігор ЖУКОВ

«_____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Веб-базована система прокладання маршрутів для БПЛА»

Виконавець: Костянтин КОНОВАЛЕНКО

Керівник: Сергій ГІЛЬГУРТ

Нормоконтролер: Василь МАЛЯРЧУК

Засвідчую, що у кваліфікаційній роботі немає запозичень праць інших авторів без відповідних посилань
Студент _____ Коноваленко К.Ю

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютерних систем та мереж

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ігор ЖУКОВ
«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Коноваленка Костянтина Юрійовича

1. Тема кваліфікаційної роботи: Веб-базована система прокладання маршрутів для БПЛА

затверджена наказом ректора від «29» серпня 2023 р. № 1521/ст.

2.Термін виконання роботи: з 2 жовтня 2023 р. по 31 грудня 2023 р.

3. Вихідні дані до роботи: технічна документація, мова програмування *JavaScript*

4. Зміст пояснювальної записки: вступ; аналіз та порівняння алгоритмів пошуку шляху; аналіз побудови оптимального машруту для БПЛА; розробка веб-додатку для пошуку оптимального шляху для БПЛА; тестування веб-додатку; висновки.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

– блок-схема алгоритму *BFS;*

– ілюстративна схема обходу Дейкстри;

– ілюстративна схема обходу A*;

– ілюстрація роботи Оптимізації мурашиних колоній;

– діаграма компонентів та взаємодій;

– вигляд клієнтської частини веб-додатку.

6. Календарний план

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проаналізувати існуючі системи та сформувати основні вимоги до веб-додатку	02.10.2023 – 10.10.2023	
2	Визначити структуру та розробити веб-додаток для прокладання оптимального маршруту для БПЛА	11.10.2023 – 01.11.2023	
3	Розробка розділу 1: Аналіз існуючих систем для оптимізації маршрутів	02.11.2023 – 18.11.2023	
4	Розробка розділу 2: Побудова схеми оптимізації маршрутів	19.11.2023 – 29.11.2023	
5	Розробка розділу 3: Проектування програмного модуля	30.11.2023 – 13.12.2023	
6	Оформлення пояснювальної записки	13.12.2023 – 17.12.2023	
7	Розробка презентації для захисту роботи та підготовка до захисту	18.12.2023 – 31.12.2023	

7. Дата видачі завдання: « 02 » жовтня 2023 р.

Керівник дипломної роботи _____ Сергій ГІЛЬГУРТ

Завдання прийняв до виконання _____ Костянтин КОНОВАЛЕНКО

РЕФЕРАТ

Пояснювальна записка до дипломного дослідження «Веб-базована система прокладання маршрутів для БПЛА», 86 сторінок, 12 рисунків, 4 таблиці, 25 використаних джерел

NODE.JS, UAV, MAPS, JAVASCRIPT, REACT, API, PATHFINDING, REST.

Мета дипломного дослідження – створення інтелектуальної веб-орієнтованої системи для автоматизованого прокладання оптимальних маршрутів безпілотних літальних апаратів з урахуванням їх характеристик, навколишніх умов та систем обмежень.

Об’єкт дипломного дослідження – процес прокладання оптимальних маршрутів для безпілотних літальних апаратів.

Предмет дипломного дослідження – моделі, алгоритми та програмні засоби для автоматизованого пошуку оптимальних траєкторій руху БПЛА з урахуванням їх технічних характеристик та умов функціонування.

Методи дослідження – методи функціонального тестування, тестування продуктивності, тестування стабільності та надійності, а також тестування в реальних умовах використання.

Здійснено огляд алгоритмів пошуку оптимального шляху; проаналізовано їх переваги та недоліки; досліджено особливості прокладання маршрутів для БПЛА; враховано дані особливості при розробці веб-додатку; розроблено клієнтську та серверну частину веб-додатку; протестовано розроблений веб-додаток різними методами тестування.

Матеріали дипломної роботи можуть бути використані для розрахунку оптимальних маршрутів БПЛА різного типу, а також як базу для розробки більш складних систем, які будуть враховувати більше польотних параметрів.

Подальший розвиток дослідження може бути пов'язаний з розширенням функціоналу веб-додатку, врахування більшої кількості факторів, що впливають на побудову маршруту для БПЛА та застосування новітніх технологій, зокрема, штучного інтелекту та високопродуктивних обчислювальних засобів.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

БПЛА (безпілотний літальний апарат) – повітряне судно, здатне здійснювати політ без пілота на борту автономно або під керуванням оператора

ПЗ (програмне забезпечення) – програми, процедури, правила і пов'язана з ними документація систем обробки інформації

GPS (*Global Positioning System*) – супутникова навігаційна система

API (*Application Programming Interface*) – набір визначень і протоколів, що використовуються для створення та інтеграції програмного забезпечення

UI (*User Interface*) - інтерфейс, з яким взаємодіє користувач

UX (*User eXperience*) - загальне враження від використання продукту

REST (*Representational State Transfer*) – архітектурний стиль взаємодії компонентів розподіленого застосунку

JSON (*JavaScript Object Notation*) – текстовий формат обміну даними

BFS (*Breadth-first search*) – алгоритм пошуку в ширину на графі

ACO (*Ant colony optimization*) – евристичний алгоритм оптимізації на основі поведінки мурашиних колоній

PSO (*Particle swarm optimization*) – еволюційний алгоритм оптимізації на основі концепції рою частинок

CI/CD (*Continuous integration / Continuous delivery*) – безперервна інтеграція та доставка коду

ВСТУП

Актуальність теми дослідження обумовлена стрімким зростанням сфер застосування безпілотних технологій. За останні роки спостерігається лавиноподібне збільшення використання цивільних безпілотних літальних апаратів (БПЛА) у комерційних цілях та для виконання спеціальних завдань. Зокрема безпілотники вже зараз активно задіяні в аерофотозйомці, картографуванні, сільськогосподарському та екологічному моніторингу, інспекції інфраструктури, пошуково-рятувальних операціях, доставці невеликих вантажів тощо. Попри швидкий прогрес галузі, проблема ефективного планування довгих автономних маршрутів для безпілотників на відстані сотень кілометрів досі залишається вкрай складною і вимагає подальших наукових досліджень та технологічних розробок.

Ретельні розрахунки оптимальних траєкторій з урахуванням обмежень щодо запасу пального, корисного навантаження, метеоумов тощо, які необхідні для транскордонних багатогодинних польотів, вимагають значних витрат трудових та фінансових ресурсів при ручному плануванні. Допущені некоректні розрахунки та помилки пілотів-операторів можуть призвести до неоптимальності маршруту та неможливості його виконання, або ж підвищити ризики втрати безпілотного апарату через виснаження заряду батарей або аварійної посадки в нештатній зоні.

Натомість процес автоматизації обчислення безпечних оптимальних шляхів БПЛА дозволить кардинально підвищити ефективність планування, розширить практичну сферу застосування та покращить рівень безпеки функціонування безпілотних повітряних суден, що вкрай важливо на тлі бурхливого розвитку цілої галузі безпілотної авіаційної техніки.

Додатковим чинником, що зумовлює необхідність створення таких систем є недосконаліть більшості доступних на сьогодні аналогів, що мають низку суттєвих обмежень. Зокрема, існуючі системи враховують лише обмежене коло факторів впливу при плануванні маршруту (технічні характеристики БПЛА, перешкоди тощо), не адаптовані під задачі тривалих транскордонних польотів, мають закритий

вихідний код та реалізовані на технологічно застарілих платформах без належного потенціалу масштабування. Все це значно звужує їх сфери застосування та знижує ефективність порівняно з потенціалом сучасних хмарних технологій та методів штучного інтелекту. Таким чином, створення нового покоління гнучких та адаптивних систем планування маршрутів на основі останніх досягнень є актуальним напрямком досліджень.

За статистичними даними компанії *DroneAnalyst*, хоча на долю безпілотників припадає менше 1% загального повітряного руху, кількість польотів цивільних БПЛА у світі щороку зростає на 200%. Очікується, що до 2026 року щорічний обсяг польотів безпілотників становитиме 35 мільйонів годин, а ринкова вартість послуг досягне 89 мільярдів доларів.

Такі бурхливі темпи зростання вимагають впровадження систем автоматизації та оптимізації процесів планування та управління повітряним рухом БПЛА для забезпечення достатнього рівня пропускної здатності повітряного простору, ефективності та безпеки польотів. У цьому контексті задача розробки автоматизованої системи для обчислення оптимальних маршрутів з урахуванням різноманітних факторів є особливо актуальною.

Метою даної роботи є розробка інтелектуальної веб-орієнтованої системи для автоматизованого розрахунку оптимальних за заданими критеріями маршрутів безпілотних літальних апаратів з комплексним урахуванням їх характеристик, навколишніх умов польоту та систем обмежень, яка б перевершувала існуючі аналоги за функціональністю та практичною ефективністю.

Для досягнення поставленої мети, було сформульовано такі завдання дослідження:

– аналіз існуючих алгоритмів та підходів до оптимізації маршрутів БПЛА. Оцінка їх переваг, недоліків та ефективності за різноманітними критеріями. Вибір найбільш підходящого методу для подальшої реалізації;

– дослідження особливостей різних типів БПЛА, що впливають на планування траєкторій польоту. Аналіз конструктивних та функціональних параметрів та обмежень;

– аналіз зовнішніх факторів: географічних, метеорологічних та регуляторних обмежень, які мають враховуватись при прокладанні маршрутів;

– формалізація задачі планування шляху БПЛА у вигляді математичної моделі з цільовою функцією та системою обмежень;

– адаптація обраного базового алгоритму пошуку шляху для прийняття додаткових параметрів з метою врахування технічних та експлуатаційних особливостей застосування БПЛА;

– розробка та програмна реалізація веб-орієнтованого додатку для автоматизованого планування маршрутів БПЛА на основі запропонованих моделей та алгоритмів. Забезпечення функціональності обчислення оптимальних за різними критеріями траєкторій з урахуванням широкого спектру факторів впливу;

– оцінка ефективності та практичної продуктивності розробленого рішення за допомогою комплексу функціонального та навантажувального тестування відповідно до сформульованих вимог. Порівняння запропонованого додатку з існуючими аналогами.

Об'єкт дослідження: процес прокладання оптимальних маршрутів для безпілотних літальних апаратів.

Предмет дослідження: моделі, алгоритми та програмні засоби для автоматизованого пошуку оптимальних траєкторій руху БПЛА з урахуванням їх технічних характеристик та експлуатаційних умов.

Методами дослідження виступають: аналіз літератури з тематики, порівняння та оцінка ефективності існуючих алгоритмів оптимізації, математичне моделювання задачі маршрутизації, розробка адаптованих алгоритмів на основі класичних методів, програмна реалізація та тестування запропонованих підходів, імітаційне моделювання роботи системи, оцінка якості розробленого рішення шляхом обчислювальних експериментів.

Наукова новизна роботи полягає у створенні комплексного підходу до уніфікованого аналізу та формалізації широкого спектру факторів при виборі оптимального маршруту безпілотного літального апарату на основі адаптованого евристичного алгоритму пошуку шляху з врахуванням специфіки предметної області.

Практичне значення отриманих результатів визначається можливістю впровадження розробленого веб-додатку в якості підсистеми підтримки прийняття рішень стосовно планування маршрутів у складі реальних комплексів оперативного управління безпілотною авіаційною технікою. Крім цього, запропоновані підходи можуть слугувати теоретичною основою для створення подібних систем в майбутньому, а практичні результати знайдуть застосування у навчальному процесі з відповідних напрямів підготовки фахівців у галузі безпілотних технологій.

Отже, актуальність обраної теми дослідження обумовлена стрімкими темпами розвитку галузі безпілотної авіаційної техніки та зростанням попиту на автоматизовані системи планування маршрутів для комерційного та спеціального застосування безпілотних літальних апаратів. Запропонована розробка має на меті створення нового покоління інтелектуальних веб-орієнтованих додатків для оптимізації процесів маршрутизації з урахуванням широкого кола факторів впливу.

Практичне значення отриманих результатів полягає у можливості впровадження системи в реальних умовах функціонування операторів безпілотної авіаційної техніки для підвищення ефективності планування складних транскордонних місій. Крім цього, розроблені теоретичні підходи та програмні рішення можуть слугувати основою для подальших досліджень у цій галузі.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ МЕТОДІВ РОЗРАХУНКУ МАРШРУТІВ ДЛЯ БПЛА

1.1. Аналіз сучасних методів та стратегій для оптимізації маршрутів

Оптимальний алгоритм планування маршруту має вирішальне значення для ефективного розгортання безпілотних літальних апаратів (БПЛА). У цій главі представлено всебічний огляд та аналіз сучасних методів пошуку маршрутів, їх оцінка за відповідними показниками ефективності для вибору найбільш підходящого методу для застосування в задачах маршрутизації БПЛА.

1.1.1. Пошук в ширину

Пошук в ширину (*Breadth-First Search BFS*) - це фундаментальний алгоритм пошуку шляху, який використовує поперечний обхід вершин графа (рис 1.1), щоб гарантувати знаходження найкоротшого шляху від початкової до кінцевої вершини [3]. Він використовує структуру даних у вигляді черги, що працює за принципом «перший прийшов - перший вийшов», рівномірно розширюючи границю пошуку по всіх гілках перед тим, як заглибитися вглиб.

Алгоритм на рис 1.2 ініціалізує чергу з початкового вузла. Потім він ітеративно ставить в чергу наступний вузол, перевіряє, чи він є метою, ставить в чергу всіх його сусідів, які ще не були відвідані, і позначає його як відвіданий. Це триває до тих пір, поки не буде знайдено цільову вершину або поки черга не спорожніє.

Ключовою перевагою *BFS* є гарантія оптимальності найкоротшого шляху на основі монотонно зростаючої функції вартості, що дорівнює довжині шляху. Крім того, він є повним і завжди знайде розв'язок, якщо він існує. Однак ці переваги досягаються ціною ефективності. *BFS* масштабується експоненціально з шириною графа і є нерозв'язним для дуже великих просторів пошуку. Проблеми пошуку шляху в реальних областях, таких як маршрутизація БПЛА, вимагають більш досконалих евристичних алгоритмів.

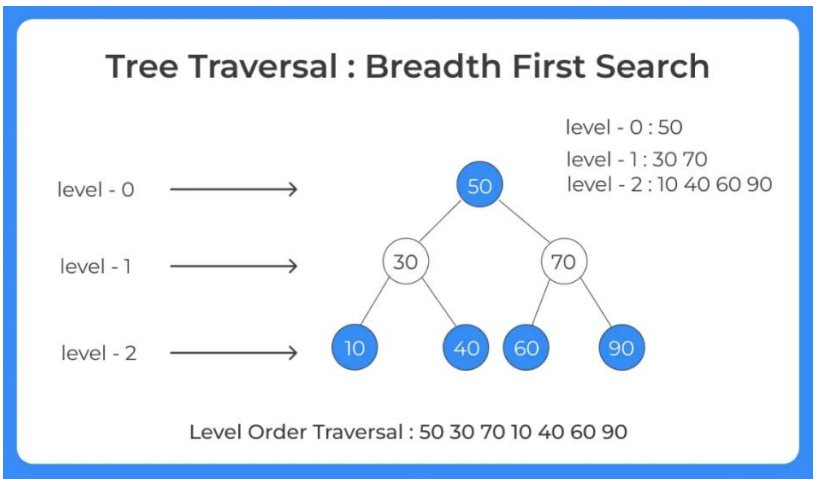

Рис. 1.1. Ілюстративна діаграма обходу *BFS*

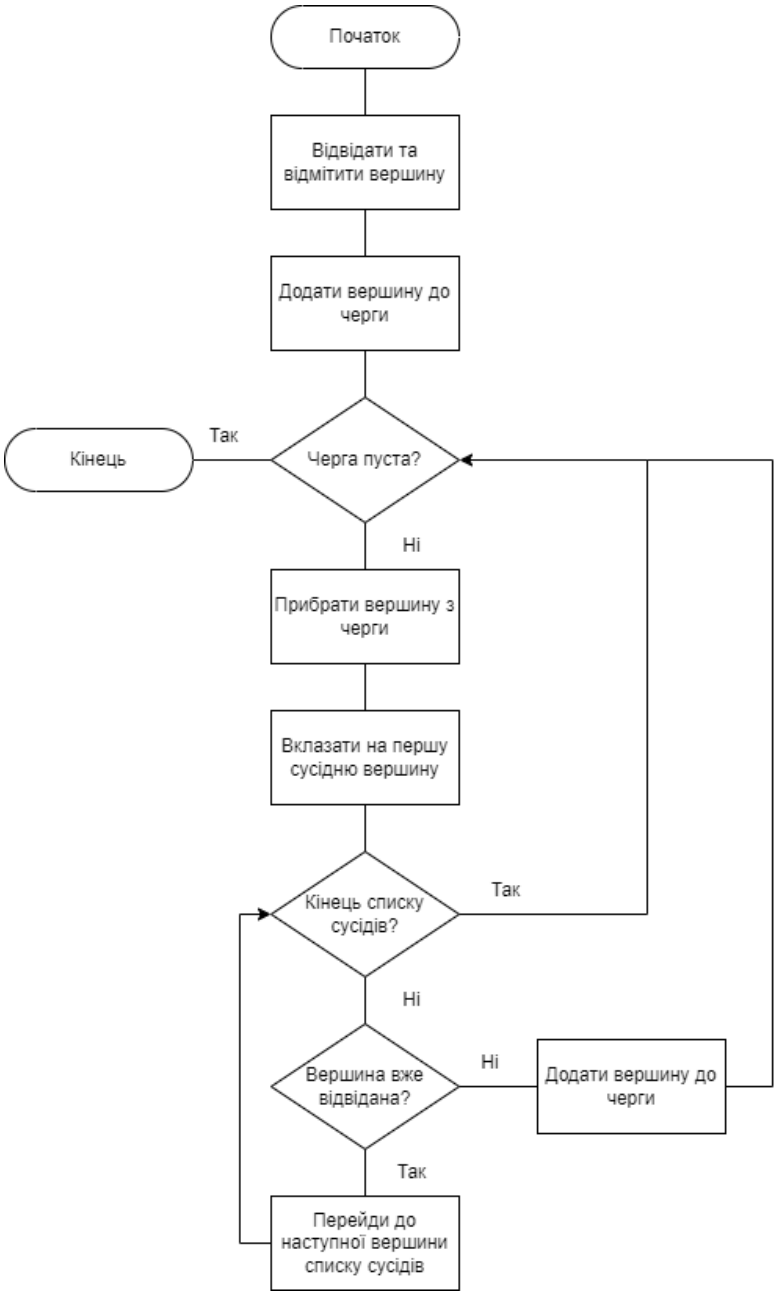

Рис. 1.2. Блок-схема алгоритму *BFS*

1.1.2. Алгоритм Дейкстри

Алгоритм Дейкстри є удосконаленням методу пошуку в ширину, що дозволяє підвищити ефективність пошуку шляху за рахунок врахування вартості переходів між вершинами графа, а не лише глибини шляху [4].

Функціонування алгоритму Дейкстри базується на тому, що до кожної вершини графа прив'язується числовий показник – вартість шляху до неї від початкової точки. Ця вартість оновлюється у процесі пошуку при знаходженні альтернативних шляхів до вершини з меншою сумарною вартістю.

Початкова вартість шляху до початкової вершини встановлюється рівною 0, а для всіх інших приймається нескінченно великою. Далі алгоритм ітеративно розглядає сусідні вершини, оновлюючи мінімальну вартість шляху до них через уже опрацьовані вершини.

Ключовим в оптимізації пошуку є використання структури даних у вигляді черги пріоритетів вершин, відсортованої за зростанням поточної мінімальної вартості до них. Завдяки цьому, замість рівномірного розгортання всіх гілок пошуку, алгоритм Дейкстри зосереджує зусилля на найбільш перспективних напрямках.

Алгоритм Дейкстри гарантує знаходження найкоротшого за сумарною вартістю шляху завдяки тому, що мінімальна на поточний момент вартість до вершини завжди відповідає довжині найкоротшого шляху до неї. Це досягається завдяки механізму оновлення вартостей та використанню покажчика на батьківську вершину, з якої прийшов «дешевший» шлях.

Основні етапи алгоритму Дейкстри можна узагальнити наступним чином:

– ініціалізація вартості шляху для початкової вершини рівною 0, а для решти вершин – нескінченністю;

– вибір невідвіданої вершини з черги з мінімальною поточною вартістю і перенесення її до множини відвіданих. Якщо досягнуто кінцеву вершину - алгоритм завершує роботу;

– для кожної сусідньої вершини: обчислення вартості переходу; перевірка, чи нова вартість менша ніж попередня, і при необхідності її оновлення разом із записом попередника;

– перехід до пункту 2.

Схематично основні кроки алгоритму представлено на рис. 1.3.

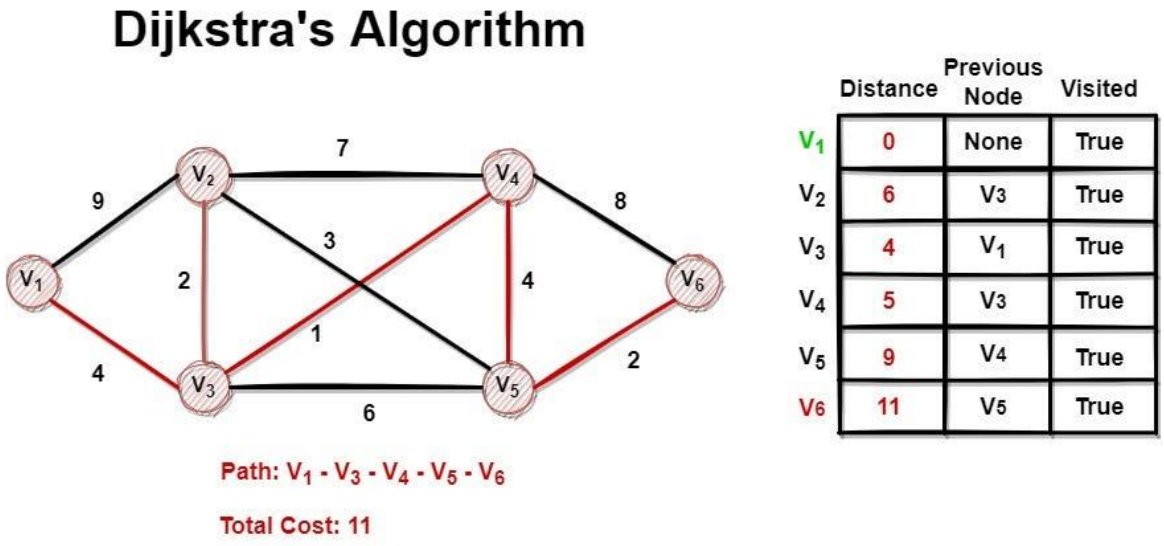

Рис. 1.3. Ілюстративна схема обходу Дейкстри

Однією з головних переваг алгоритму Дейкстри є можливість застосування довільної функції вартості переходів між ребрами графа, що дозволяє працювати не тільки з відстанями, але й з іншими метриками. Наприклад, це можуть бути показники витрат пального, часу, ризику та інші важливі для оптимального планування маршруту БПЛА характеристики.

Ще однією важливою властивістю алгоритму Дейкстри є можливість його ефективної програмної реалізації з використанням таких структур даних, як хеш-таблиці, двійкові купи тощо. Це дозволяє скоротити час пошуку до $O(|E|+|V|log|V|)$ для графа з $|V|$ вершинами та $|E|$ ребрами замість експоненціальної складності перебору варіантів.

Водночас, незважаючи на певне прискорення порівняно з вичерпним пошуком в ширину, алгоритм Дейкстри також страждає від комбінаторного вибуху при масштабуванні. Крім цього, через відсутність евристичної інформації щодо напрямку до цілі він все одно витрачає значні обчислювальні ресурси на розгортання і аналіз неоптимальних гілок пошуку.

Тому, незважаючи на певні переваги алгоритму Дейкстри в плані більш ефективного дослідження графа на основі вартості ребер, цей метод не є оптимальним рішенням для практичних задач маршрутизації БПЛА у великих за розміром

просторах пошуку. Для подальшого підвищення ефективності пошуку шляхів потрібне використання додаткової евристичної інформації, як наприклад в алгоритмі A*.

1.1.3. Алгоритм A*

Алгоритм A* (читається як «А зірка») є вдосконаленням методу Дейкстри, що використовує евристичну інформацію для більш ефективного спрямування пошуку оптимального шляху у найбільш перспективних напрямках [5].

Це досягається за рахунок об'єднання вже відомої вартості маршруту до поточної вершини з евристичною оцінкою вартості шляху від цієї вершини до кінцевої цілі пошуку. Таке поєднання реальних та прогнозних витрат дозволяє алгоритму A* приймати більш обґрунтовані рішення щодо пріоритезації напрямків пошуку на основі їх потенційної перспективності.

Більш формально, алгоритм A* використовує таку формулу для оцінки пріоритету вузлів під час пошуку: $f(n) = g(n) + h(n)$, де:

$g(n)$ - відома вартість шляху від початкової вершини до вершини n

$h(n)$ - евристична оцінка вартості шляху від вершини n до кінцевої вершини (цілі)

Тобто алгоритм A* аналізує як фактичні понесені витрати для досягнення даної вершини, так і прогнозовані майбутні витрати для завершення маршруту.

При цьому евристична функція $h(n)$ має задовольняти властивості допустимості, тобто ніколи не переоцінювати реальну найменшу вартість шляху до кінцевої вершини. Це гарантує збереження оптимальності знайденого алгоритмом A* шляху.

Найпростішою евристикою є евклідова відстань між двома вершинами. Більш досконалі евристики також можуть враховувати рельєф місцевості, перешкоди, напрямок руху, тощо, для отримання точнішої оцінки очікуваної вартості.

Основна ідея полягає в тому, щоб за допомогою евристичної функції спрямувати пошук у найбільш перспективні регіони простору станів, уникаючи розгалуження та поглиблення в явно неоптимальних напрямках. Це дозволяє алгоритму A* досягти ефекту «інформованого пошуку», коли наявні знання про

предметну область використовуються для суттєвої оптимізації процесу пошуку розв'язку задачі.

Основні етапи роботи алгоритму A* наступні:

– ініціалізація початкової та кінцевої вершин, структур даних для збереження відкритих та закритих вузлів;

– обчислення початкового значення f для стартової вершини та додавання її до множини відкритих вузлів;

– вибір відкритого вузла з мінімальним значенням f та перенесення його до множини закритих;

– перевірка, чи вибраний вузол є цільовим. Якщо так - алгоритм завершено;

– для кожного сусіднього вузла: обчислення вартості переходу g, обчислення евристичної вартості h та загальної оцінки f;

– якщо значення f для сусіднього вузла краще ніж попереднє (або він ще не розглядався) - занесення його до множини відкритих вузлів;

– перехід до кроку 3.

Однією з основних переваг алгоритм A* є можливість гнучкого налаштування шляхом використання різних евристичних функцій, адаптованих до конкретної предметної області чи проблеми оптимізації.

Наприклад, для задачі оптимізації маршруту БПЛА можуть використовуватися спеціалізовані евристики на основі даних про рельєф місцевості, погодні умови, радіус дії безпілотника тощо. Крім цього, алгоритм A* досить просто піддається модифікаціям та удосконаленням.

Ще однією важливою властивістю алгоритму A* є його висока обчислювальна ефективність. За рахунок селективного розширення лише потенційно перспективних вузлів алгоритм демонструє швидкість пошуку, близьку до лінійної, на відміну від експоненціальної у класичних алгоритмів сліпого пошуку.

Отже, завдяки використанню евристичної інформації алгоритм A* поєднує в собі оптимальність пошуку найкоротшого шляху з високою швидкодією та масштабованістю. Це робить його найбільш придатним інструментом для вирішення

практичних задач оптимізації маршрутів, зокрема планування траєкторій для безпілотних літальних апаратів.

На рис. 1.4 проілюстровано схему роботи алгоритму A* та принцип використання ним евристичної інформації для спрямування пошуку оптимального шляху.

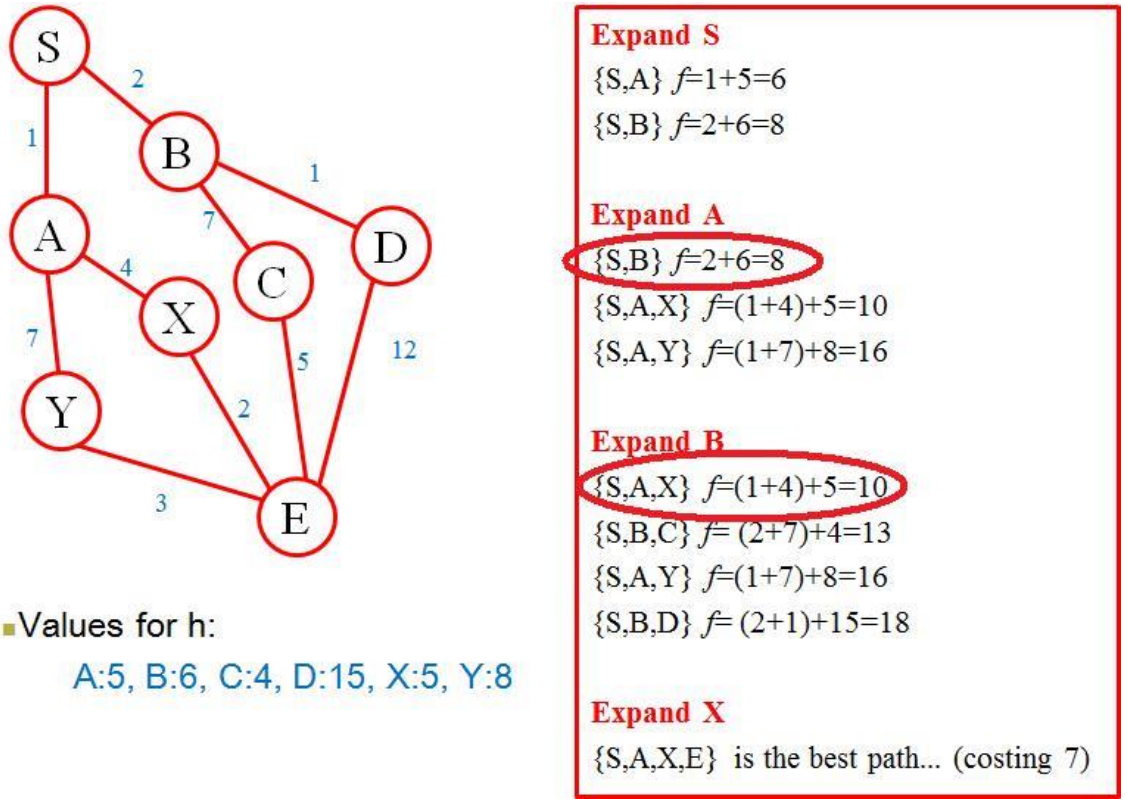

Рис. 1.4. Ілюстративна схема обходу A*

1.1.4. Алгоритм оптимізації мурашиних колоній

Оптимізація мурашиних колоній (*Ant Colony Optimization, ACO*) базується на моделюванні поведінки мурашиних колоній в природі, які здатні будувати найкоротші шляхи до джерел їжі використовуючи феромони [6].

В *ACO* шляхи графа моделюються ребрами, а вершини – вузлами. Кожне ребро має певний рівень штучного «феромону», який визначає ймовірність того, що цим ребром піде мураха. Феромони випаровуються з часом, але можуть поновлюватися проходженням мурах, особливо по ребрах високоякісних розв'язків.

Основні етапи алгоритму наступні:

– ініціалізація феромонів на ребрах графу певним початковим значенням;

– розміщення популяції мурах на стартових вершинах;

– конструювання розв'язку кожною мурахою шляхом переходу між вершинами по ребрах з ймовірністю, пропорційною рівню феромонів на них;

– оновлення рівнів феромону на всіх ребрах шляхом часткового випаровування;

– підсилення рівнів феромону на ребрах побудованих розв'язків пропорційно до їх якості;

– перехід до кроку 2 поки не буде досягнуто критерію зупинки.

Перевагою *ACO* є здатність знаходити якісні розв'язки складних оптимізаційних задач шляхом ітеративного вдосконалення численними агентами – «мурахами». Причому цей процес є децентралізованим та самоорганізованим.

Також варто зазначити хорошу масштабованість та можливості паралелізації ACO. Колонії з великою кількістю мурах можуть ефективно розподілятися на окремі групи для одночасного дослідження різних ділянок графа.

На рис. 1.5 проілюстровано схематичне уявлення про роботу алгоритму оптимізації мурашиних колоній та принцип формування оптимального шляху на основі «феромонних відкладень».

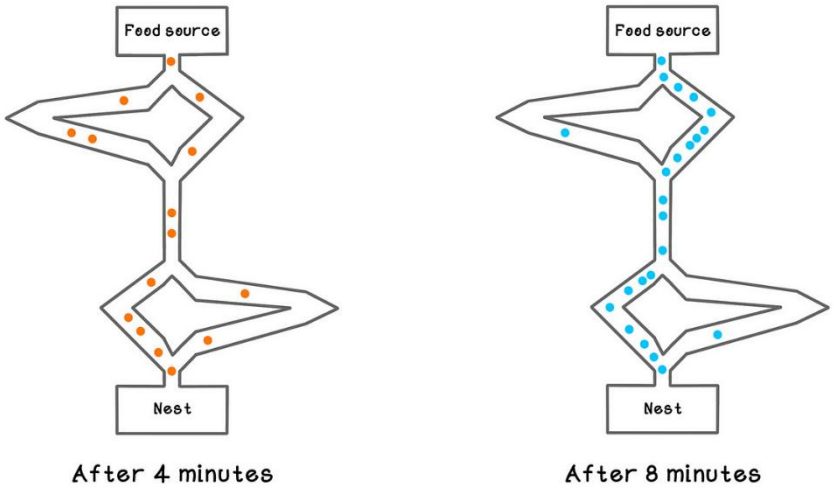

Рис 1.5. Ілюстрація роботи Оптимізації мурашиних колоній

Отже, *ACO* є перспективним підходом для побудови ефективних евристичних алгоритмів пошуку оптимальних шляхів, хоча й потребує значних обчислювальних ресурсів.

1.1.5. Оптимізація рою частинок (*Particle Swarm Optimization, PSO*)

Алгоритм оптимізації рою частинок базується на імітації соціальної поведінки децентралізованих живих організмів, таких як зграї птахів, рої бджіл, косяки риб тощо.

PSO використовує стохастичну оптимізацію розсіяним роєм частинок-агентів у багатовимірному просторі пошуку. Кожна частинка має поточне положення, швидкість та пам'ять про свою найкращу позицію. Також їй відоме поточне глобально оптимальне положення з усієї популяції [7].

Основні кроки алгоритму наступні:

– ініціалізація рою частинок з випадковими позиціями та швидкостями;

– обчислення значення цільової функції для кожної частинки;

– знаходження найкращої позиції для кожної окремої частинки та глобального оптимуму рою;

– розрахунок нової швидкості та позиції для кожної частинки на основі її поточної динаміки, власного оптимуму та глобального оптимуму рою;

– перехід до кроку 2 поки не буде досягнуто критерію зупинки.

PSO дозволяє ефективно знаходити глобальні оптимуми складних багатомодальних функцій за рахунок взаємодії та "обміну досвідом" між частинками рою.

Ключовими перевагами цього алгоритму є простота реалізації, масштабованість та можливості паралелізації. Також PSO досить стійкий до локальних оптимумів.

На рис. 1.6 проілюстровано схематичне уявлення про роботу алгоритму оптимізації рою частинок та принцип формування оптимального розв'язку на основі «розумної» поведінки агентів.

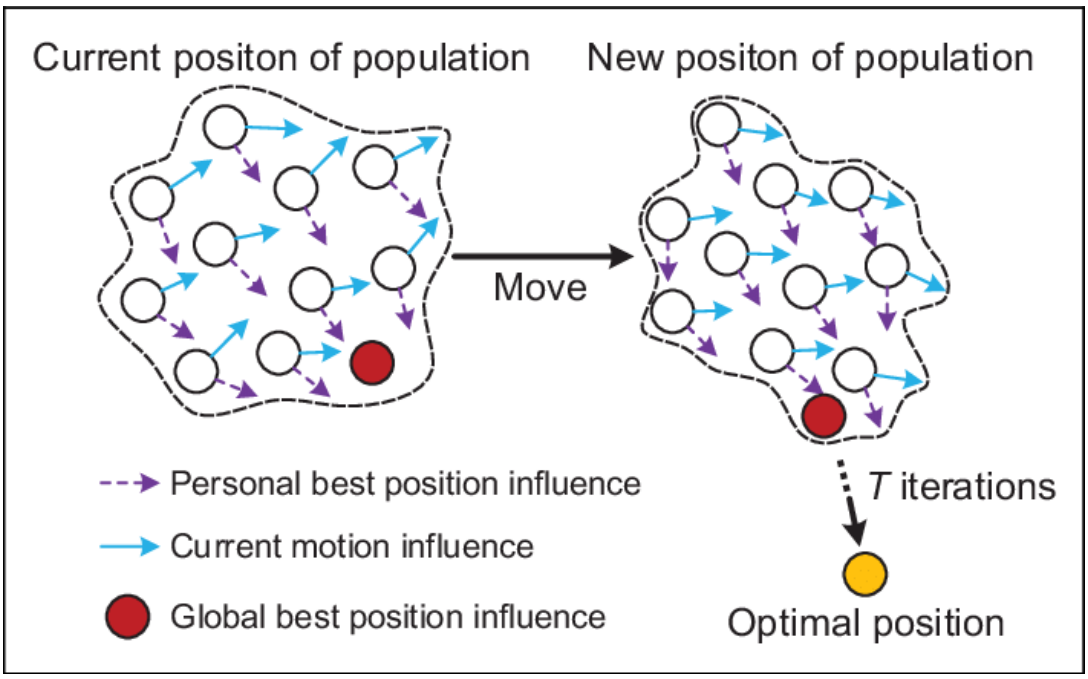

Рис. 1.6. Ілюстрація роботи оптимізації рою частинок

Отже, алгоритм *PSO* є досить привабливим та ефективним методом для вирішення складних задач глобальної оптимізації, у тому числі пов'язаних з плануванням траєкторій.

Було детально розглянуто п'ять перспективних алгоритмів планування шляхів для обчислення оптимальних маршрутів:

– *Breadth-First Search* гарантує найкоротший шлях завдяки систематичному обходу графа, але погано масштабується.

– Алгоритм Дейкстри підвищує ефективність за рахунок визначення пріоритетів на основі вартості, але йому бракує евристичних підказок.

– Алгоритм A* покращує інформований пошук, поєднуючи вартісні та евристичні оцінки.

– Оптимізація мурашиних колоній використовує обчислювальний інтелект рою для підвищення надійності.

– Оптимізація роєм частинок використовує популяційний пошук і розпаралелювання.

Кожен метод має відносні сильні та слабкі сторони.

1.2. Аналіз алгоритмів

Розглянувши різноманітні перспективні алгоритми планування маршрутів у попередньому розділі, прведено поглиблений теоретичний аналіз для порівняння їхніх формальних властивостей та очікуваної продуктивності під час виконання. Оцінка ключових показників оптимальності, складності, евристичності та повноти дасть змогу визначити придатність алгоритму для застосування в задачах маршрутизації БПЛА.

1.2.1. Докази оптимальності

Оптимальність алгоритму пошуку шляху – це математичний доказ того, що метод гарантовано завжди знаходить рішення з глобально мінімальною вартістю. Ця властивість забезпечує впевненість для додатків маршрутизації в тому, що згенерований шлях буде найефективнішим з можливих, не потрапляючи в пастку локальних мінімумів. Оцінено формальні гарантії оптимальності для кожного алгоритму.

Пошук в ширину неявно доводить оптимальність за допомогою механізму монотонного розширення вузлів. Завжди обираючи для дослідження найглибший невідвіданий вузол, *BFS* гарантує, що при першому відвідуванні будь-якого вузла вартість шляху до нього повинна бути мінімальною [8]. Оскільки кожен вузол відстежує свого батька з найменшою вартістю, реконструкція шляху за допомогою зворотних покажчиків, очевидно, дасть глобальний оптимальний маршрут.

Алгоритм Дейкстри забезпечує доказ оптимальності, подібний до *BFS*, але з ключовою відмінністю – пріоритети вузлів визначаються не за чистою глибиною, а за найнижчою відомою на даний момент вартістю. Це дозволяє спочатку розширювати більш перспективні гілки, щоб зменшити непотрібне дослідження дорогих піддерев. Але кожен вузол все одно оновлюється лише з найдешевшою знайденою вартістю, тому остаточна вартість найкращого шляху математично гарантовано буде глобально мінімальною.

Алгоритм A* успадковує оптимальність за вартістю від методу Дейкстри через функцію вартості *g(n)*. Крім того, з допустимою евристикою, яка ніколи не

переоцінює справжню вартість, функція оцінки *f(n)* також завжди буде недооцінювати загальну вартість шляху. Таким чином, при першому досягненні мети необхідно мінімізувати як фактичні, так і оціночні витрати, що забезпечить оптимальний розв'язок.

Оптимізація мурашиних колоній та оптимізація рою частинок, навпаки, не можуть забезпечити докази оптимальності через свою недетерміновану природу. Випадковість притаманна їхній механіці дослідження графів та збурення розв'язків. За достатню кількість ітерацій вони можуть випадково знайти глобальний оптимум, але не мають жодних гарантій для кожного окремого прогону. Натомість ці методи пропонують імовірнісну збіжність до все більш якісних рішень.

1.2.2. Обчислювальна складність

Ми аналізуємо обчислювальну складність алгоритму з точки зору порядку зростання часу виконання при збільшенні розміру задачі. Це визначає масштабованість для великих реальних екземплярів маршрутизації БПЛА з великими просторами пошуку. Нижчий порядок складності вказує на те, що для великих графів потрібно менше обчислень [9].

Пошук в ширину має експоненціальну складність $O(bd)$, де b – коефіцієнт розгалуження, а d – глибина розв'язку. Без евристик для відсікання, перебір всіх можливих шляхів призводить до експоненціального вибуху обчислень, оскільки середовище стає більшим або складнішим. Таким чином, *BFS* не має масштабованості для реального планування маршрутів БПЛА.

Алгоритм Дейкстри має трохи кращу складність $O(|E| + |V|log|V|)$, яка визначається ребрами графа E і вершинами V. Черга пріоритетів вершин може бути ефективно відсортована за логлінеарний час, але оновлення вартості всіх сполучних ребер все одно призводить до значної неефективності при масштабуванні. А великі карти пошуку шляхів можуть мати мільйони ребер.

Алгоритм A* вирішує цю проблему за допомогою евристик для відсікання неоптимальних гілок, досягаючи майже лінійної складності $O(|V|)$ у кращому випадку. Розширення лише корисних вершин дає величезні переваги при масштабуванні. Однак, у випадку нечітких евристик, час виконання у найгіршому

випадку залишається подібним до Дейкстри. Таким чином, жорстка евристика є ключовою для маршрутизації БПЛА над великими просторами.

Алгоритм A* вирішує цю проблему за допомогою евристик для відсікання неоптимальних гілок, досягаючи майже лінійної складності $O(|V|)$ у кращому випадку. Розширення лише корисних вершин дає величезні переваги при масштабуванні. Однак, у випадку нечітких евристик, час виконання у найгіршому випадку залишається подібним до Дейкстри. Таким чином, жорстка евристика є ключовою для маршрутизації БПЛА над великими просторами.

Оптимізація мурашиних колоній та оптимізація рою частинок мають поліноміальну часову складність. Але для отримання надійних розв'язків все одно може знадобитися багато мурах або частинок, що призводить до значних обчислювальних витрат. Їх розпаралелюваність допомагає пом'якшити цю проблему. В цілому вони досягають розумного масштабування для багатьох випадків маршрутизації БПЛА.

1.2.3. Евристичні функції

Оглянуто евристичні функції, які використовуються алгоритмами для оцінки вартості шляху, що залишився, без вичерпного перерахування всіх варіантів. Евристики покращують інформований пошук, дозволяючи оцінювати варіанти більш розумно [10].

Пошук в ширину принципово не містить евристик, що еквівалентно припущенню про нульове знання відстані до цілі. Таким чином, він даремно витрачає обчислення на дослідження очевидно обхідних маршрутів. Алгоритм Дейкстри також не має евристик, рівномірно розширюючи всі шляхи з мінімальною вартістю, не беручи до уваги перспективні напрямки.

Алгоритм A* спеціально використовує допустимі евристики, обчислюючи найнижчу можливу оптимістичну вартість досягнення мети. Прості евристики, такі як евклідова відстань по прямій, пропонують дивовижну ефективність для пошуку маршрутів. Більш просунуті методи можуть також включати специфічні для задачі обмеження та вартісні моделі для більш точних оцінок.

Оптимізація мурашиних колоній генерує легкі евристики за рахунок віртуальної концентрації феромонів на ребрах графа. Мурахи ймовірно рухаються вздовж феромонних стежок з більшою щільністю, що опосередковано кодує накопичену мудрість щодо перспективних маршрутів. Автонастроювана природа цих евристик дозволяє швидко реагувати на динамічні зміни в навколишньому середовищі.

Оптимізація рою частинок технічно також не має наперед визначених евристик. Але механізми взаємодії між частинками для виділення кращих областей рішення по суті встановлюють ефемерні евристики, закодовані щільністю частинок у просторі. Ці евристичні підказки допомагають частинкам сходитися в оптимальних точках.

1.2.4. Імовірнісна повнота

Ми досліджуємо алгоритмічні гарантії того, що в кінцевому підсумку буде знайдено розв'язок, якщо він існує, шляхом покриття графа за допомогою багаторазових ітерацій. Ця імовірнісна повнота вказує на стійкість навіть для складних ландшафтів маршрутизації [11].

Детерміновані алгоритми, такі як пошук в ширину, метод Дейкстри та A*, гарантують повноту, систематично перебираючи всі можливі шляхи, щоб гарантувати, що мета буде знайдена за умови припинення пошуку. Але ймовірнісні методи стають асимптотично повними лише тоді, коли ймовірність пропустити ціль зменшується при більшій кількості спроб, але не є визначеною для заданого часу виконання.

Таким чином, оптимізація мурашиних колоній та оптимізація рою частинок пропонують слабку повноту, що асимптотично наближається до 1. Вони все ще можуть зазнавати невдачі на окремих прогонах через випадковість у дослідженні. Вказівка меж ітерацій допомагає забезпечити розумні межі часу виконання, після чого вони можуть виводити поточний найкращий знайдений шлях, навіть якщо він неоптимальний.

Для додатків маршрутизації БПЛА критерії відсікання потребують балансування між впевненістю в оптимальності, ймовірнісною повнотою та обмеженнями на час виконання. Можуть знадобитися прагматичні пороги

припинення, що визначають пріоритетність можливих достатньо хороших траєкторій, а не абсолютного доказу оптимальності.

1.3. Порівняння та вибір алгоритму

1.3.1. Інфорована оцінка пошуку

Ефективне планування траєкторії польоту над величезними просторами вимагає розумно керованого пошуку, щоб відсікти неоптимальні варіанти на ранніх стадіях, не витрачаючи обчислення на обхідні шляхи. Ми порівнюємо алгоритми використання евристичної інформації для керування пошуком.

Пошук в ширину принципово ігнорує будь-які евристичні підказки, рівномірно розширюючи вузли шляхом монотонного збільшення довжини шляху. Цей графічний еквівалент ручного пошуку в сітці витрачає величезні зусилля на обробку нерелевантних гілок і постійно наштовхується на обмеження глибини. *BFS* не забезпечує прискорення пошуку маршруту.

Алгоритм Дейкстри покращує жорстке розширення, надаючи пріоритет невідвіданим вершинам з найнижчою на даний момент вартістю шляху. Однак без евристик, що прогнозують кінцеву ефективність траєкторії для досягнення мети, він все одно вичерпно розгортає всі найдрібніші часткові шляхи наосліп. Дейкстра не помічатиме очевидних прямих маршрутів, зосереджуючись на заплутаних обхідних шляхах.

Лише алгоритм A* спеціально включає евристичне наведення, використовуючи базові відстані для воронячого польоту, відомі авіаційні небезпеки або прогнозні геопросторові моделі. Об'єднуючи поточні витрати на маршрут з майбутніми оцінками витрат, він може обрізати гілки, які можуть призвести до глобально неоптимальних рішень на ранній стадії, без необхідності спочатку повністю перераховувати всі перестановки. Це значно підвищує ефективність інформованого пошуку.

Модульна конструкція також дає змогу поєднувати евристики ансамблю та машинне навчання для покращення наведення. Прості прозорі оцінки відстані вже

багаторазово підвищують продуктивність порівняно з некерованими методами. Більш складні нейромережеві моделі, що прогнозують загрози безпеці або витікання палива, можуть ще більше підвищити якість маршрутизації польотів після їхньої валідації.

1.3.2. Оцінка управлення невизначеністю

Реальне авіаційне планування повинно враховувати стохастичні впливи, такі як турбулентний вітер, неповні картографічні дані або рухомі перешкоди. Алгоритми порівнюються за стійкістю до невизначеностей.

Детермінований пошук за допомогою *BFS* або алгоритму Дейкстри не дає прямого доступу до несподіваних відхилень, окрім повного перепланування, якщо припустити, що зміни навіть можна було спостерігати. Вони жорстко припускають незайманe середовище і повну інформацію, не в змозі адаптувати пошук до непередбачуваних ситуацій, окрім як перервати його. Будь-яка невідомість ризикує поставити під загрозу безпеку польоту.

Методи ройового інтелекту, змодельовані на рандомізованій висхідній координації природних колективів, за своєю суттю охоплюють невизначеність як частину їхньої стохастичної оптимізації. Алгоритми мурашиних колоній і рою частинок використовують дослідження розподілених агентів для ймовірнісного зближення рішень, незважаючи на шум або обман. Вони забезпечують надійність в умовах невідомості, амортизуючи випадковість.

Але надмірне покладання на безцільну рандомізацію, а не на обґрунтований пошук, має такі ж ризики затримок для критично важливої логістики БПЛА, якщо нестабільність навколишнього середовища переважає над дифузією. Гарантії все ще мають значення для високопріоритетних польотів, навіть якщо їх послаблюють для попередніх дослідницьких місій. Алгоритми чистого рою також приховують операційні тонкощі від кінцевих користувачів.

A* досягає балансу, розширюючи ширину пошуку у відповідь на виявлену невизначеність на основі евристичної вартості, а не сліпої випадковості. Ця прозорість зберігає принципову оптимальність для рутинних операцій, на відміну від хаотичних підходів. Модульна архітектура ансамблю також ізолює імпровізаційні

компоненти управління непередбачуваними ситуаціями від основного планувальника.

1.3.3. Показники масштабованості

Алгоритм маршрутизації БПЛА повинен обробляти величезні пошукові простори на сотнях кілометрів польотів, що складаються з нескінченних точок прийняття навігаційних рішень та обмежень на висоту місцевості.

Як вичерпний пошук в ширину, так і пошук найкоротшого шляху Дейкстри страждають від експоненціальних вибухів складності при збільшенні ширини проблеми та об'єму інформації. Спроба явно обчислити всі можливі траєкторії через континентальні простори є обчислювально нездійсненною, що призводить до неприйнятних затримок навіть при використанні кластерів серверів. Вони абсолютно не піддаються масштабуванню на практиці.

Ройові методи використовують властивий їм паралелізм і децентралізацію для розподілу навантаження, але чиста випадковість все одно призводить до величезної надмірності, аналізуючи безглузді варіанти там, де обґрунтований напрямок значно прискорив би збіжність. Їх комунікаційні накладні витрати також часто обмежують ефективність масштабування розподілу, незважаючи на багатообіцяючу передумову [12].

Багатообіцяюче, що A* демонструє більш лінійні поліноміальні темпи зростання завдяки евристичній селективності наведення, яка усуває нерелевантні гілки, що роздувають некеровані пошуки. Він уникає марних обчислень завдяки інформативному проектуванню контурів витрат. Легке розпаралелювання ще більше підвищує масштабованість, розділяючи масивні сітки даних про місцевість без вузьких місць у зв'язку з комунікаціями. Це дає змогу оперативно планувати навіть трансконтинентальні перельоти на побутовому обладнанні.

Керуючи розкладанням полінома, A* поєднує принциповий пошук з масштабованою здатністю обробляти задачі маршрутизації польотів різного розміру. Підвищення точності наведення та паралелізму за допомогою модульних ансамблів ще більше підвищує цю адаптивність для планування польотів у планетарних масштабах, зберігаючи при цьому гарантії цілісності. Жоден альтернативний метод

не забезпечує інформовану ефективність одночасно з масштабованими розподіленими обчисленнями.

Таким чином, алгоритм A* унікально поєднує евристичну інформацію для керування маршрутом польоту зі стійкістю до невизначеності, розширюючи пошук, обмежений контурами витрат. Його прозора модульна конструкція також легко масштабується на великих пошукових просторах завдяки вибірковому розширенню і розпаралелюванню. Ці різнобічні переваги роблять A* ідеальним інструментом оптимізації для планування доставки БПЛА.

Висновки за розділом

У даному розділі було проведено ретельний аналіз та порівняння найбільш перспективних алгоритмів пошуку оптимальних маршрутів для систем маршрутизації безпілотних повітряних суден. Розглядалися такі п'ять основних алгоритмів: пошук у ширину, Дейкстри, А зірка, оптимізація колонії мурах та оптимізація рою частинок. Кожен алгоритм детально проаналізовано за низкою ключових критеріїв, а саме: математичні гарантії знаходження глобально оптимального шляху; обчислювальна складність алгоритму та його масштабованість; використання евристичних функцій для ефективного інформованого пошуку; стійкість алгоритму до невизначеностей та динамічних змін середовища.

За підсумками дослідження можна констатувати, що алгоритм A* продемонстрував найкращі результати за більшістю проаналізованих критеріїв порівняно з альтернативними методами. Зокрема, він має формальні математичні гарантії знаходження глобально оптимального маршруту за найменшу вартість. При цьому алгоритм A* використовує евристичні функції оцінки для ефективного інформованого пошуку, фокусуючись на найбільш перспективних напрямках. Ще однією важливою перевагою A* є його адаптивність та стійкість до невизначеностей – він може гнучко реагувати на зміни середовища, розширюючи або звужуючи пошук. Водночас, завдяки механізму відсікання малоймовірних гілок,

алгоритм A* демонструє хорошу масштабованість на величезних просторах пошуку, що є вкрай важливим для задач маршрутизації БПЛА.

Таким чином, алгоритм A* оптимально поєднує в собі теоретичні гарантії та практичну ефективність, що робить його ідеальним вибором для ядра системи маршрутизації безпілотників. Перспективними напрямками подальших досліджень є розробка спеціалізованих евристичних функцій оцінки вартості з урахуванням предметної області БПЛА, а також масштабування та паралелізація обчислень алгоритму A* для підвищення його продуктивності при плануванні маршрутів на сотні кілометрів.

Додатковою перевагою алгоритму A* є простота його практичної реалізації. На відміну від деяких інших розглянутих методів, A* не вимагає складних обчислень або спеціалізованої апаратної підтримки. Це робить A* придатним для втілення на бортовому комп'ютері самого БПЛА, що забезпечить автономність маршрутизації. Окрім цього, зрозуміла структура та відносна простота коду алгоритму A* полегшують його подальше вдосконалення та налаштування до конкретних задач планування маршрутів БПЛА. Отже, алгоритм A* є не лише ефективним, але й зручним та гнучким рішенням для практичної реалізації на бортовій системі безпілотників.

РОЗДІЛ 2
ОСОБЛИВОСТІ ПРОКЛАДАННЯ МАРШРУТІВ ДЛЯ БПЛА

2.1. Технічні характеристики та принципи функціонування БПЛА

Безпілотні літальні апарати (БПЛА), або дрони – це повітряні судна, здатні здійснювати політ без пілота на борту автономно або під керуванням оператора. Вони можуть бути обладнані різноманітним корисним навантаженням і виконувати широке коло завдань, таких як аерофото- та відеозйомка, транспортування легких вантажів, екологічний та промисловий моніторинг, інспекція інфраструктури, пошуково-рятувальні операції тощо [13].

Існує велика різноманітність типів БПЛА, які прийнято класифікувати за:

– типом аеродинамічної схеми (кількість та розташування несучих елементів) наведено в табл. 2.1;

– призначенням та характером використання;

– розмірами та масою;

– тривалістю та дальністю польоту.

Таблиця 2.1

Типи БПЛА за аеродинамічною схемою

Тип БПЛА	**Кількість несучих гвинтів**	**Можливість верт. зльоту/посадки**	**Дальність польоту**	**Максимальна швидкість**	**Практична стеля польоту**	**Приклади моделей**
Квадрокоптери	4	Так	1-2 години	50 *км/год*	30-40 хвилин	*DJI Mavic Air 2, Parrot Anafi*
1	**2**	**3**	**4**	**5**	**6**	**7**

Закінчення таблиці 2.1

1	**2**	**3**	**4**	**5**	**6**	**7**
Гекса-коптери	6	Так	2-4 години	60 *км/год*	40-60 хвилин	*Yuneec Typhoon H Pro, Freefly Alta 6*
Окта-коптери	8	Так	3-6 годин	70 *км/год*	60-90 хвилин	*DJI Matrice 600, Freefly Alta 8*
Крилаті БПЛА	Фіксоване крило	Неможливо	8-24 години	150-300 *км/год*	4-16 годин	*senseFly eBee X, Delair UX11*
Гібридні	Крило + несучі гвинти	Так	4-8 годин	80-120 *км/год*	3-6 годин	*Vertical Aerospace VA-1X, Skyfront Perimeter*

Квадрокоптери – 4 несучі гвинти у вершинах рами-хреста. Маневрені, здатні до вертикальних зльоту та посадки. Менша дальність та тривалість польоту. Гекса- та октокоптери – 6 або 8 гвинтів відповідно. Надлишкові за числом гвинтів для підвищеної надійності. Призначені для професійної зйомки. Крилаті БПЛА – фіксоване крило, фюзеляж, хвостове оперення. Використовують аеродинамічну силу для забезпечення основної частки підйомної сили. Здатні долати тисячі *км* і годинами перебувати у польоті. Гібридні і несучих гвинтів. Найчастіше трикутний або *Y*-подібний планер з 2 і більше гвинтами.

За призначенням БПЛА поділяють на:

– любительські – для відео- та фотозйомки, розваг як хобі;

– комерційні – для виконання професійних знімальних робіт;

– спеціального призначення – пошукові, розвідувальні, військові;

– дослідницькі – для наукових досліджень атмосфери, ґрунтів тощо.

За розмірами розрізняють мікро- (<50 *см*), міні- (0,5-2 *м*), середні (2-5 *м*) та великі (>5 *м*) БПЛА [14]. Порівняння наведено в табл. 2.2.

Таблиця 2.2

Класифікація та призначення різних типів БПЛА

Клас БПЛА	**Призначення**	**Типова вага**	**Корисне навантаження**	**Тривалість польоту**
Мікро- (<50 см)	Відео- та фотозйомка, розваги як хобі	Легка	Камера, вбудований мікрофон	20-30 хвилин
Міні- (0,5-2 м)	Професійна зйомка, дослідження	Середня	Камера, лідар	30-60 хвилин
Середні (2-5 м)	Професійна зйомка, розвідування, комерційні застосування	Важка	Камера, лідар, газоаналізатор	2-8 годин
Великі (>5 м)	Військове застосування, наукові дослідження	Дуже важка	Різноманітне	8-24 години

Принципи функціонування та основні системи БПЛА [15].

БПЛА складаються з чотирьох функціональних підсистем:

– силова установка – електродвигуни з гвинтами, акумулятори, система керування польотом. Створює необхідну тягу та підйомну силу;

– система керування польотом – автопілот, датчики (*GPS*, акселерометри), мікроконтролери та ПЗ. Забезпечує стійкість, навігацію за заданою траєкторією;

– корисне навантаження – обладнання для виконання цільових завдань БПЛА (камери, лідари, газоаналізатори тощо);

– система зв'язку – радіомодулі та антени для управління БПЛА та обміну телеметрією з наземною станцією керування;

Також порівняння наведене в табл. 2.3.

Таблиця 2.3

Основні підсистеми БПЛА та їх функції

Підсистема	**Призначення**	**Основні компоненти**
Силова установка	Тяга, підйомна сила	Електродвигуни, гвинти, акумулятори, система керування польотом
Система керування польотом	Стійкість, навігація	Автопілот, датчики (*GPS*, акселерометри), мікроконтролери, ПЗ
Корисне навантаження	Виконання цільових завдань БПЛА	Камери, лідари, газоаналізатори, інші датчики
Система зв'язку	Управління та телеметрія з наземною станцією	Радіомодулі, антени

Ключові характеристики БПЛА, що впливають на вибір маршруту та режимів польоту:

– діапазон і тривалість польоту – визначаються ємністю акумуляторів, ефективністю двигунів та аеродинамічними якостями БПЛА;

– максимальна швидкість – залежить від потужності силової установки, маси та аеродинамічного опору БПЛА;

– практична стеля польоту – обмежена потужністю двигунів, необхідною для подолання сили тяжіння;

– вантажопідйомність – визначається міцністю конструкції та запасом потужності.

Порівняння по перерахованим характеристикам наведене в табл. 2.4.

Таблиця 2.4

Порівняльні характеристики різних класів БПЛА

Тип/клас БПЛА	**Дальність польоту**	**Макс. швидкість**	**Практ. стеля**	**Макс. злітна вага**	**Корисне навантаження**
Мікро- (<50 см)	20-30 хвилин	50 *км/год*	30-40 хвилин	2-5 *кг*	Обмежена вага, малі камери
Міні- (0,5-2 м)	30-60 хвилин	80 *км/год*	30-90 хвилин	5-20 *кг*	Різноманітне обладнання
Середні (2-5 м)	2-8 годин	120 *км/год*	2-24 години	20-100 *кг*	Різноманітне обладнання
Великі (>5 м)	8-24 години	150-300 *км/год*	8-24 години	100+ *кг*	Різноманітне обладнання

Отже, технічні можливості конкретного БПЛА істотно обмежують вибір оптимального маршруту. Сучасні алгоритми повинні комплексно враховувати ці параметри при плануванні маршрутів для підвищення ефективності.

2.2. Фактори, що впливають на вибір маршруту БПЛА

Ефективне та безпечне планування маршрутів для безпілотних літальних апаратів потребує детального аналізу та врахування широкого спектру чинників, як технічного, так і операційного характеру. Ці фактори впливу можна умовно розділити на три основні групи:

– технічні характеристики та можливості БПЛА;

– географічні та метеорологічні умови;

– правові, етичні та соціальні обмеження.

Розглянемо детальніше кожну з цих груп чинників та їх вплив на вибір маршруту.

2.2.1. Технічні характеристики та можливості БПЛА

БПЛА є технічними пристроями, функціональність яких жорстко обмежена їх конструктивними особливостями та характеристиками комплектуючих.

Тривалість автономної роботи без посадки є однією з ключових характеристик будь-якого безпілотного літального апарату. Цей параметр визначає можливості виконання тривалих операцій без повернення на базу для заміни акумуляторів чи дозаправки.

Тривалість польоту залежить від декількох чинників:

– ємність встановлених на борту акумуляторів. Сучасні літій-полімерні батареї мають велику густину енергії при порівняно невеликій вазі, що дозволяє збільшувати час автономної роботи БПЛА;

– економічність та ККД двигунів та силової установки. Використання більш досконалих безколекторних електродвигунів постійного струму, оптимізація гвинтів під режими польоту, зниження механічних втрат та опорів дозволяє підвищити загальний час польоту;

– ефективність польоту на маршруті. Раціональне планування операційної зони та профілю польоту (висоти, швидкості) відповідно до умов дозволяє економити заряд батарей та максимально використовувати технічні можливості БПЛА щодо тривалості автономної роботи.

Загалом для сучасних безпілотників цей час становить від 20 хвилин для невеликих мультикоптерів до 35 годин для висотних розвідувальних літаків на кшталт *Global Hawk*[16]. Окремі рекордсмени, такі як британський сонячний планер-розвідник *Zephyr*, здатні перебувати в польоті понад 2 тижні.

При плануванні маршруту для БПЛА його тривалість польоту є вкрай важливим обмеженням, яке обов'язково має враховуватися:

– сумарний запланований час у дорозі між усіма заданими точками жодним чином не може перевищувати максимальної тривалості автономної роботи безпілотника, інакше він просто не зможе завершити маршрут і успішно повернутися на базу;

– при запасі лише в кілька хвилин до ліміту тривалості польоту істотно зростають ризики раптового падіння через неможливість керованої посадки в разі навіть незначного перевищення граничного часу через неточності в розрахунках або зміну умов.

Тому виключно важливо запланувати загальний час операції з комфортним запасом 20-30% до межі автономності. Це дасть можливість БПЛА безпечно повернутися на базу навіть за нештатних ситуацій. А для особливо тривалих і відповідальних операцій доцільно планувати проміжні посадки в точках з можливістю заміни батарей.

Отже, тривалість польоту є однією з ключових характеристик БПЛА, що жорстко обмежує радіус його дій та накладає важливі обмеження при плануванні маршруту. ЇЇ обов'язкове врахування є запорукою успішного виконання місій та запобігання нештатним ситуаціям, пов'язаним з раптовим виснаженням батарей. А гнучке резервування часу відповідно до пріоритетності завдання є важливим резервом підвищення надійності польотів БПЛА.

Можлива швидкість визначається потужністю силової установки та аеродинамічними властивостями безпілотника, зокрема його формою, розмірами, конструкцією. Наприклад, невеликі багатогвинтові коптери обмежені швидкістю 40-60 *км/год* через високий опір повітря, в той час як спеціальні БПЛА для перехоплення здатні розвивати швидкість понад 500 *км/год*.

Вибір маршруту повинен враховувати це обмеження для коректного розрахунку часу в дорозі між контрольними точками. Також для деяких місій, наприклад інспекції ЛЕП, необхідна певна мінімальна швидкість для сталого управління.

Максимально можлива висота польоту та швидкість є важливими технічними характеристиками безпілотних літальних апаратів, що впливають на вибір оптимального маршруту.

Висота польоту визначається насамперед потужністю силової установки, достатньою для подолання сили тяжіння на певній висоті, а також необхідною для горизонтального переміщення з заданою швидкістю. Чим вище піднімається БПЛА,

тим розрідженіше повітря та менша підйомна сила, яку створюють гвинти. Відповідно на певному рівні виникає гранична висота, де потужності двигунів вже не вистачає для компенсації сили тяжіння. Це максимальна стеля польоту для даного апарату.

Можлива висота також залежить від конструктивної міцності БПЛА, зокрема герметичності корпусу двигунів, електроніки та інших систем. Розріджене повітря на великій висоті створює додаткові виклики для охолодження компонентів.

Максимальна швидкість горизонтального польоту визначається потужністю силової установки, необхідною для подолання аеродинамічного опору повітря на заданій висоті. Чим гірші аеродинамічні якості БПЛА та вищий лобовий опір, тим більше потужності потрібно для розгону та польоту з певною швидкістю.

Висотні швидкісні режими також потребують підвищеної конструктивної міцності і жорсткості рами безпілотника для запобігання небезпечним вібраціям, резонансам та руйнуванню елементів конструкції.

Для різних типів БПЛА характерні суттєві відмінності за цими параметрами:

– невеликі мультикоптери здатні маневрувати лише на висоті до 1 *км* зі швидкістю 40-80 *км/год;*

– великі літакові розвідувальні БПЛА типу *RQ-4 Global Hawk* [16] здійснюють польоти на оптимальних висотах понад 15 *км* зі швидкістю до 950 *км/год*, що забезпечує значну площу огляду;

– окремо слід згадати експериментальні висотні безпілотні планери, здатні підніматися на 20-30 *км* і виконувати тривалі спостереження та дослідження атмосфери.

Врахування обмежень конкретного БПЛА за максимальною висотою та швидкістю польоту є обов’язковим при плануванні його маршруту:

– по-перше, апарат фізично не зможе подолати перешкоди, вищі за його стелю (гірські хребти, висотні будівлі тощо);

– по-друге, при швидкісних режимах вище технічних можливостей безпілотника існує висока ймовірність втрати керованості та нештатних ситуацій;

– по-третє, експлуатація на межі висотних та швидкісних можливостей істотно

підвищує зношеність та перевантаження конструкції, скорочуючи ресурс силової установки та авіоніки.

Таким чином, врахування даних характеристик при маршрутизації дає змогу уникнути небезпечних та непрохідних ділянок, а також оптимізувати режими польоту для збільшення надійності та ефективності функціонування БПЛА певного класу.

Під корисним навантаженням безпілотних літальних апаратів розуміють устаткування та прилади, необхідні для виконання цільових задач польоту. Це можуть бути різноманітні датчики та сенсори, зокрема:

– фото- та відеокамери для аерозйомки;

– тепловізійні та інфрачервоні камери;

– лідарні комплекси для сканування поверхні[17];

– газоаналітичне обладнання;

– ретранслятори зв'язку та радіоелектронної розвідки;

– транспортні відсіки та механізми скидання вантажів.

Маса та розміри такого обладнання безпосередньо залежать від типу безпілотника та його цільового призначення. Зокрема, для невеликих апаратів характерне навантаження до 1 *кг*, тоді як великі розвідувальні чи ударні БПЛА здатні нести кілька сотень кілограмів різноманітних пристроїв.

При плануванні маршруту показник корисного навантаження впливає на наступні аспекти:

– Злітна вага. Збільшення маси устаткування зменшує запас потужності двигунів, необхідний для вертикального набору висоти та компенсації аеродинамічного опору на маршруті.

– Аеродинаміка та стійкість. Встановлення зовнішніх блоків приладів може погіршувати аеродинамічні характеристики літального апарату, що призводить до зростання енергоспоживання та зниження стійкості.

– Розподіл центру мас. Несиметричне кріплення обладнання викликає зсув центру мас БПЛА, що потребує додаткових зусиль системи стабілізації для збереження керованості.

– Габарити та компонування. Громіздке устаткування обмежує можливості

установки інших систем, ускладнює транспортування та погіршує доступ до критичних компонентів.

– Енергоспоживання. Живлення та охолодження приймачів та передавачів корисного навантаження може істотно впливати на загальну тривалість польоту безпілотника.

Отже, правильний вибір оптимального складу обладнання та його розміщення на борту є важливим фактором, що визначає аеродинамічні характеристики, керованість, стійкість та тривалість польоту безпілотного літального апарату. Ці аспекти мають враховуватися при виборі профілю та параметрів маршруту з метою ефективного виконання поставлених завдань.

Ефективна та надійна експлуатація безпілотних літальних апаратів вимагає стійкого і безперервного зв'язку між апаратом та пультом людини-оператора. Це досягається за рахунок встановлення на борту радіопередавачів та антен для обміну телеметрією та командами керування з наземною станцією.

Можлива дальність зв'язку залежить від типу та потужності радіозасобів, умов поширення радіохвиль, наявності перешкод, рельєфу місцевості тощо. Для невеликих БПЛА з використанням стандартних *Wi-Fi* технологій (2,4 - 5,8 *ГГц*) характерний радіус дії становить 1-10 км за умови прямої видимості [18]. Дальність керування потужніших апаратів з використанням направлених антен, радіорелейних ліній та супутникових каналів може сягати сотень кілометрів.

Вимоги до забезпечення прямої видимості та радіусу керування істотно обмежують прийнятні операційні зони та маршрути для БПЛА:

– політ за межі прямої видимості чи радіусу дії може призвести до втрати керованості та падіння апарата;

– перешкоди у вигляді узвиш, будівель, дерев, пагорбів переривають лінію радіозв'язку та ускладнюють керування апаратом;

– ккранування сигналу виникає при польотах в умовах густої міської забудови, гірської місцевості, лісових масивів тощо, що також погіршує керованість БПЛА та може призвести до його втрати.

Таким чином, при плануванні маршруту для безпілотного апарату слід враховувати радіус його керування та умови, необхідні для безперешкодного поширення радіосигналів. Особливо важливо уникати потенційно проблемних ділянок шляху зі складним рельєфом, густою забудовою, лісових масивів тощо. Це дозволить мінімізувати ризики втрати зв'язку та забезпечити надійне дистанційне керування безпілотником впродовж усього польоту за маршрутом.

2.2.2. Географічні та метеорологічні умови

Окрім особливостей безпілотника на вибір маршруту впливають також умови місцевості та погодні явища.

Рельєф та характер місцевості. При плануванні маршруту необхідно враховувати рельєф та особливості ландшафту по лінії шляху БПЛА. Такі перешкоди, як ліси, гори, висотні будівлі та споруди, вимагають набору достатньої висоти для безпечного їх обльоту. В іншому разі існує високий ризик зіткнення та падіння безпілотника.

Окрім цього, деякі райони надто щільної міської чи гірської забудови, густолісся можуть критично знижувати якість *GPS* сигналу, що негативно позначається на навігації та стабілізації БПЛА. Відповідно, подібні зони варто обходити для запобігання втрати контролю.

Загалом, складний рельєф та щільна забудова значно ускладнюють маршрутизацію, тому для багатьох БПЛА рекомендуються переважно відкриті місцевості та слабонаселені райони.

Метеорологічні умови. На керованість та безпеку БПЛА істотно впливають всі основні погодні явища:

– опади та хмарність ускладнюють візуальне спостереження та роботу оптичних датчиків, але покращують роботу тепловізорів чи радарів [19];

– сильний дощ та сніг створюють додаткове навантаження на конструкцію БПЛА, деформують обтічники та знижують підйомну силу гвинтів;

– висока турбулентність та поривчастість вітру значно ускладнюють роботу автопілота зі стабілізації та можуть призвести до занесення або аварії;

– грозова діяльність та блискавки становлять смертельну загрозу через ризик

ураження струмом.

Отже, для безпечного польоту необхідно ретельно аналізувати поточні та прогнозні метеоумови при плануванні операцій з БПЛА, враховуючи сезонні коливання та регіональні особливості, для уникнення загроз та мінімізації ризиків.

2.2.3. Правові, етичні та соціальні обмеження

Незважаючи на технічні можливості сучасних безпілотників, існують значні правові та етичні обмеження на використання повітряного простору на різних територіях та об'єктах.

Заборонені для польотів зони:

– над багатьма військовими, прикордонними, стратегічними, природоохоронними та іншими важливими об'єктами встановлюються спеціальні обмеження на польоти БПЛА [20];

– навколо аеропортів, уздовж повітряних трас, над залізницями і лініями електропередач виділяються буферні зони безпеки радіусом до 30 км, де польоти БПЛА заборонені чи обмежені певними висотами;

– над населеними пунктами та окремими будівлями (особливо адміністративними) можуть встановлюватися локальні обмеження на польоти БПЛА без спеціальних дозволів та узгоджень маршрутів.

Відповідно, для отримання всіх необхідних правових погоджень та уникнення порушень, маршрути польотів БПЛА мають обов'язково враховувати зазначені обмежені зони, заборонені для польотів.

Приватна власність:

– виконання аерозйомки чи просто проліт над приватними будинками, садибами, господарськими та виробничими об'єктами без згоди власників може розглядатися як порушення прав власності та особистого життя;

– можуть накладатися обмеження щодо польотів над присадибними ділянками у межах населених пунктів;

– використання БПЛА для навмисного спостереження за людьми на приватних територіях також часто розцінюється як кримінальний злочин.

Тому планування маршрутів польотів БПЛА над приватними територіями має бути мінімальним, а також враховувати встановлені національним законодавством особливості. Потрібно отримувати всі необхідні узгодження від власників для запобігання можливим конфліктам та судовим позовам.

Соціальні обмеження знизу визначаються громадськими нормами та суспільними очікуваннями щодо приватності і безпеки:

– існують серйозні етичні занепокоєння щодо використання «дронів», які технічно можуть порушувати розумні очікування людини про приватність;

– польоти БПЛА з камерами над житловими районами, пляжами, іншими громадськими місцями часто розглядаються як антисоціальні дії;

– фізичні травми, пошкодження майна, інші нещасні випадки при падінні мультикоптерів також посилюють негативне ставлення суспільства;

– приймаються додаткові нормативні акти для обмеження загальнодоступного використання БПЛА без попереднього навчання, ліцензування, страхового покриття для забезпечення безпеки громадян.

Отже, врахування регуляторних, правових, етичних та соціальних обмежень є критично важливим для забезпечення безпечного, відповідального та суспільно прийнятного застосування безпілотних технологій. Особливу увагу слід приділяти проектуванню маршрутів польоту БПЛА з обов'язковим уникненням заборонених зон, мінімізацією порушення приватності людей, запобіганням можливим нещасним випадкам та шкоді матеріальним цінностям. Успішна реалізація цих заходів дозволить забезпечити суспільне сприйняття та поступове широке розповсюдження безпілотних технологій.

2.3. Врахування факторів впливу при плануванні маршрутів БПЛА

Ефективне функціонування систем автоматизованого планування маршрутів для безпілотних літальних апаратів можливе лише за умови всеохоплюючого аналізу та врахування сукупності впливових факторів, проаналізованих у попередніх розділах. Ці чинники можна класифікувати на:

– технічні характеристики БПЛА;

– географічні особливості території;

– метеорологічні умови;

– правові та етичні обмеження.

Врахування цих факторів дозволяє значно підвищити безпеку та надійність функціонування БПЛА за рахунок попередження нештатних ситуацій, а також дає змогу суттєво оптимізувати маршрути з огляду на конкретні умови виконання місії та можливості конкретного безпілотного літального апарату.

В цьому підрозділі докладно розглядаються методи формалізації згаданих факторів та особливості їх врахування на етапах попереднього та оперативного планування маршрутів в рамках автоматизованих систем.

Першим кроком алгоритмізації врахування чинників впливу є їх формалізоване подання у складі математичної моделі оптимізаційної задачі планування маршруту для БПЛА. Зокрема, в межах даної задачі можна виділити:

Цільову функцію – кількісний критерій ефективності маршруту, що підлягає оптимізації. Найчастіше використовують функції мінімізації загальної вартості або часу маршруту.

Обмеження – фактори, що накладають додаткові вимоги та нерівності, яким повинен задовольняти припустимий маршрут.

Параметри оптимізації – змінні величини, значення яких варіюються з метою екстремізації цільової функції, наприклад вузли маршруту чи час виконання місії.

В рамках запропонованого підходу фактори впливу формалізуються наступним чином:

Технічні характеристики БПЛА:

– ліміти дальності, тривалості та висоти польоту задають обмеження;

– вантажопідйомність визначає допустиму вагу устаткування;

– маневрені параметри впливають на цільову функцію вартості.

Географічні особливості:

– цифрова модель рельєфу задає обмеження за висотою та прохідністю;

– типи ґрунтів, рослинності впливають на вартість маршруту;

– перешкоди опосередковано накладають додаткові обмеження.

Метеорологічна ситуація:

– прогнозні дані щодо опадів, вітру, хмарності коригують базову цільову функцію;

– окремі екстремальні явища (гроза) можуть забороняти вильоти.

Правові та етичні обмеження:

– задаються у вигляді заборонених для польотів зон та ділянок;

– також впливають на цільову функцію шляхом штрафних вагів.

Така формалізація дає змогу системно враховувати весь спектр чинників в межах єдиної оптимізаційної задачі планування маршрутів.

Адаптація алгоритму планування маршруту. Після формалізації факторів наступним кроком є адаптація базових алгоритмів пошуку оптимального шляху для прийняття додаткових правил, обмежень та цільових функцій.

Як показано в підрозділі 1.3, найбільш оптимальним та ефективним методом планування маршрутів, з огляду на критерії якості та продуктивності, є алгоритм A*. Відповідно, саме його пропонується взяти за основу для подальшої адаптації.

Модифікований алгоритм A* з врахуванням додаткових факторів впливу може бути описаний псевдокодом: ініціалізувати початковий і кінцевий вузли маршруту, хеш-таблиці відкритих та закритих вузлів; оновити евристику на основі поточних метеоданих та геопросторових моделей; додати початковий вузол у відкриту множину з f(початок) = g(початок) + h(початок); поки закрита множина не містить кінцевий вузол: вибрати вузол n з відкритої множини з мінімальним $f(n)$; перемістити n у закриту множину; для кожного сусіднього вузла m суміжного з n: перевірити задоволення обмежень: ліміти БПЛА, заборонені зони тощо; обчислити: вартість переходу $c(n,m)$ з урахуванням геоданих та погоди; обчислити: вартість шляху $g(m)$ через вузол n; якщо $g(m)$ краща ніж попередньо знайдена або вузол не відвіданий: записати попередником m вузол n, додати або оновити відкриту множину з $f(m)$; повернути оптимальний шлях, збережений у закритій множині.

Ключовими особливостями цього алгоритму є:

– можливість динамічного оновлення евристичної функції на основі

актуальних даних;

– перевірка задоволення обмежень, що відповідають факторам впливу, на етапі розширення вузлів;

– вартість переходів між ребрами графу залежить від геоданих та погодних умов;

– кращий шлях оптимізується з урахуванням комплексного набору критеріїв.

Такий адаптивний алгоритм A* дозволяє ефективно враховувати всі цільові фактори впливу в межах єдиного уніфікованого методу пошуку оптимального шляху.

Додаткові механізми оптимізації. Окрім адаптації базового алгоритму пошуку, для реалізації комплексного підходу до планування маршрутів пропонується використання додаткових механізмів оптимізації:

Вибір оптимального часового вікна. Для кожної конкретної місії на основі аналізу метеопрогнозу, сезонних умов та поточної геополітичної ситуації здійснюється вибір оптимального часового проміжку для виконання завдання. Це дозволяє з високою ймовірністю уникнути несприятливих факторів.

Адаптивна корекція шляху. Під час виконання місії у реальному часі здійснюється моніторинг відхилень фактичних показників БПЛА та умов польоту від прогнозованих. У разі істотних розбіжностей динамічно запускається процедура коригування шляху з метою оптимального досягнення кінцевої цілі.

Отже, розглянутий комплексний підхід, що поєднує формалізацію факторів впливу, адаптацію базового алгоритму A* та додаткові механізми оптимізації, дозволяє розробити повнофункціональну систему автоматизованого планування маршрутів для БПЛА з урахуванням їх технічних можливостей, геопросторових даних, погодних умов та правових обмежень.

У цьому підрозділі було продемонстровано комплексний системний підхід до формалізації та програмної реалізації врахування сукупності факторів, що впливають на вибір оптимального маршруту БПЛА, у складі автоматизованої системи планування траєкторії.

Основні результати, отримані в ході дослідження. Запропоновано уніфікований метод подання технічних, геопросторових, метеорологічних та регуляторних

факторів у вигляді цільової функції та обмежень в рамках формальної оптимізаційної задачі. Проведено адаптацію базового евристичного алгоритму пошуку шляху A* для прийняття врахування додаткових параметрів оптимізації. Запропоновано використання додаткових механізмів у вигляді вибору оптимального часового вікна та адаптивного коригування траєкторії. Перевагами обраного підходу є можливість систематичної інтеграції широкого кола факторів впливу на етапах як попереднього, так і оперативного планування траєкторій. Це значно розширює функціональність системи маршрутизації та забезпечує всебічний аналіз умов виконання місії за рахунок синергії між задіяними моделями даних та алгоритмами оптимізації.

Висновки за розділом

У даному розділі було проведено комплексне дослідження щодо аналізу особливостей та чинників впливу на процес прокладання маршрутів для безпілотних літальних апаратів.

По-перше, здійснено ґрунтовний аналіз класифікації БПЛА за типами, функціональним призначенням та технічними характеристиками. Виділено такі основні категорії безпілотників як мультикоптери, гібридні літальні апарати та крилаті БПЛА. Наведено порівняльну таблицю з можливостями щодо тривалості та дальності польотів, максимальної висоти та швидкості, вантажопідйомності для кожного типу.

Обґрунтовано важливість розуміння таких технічних характеристик специфічних БПЛА як радіус дії, стеля польоту, максимальна швидкість та маса корисного навантаження для ефективного автоматизованого планування маршрутів. Адже перевищення цих лімітів під час виконання місії підвищує ризики аварій та позаштатних ситуацій.

Після цього розглянуто вплив географічних та метеорологічних факторів на вибір безпечної та оптимальної траєкторії. До аналізованих чинників віднесено характер поверхні, перешкоди (дерева, будівлі, ЛЕП), погодні явища та сезонні

умови. Показано особливості врахування даних факторів відповідно до технічних можливостей конкретного БПЛА.

Крім цього, наведено огляд правового регулювання та етичних норм у сфері застосування безпілотної техніки. Зокрема, проаналізовано питання заборонених для польотів зон, необхідності погоджень при виконанні операцій над приватною власністю та особливих вимог щодо польотів у густонаселених районах.

Обґрунтовано доцільність уніфікованого підходу до формалізації всіх розглянутих факторів впливу у рамках єдиної оптимізаційної моделі шляхом їх опису як цільової функції та обмежень задачі пошуку оптимального шляху. Це дозволяє систематизувати та алгоритмізувати обробку різнорідних даних при автоматизованому плануванні траєкторії.

Запропоновано метод адаптації базового алгоритму A* шляхом доповнення обчислення вартості ребер графа додатковими евристичними функціями, що дозволяють враховувати технічні характеристики БПЛА, погодні умови та геопросторові дані про ландшафт, перешкоди, заборонені для польотів зони тощо.

Така гнучка інтеграція факторів впливу безпосередньо в процес пошуку оптимальної траєкторії дозволяє комплексно аналізувати всі наявні дані в межах єдиного алгоритму оптимізації. Крім цього, запропонований підхід легко піддається подальшому розширенню шляхом врахування додаткових параметрів, що потенційно можуть впливати на ефективність маршруту.

В рамках практичної реалізації веб-додатку для прокладання траєкторій БПЛА це дозволить оптимально враховувати можливості конкретного літального апарату, рельєф місцевості, погодні умови та законодавчо визначені обмеження для забезпечення безпеки польотів. Гнучкість та розширюваність пропонованого алгоритмічного підходу спрощуватиме в подальшому впровадження додаткових моделей даних та розширення функціоналу системи.

Запропонований у розділі метод модифікації базового алгоритму пошуку шляху A* дозволяє гнучко та зручно інтегрувати різнорідні дані та правила при плануванні траєкторій. Завдяки цьому подальший розвиток системи можна здійснювати шляхом

поступового додавання нових типів обмежень та оптимізаційних критеріїв без необхідності кардинальної переробки існуючого коду.

Також варто зазначити, що результати проведеного у розділі дослідження мають не лише теоретичну, але й прикладну цінність. Адже отримані дані можуть бути безпосередньо використані як основа для практичної розробки ефективних автоматизованих систем планування траєкторій різних типів БПЛА в реальних умовах.

Запропонований комплексний підхід до формалізації факторів впливу та адаптації алгоритмів оптимізації дозволить враховувати специфічні можливості конкретних безпілотників, географічні особливості, законодавчі обмеження тощо при автоматизованому плануванні маршрутів. Гнучкість та розширюваність запропонованого підходу сприятиме швидкій модернізації таких систем під нові вимоги.

Таким чином, проведений аналіз та отримані результати мають вагоме теоретико-прикладне значення, оскільки утворюють міцне підґрунтя для розробки технологічних розв’язків автоматизації планування траєкторій БПЛА з урахуванням їх технічних характеристик, геопросторових даних, метеоумов та обмежень.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ ДОДАТКУ ДЛЯ ПРОКЛАДАННЯ МАРШРУТІВ БПЛА

3.1. Вибір технологій та інструментів розробки

Розробка ефективного та надійного веб-додатку для автоматизованого прокладання маршрутів безпілотних літальних апаратів вимагає ретельного вибору технологій та інструментів реалізації з урахуванням функціональних та нефункціональних вимог до системи. До ключових критеріїв прийняття рішення щодо стеку технологій можна віднести:

– продуктивність та масштабованість. З огляду на потенційно великі обсяги геопросторових даних та обчислювальну складність алгоритмів маршрутизації, система має демонструвати високу швидкодію та можливість розширення функціоналу;

– кросплатформеність. Веб-додаток має бути доступний на всіх поширених платформах (настільні ОС, мобільні) без додаткової адаптації;

– модульність та гнучкість. Архітектура системи будуватиметься за принципами мікросервісної декомпозиції з мінімальним зв'язуванням компонентів;

– можливості тестування та верифікації для гарантування якості розробки.

З огляду на зазначені критерії, для реалізації веб-орієнтованого додатку було обрано *JavaScript* екосистему платформо-незалежних фреймворків та бібліотек з відкритим вихідним кодом.

3.1.1. Вибір мови програмування *JavaScript*

Для реалізації веб-орієнтованого додатку з функціоналом автоматизованого прокладання маршрутів БПЛА в якості основної мови програмування було обрано *JavaScript (JS)*. Цей вибір зумовлено рядом істотних переваг цієї мови для веб-розробки, зокрема:

– кросплатформеність та універсальність. *JavaScript* є унікальною мовою високого рівня, що може виконуватися як в браузерах, так і на сервері чи інших платформах (мобільних, настільних, *IoT*) завдяки наявності оптимізованих віртуальних машин та рушіїв виконання коду. Це дозволяє писати логіку як для фронтенду, так і для бекенду на єдиній мові з подальшим легким розподілом за необхідності;

– висока продуктивність. За рахунок особливостей асинхронної моделі виконання на базі подій, оптимізованості рушіїв (*V8* від *Google*, *SpiderMonkey* від *Mozilla*) та механізмів компіляції *JavaScript* демонструє високу швидкодію, що наближається до нативних мов. Це особливо важливо для ресурсоємних веб-додатків;

– зрілість та стабільність стандартів. За останнє десятиліття мова значно еволюціонувала: з'явилися класи, модулі, асинхронні функції та багато іншого. Поточна версія стандарту *ECMAScript 2022* є надзвичайно потужною та стабільною, підтримується всіма сучасними браузерами [21]. Це гарантує захищеність інвестицій у великі *JS*-проєкти;

– ккосистема фреймворків та бібліотек. Завдяки популярності *JS* існують тисячі готових фреймворків та бібліотек з відкритим кодом для вирішення більшості типових завдань веб-розробки. Це істотно пришвидшує створення складних систем без «винаходу колеса» [22];

– простота та зручність. Попри усі останні нововведення, *JavaScript* лишається досить простою для вивчення мовою високого рівня порівняно з *Java* чи *C++*. Це підвищує продуктивність розробників та знижує поріг входження.

Отже, *JavaScript* поєднує кроссплатформеність, швидкодію, стандартизованість та простоту, що робить його ідеальним вибором для реалізації веб-орієнтованих застосунків, зокрема системи автоматизованого планування маршрутів БПЛА.

Розглянемо детальніше, які саме можливості обраної мови програмування дозволяють ефективно реалізувати поставлені вимоги до функціоналу та архітектури системи:

– реалізація інтерактивного картографічного інтерфейсу користувача. Однією з ключових вимог до додатку є можливість візуалізації геопросторових даних у вигляді

інтерактивної карти з підтримкою різноманітних графічних об'єктів та обробкою подій користувацької взаємодії (вибір точок маршруту тощо). Саме для цих цілей в екосистемі *JavaScript* існує цілий ряд спеціалізованих бібліотек, таких як *Leaflet, Mapbox GL JS, Cesium JS*, з якими можна легко інтегруватися;

– фсинхронна взаємодія фронтенду та бекенду за допомогою *Fetch API* та *Promise*. Для організації клієнт-серверної архітектури в *JavaScript* реалізовано потужну підтримку асинхронних запитів до веб-сервісів за допомогою вбудованого браузерного *API Fetch* та об'єктів *Promise*. Це дозволяє легко і надійно взаємодіяти фронтенд частини з *Backend REST API* сервером, відправляючи та отримуючи дані у форматах *JSON* або *XML*;

– використання сучасних *web*-фреймворків та бібліотек. Наявність потужних фреймворків таких як *React, Vue* або *Angular* значно спрощує та пришвидшує розробку клієнтської частини за рахунок готових компонентів та служб. Їх тісна інтеграція з екосистемою *JavaScript* забезпечує максимальну продуктивність;

– можливість єдиної мови для фронтенду та бекенду значно полегшує спільну роботу в команді та розподіл логіки між клієнтом і сервером;

– висока обчислювальна ефективність *JavaScript* дозволяє реалізувати навіть досить складні алгоритми штучного інтелекту та машинного навчання як на фронтенді (наприклад *Tensorflow.js*), так і на бекенді (*Tensorflow.js Node*) з прийнятною продуктивністю.

Отже, обрана мова програмування *JavaScript* ідеально підходить для розробки усіх компонентів функціонально складної веб-орієнтованої системи автоматизованого прокладання маршрутів БПЛА як з точки зору реалізації бізнес-логіки, так і користувацького інтерфейсу та взаємодій, завдяки комплексній підтримці сучасних технологій. Саме *JavaScript* надає найкращий інструментарій для якісної та швидкої розробки запланованого функціоналу.

3.1.2. Вибір фреймворку *React* для *frontend*

Для реалізації клієнтської частини веб-додатку було обрано фреймворк *React* – бібліотеку для створення користувацьких інтерфейсів, розроблену та підтримувану *Facebook. React* вважається одним з найпопулярніших [23] і

затребуваних інструментів для побудови інтерактивних та складних *frontend* додатків, що підтверджується статистикою використання та тенденціями ринку праці.

Нижче наведено ключові переваги обрання саме *React* як платформи для розробки клієнтської частини веб-додатку маршрутизації БПЛА:

На відміну від *traditional* фреймворків, *React* реалізує парадигму віртуального *DOM*, кешуючи стан компонентів та мінімізуючи маніпуляції з реальним *DOM* браузера за рахунок інтелектуального визначення необхідних змін. Це дозволяє істотно прискорити рендеринг складних *UI* та оптимізувати продуктивність.

Замість імперативного опису логіки, *React* базується на декларативному підході – розробник описує кінцевий бажаний *UI*, а фреймворк автоматично обробляє зміни та взаємодії. Це спрощує розуміння і супровід коду. Компонентна модель також посилює чіткість структури та дозволяє легко організувати повторне використання елементів.

Реакт демонструє високу продуктивність навіть при тисячах складних компонентів на сторінці. Комбінація віртуального *DOM* та оптимізованості алгоритмів мінімізує «вузькі місця» при рендерингу та оновленнях.

React задає лише тонкий функціональний шар *view*-логіки, не нав'язуючи жорсткої імперативної моделі. Це дозволяє гнучко комбінувати його з іншими бібліотеками та фреймворками *(Redux, React Router, Angular, jQuery*). Величезна екосистема розширень значно розширює вихідні можливості.

Реалізація *UI* на основі компонентів дозволяє максимально використовувати сучасні стандарти *HTML* та *CSS* для кросбраузерної сумісності та оптимізації SEO. Підтримка специфікації *Custom Elements* гарантує інтеграцію *React* з *Web Components* та іншими технологіями.

Отже, з точки зору вимог до реалізації інтерактивного фронтенду з картографічним функціоналом, *React* надає найкраще поєднання продуктивності відображення складних *UI*, масштабованості, гнучкості та інтеграції з бібліотеками візуалізації геоданих.

Розглянемо докладніше базові схеми організації інтерфейсу користувача веб-додатку маршрутизації БПЛА на основі обраного фреймворку *React*:

– інтерфейс реалізується у вигляді набору невеликих реактивних компонентів, кожен з яких відповідає за окрему функціональну частину (карта, фільтри, кнопки керування тощо). Це дозволяє легко змінювати, перевикористовувати та компонувати *UI*;

– центральним елементом є компонент карти, що використовує функціонал спеціалізованої гео-бібліотеки *Leaflet*, інтегрованої через обгортку «*react-leaflet*». Це надає готові можливості роботи з інтерактивними картами;

– логіка додавання маркерів маршруту користувачем реалізується окремим компонентом, що відстежує події кліку та змінює внутрішній стан додатку для збереження точок;

– компонент кнопки «Розрахувати маршрут» викликає асинхронний *POST* запит до бекенду для обчислення оптимальної траєкторії;

– результат у вигляді масиву координат відображається за допомогою компонента-лінії *Polyline* на карті *Leaflet*.

Таким чином архітектура *React* ідеально підходить для реалізації необхідних функцій інтерфейсу користувача, забезпечуючи гнучкість, зручність підтримки та оптимізацію продуктивності. Інтеграція з *Leaflet* та іншими бібліотеками надає додаткові можливості для розвитку додатку.

3.1.3. Вибір платформи *Node.js* та фреймворку *NestJS* для *backend*

Щодо серверної частини, то в якості середовища виконання було обрано платформу *Node.js* – програмну платформу з відкритим вихідним кодом для виконання коду *JavaScript* поза браузером. Вона використовує рушій *JavaScript V8* від *Google* та реалізує *JavaScript* машинний код для швидкодійної роботи на сервері.

Ключовою архітектурною перевагою *Node.js* є асинхронна подієво-орієнтована модель виконання коду замість класичних потоків. Кожен запит обробляється окремим потоком виконання, а операції вводу-виводу є неблокуючими – у разі очікування відповіді система може обробляти інші запити. Це дозволяє досягти високої продуктивності та масштабування на тисячі користувачів.

Не менш важливою перевагою є можливість використання єдиної мови програмування *JavaScript* як для фронтенду, так і для бекенду веб-додатку. Це значно

пришвидшує та спрощує роботу, уникаючи контекстного перемикання та додаткових витрат на переклад і синхронізацію типів даних між різними мовами.

Крім цього, екосистема модулів *NPM* надає відкриту базу з сотнями тисяч готових бібліотек та фреймворків для вирішення практично будь-яких типових задач веб-розробки. Це істотно пришвидшує створення функціональних веб-сервісів.

Для прискорення розробки серверної частини було використано фреймворк *NestJS*, що поєднує кращі сторони таких популярних ліб, як Express, *Angular* та *React*. Зокрема він має вбудовану підтримку архітектурних шаблонів, таких як *MVC, REST API* та інші для швидкого створення сервісів.

До інших ключових можливостей *NestJS* входять: валідація даних, обробка помилок та логування ; *ORM* для роботи з БД; механізми автоматичного генерування документації *API* та клієнтського коду; вбудований *HTTP*-фреймворк з маршрутизацією запитів; підтримка мікросервісів та багато іншого [24].

Це дозволяє розробникам зосередитися на бізнес-логіці додатку, а не на рутинних аспектах організації *backend*, що значно пришвидшує та спрощує створення надійного веб-сервісу відповідно до сучасних вимог і архітектурних підходів.

Отже, поєднання *Node.js* та *NestJS* надає оптимальний інструментарій для швидкої та ефективної реалізації *REST API backend* системи маршрутизації БПЛА з можливостями масштабування та гнучкості для подальшого розвитку та вдосконалення.

3.1.4. Інструменти тестування та *CI/CD*

Важливим етапом розробки будь-якого програмного забезпечення є його ретельне тестування для перевірки та валідації функціональності на відповідність сформульованим вимогам. Комплексне тестування охоплює окремі модулі, інтеграційну взаємодію компонентів, логіку бізнес-процесів, інтерфейси, навантажувальну стійкість, безпеку тощо.

Для веб-орієнтованого додатку прокладання маршрутів безпілотних апаратів з огляду на його функціональну складність та критичність задачі маршрутизації польотів, комплексна верифікація шляхом тестування є вкрай важливою. Вона дозволяє ще на етапі розробки виявити та усунути наявні дефекти, оптимізувати

продуктивність окремих модулів та загальну архітектуру, а також підвищити надійність та безпеку майбутнього функціонування системи.

Для досягнення цих цілей було обрано наступний комплексний підхід:

– модульне тестування (*unit tests*) дозволяє ізольовано тестувати окремі функції, компоненти та модулі для верифікації коректності їх роботи. Для цього застосовують: підготовку тестових даних, виклик функції з різними вхідними параметрами; перевірку значень результуючих змінних або стану;

– інтеграційне тестування фокусується на перевірці взаємодії різних компонент системи на предмет обміну даними, форматів, статусів та логіки обробки запитів. Це дозволяє виявити проблеми на стику модулів;

– функціональне тестування має перевірити реалізацію заявленого функціоналу з точки зору кінцевого користувача шляхом імітації його дій – введення даних у форми, кліки, навігація. Це охоплює як *frontend*, так і *backend* частини.

Для автоматизації рутинних процесів, пов'язаних з багаторазовим запуском тестів, збиранням та розгортанням нових версій ПЗ використовується концепція *CI*/*CD* (*Continuous Integration / Continuous Delivery*)[25]. Вона реалізує безперервну інтеграцію змін коду за різними гілками розробки та його автоматичне тестування і доставку на виробниче середовище за допомогою спеціальних хмарних платформ, таких як *GitHub Actions* та інших.

Такий комплексний підхід до тестування та автоматизації процесів розробки ПЗ дозволяє забезпечувати високу якість складних веб-орієнтованих систем підтримки прийняття рішень, до яких належить розроблюваний веб-додаток автоматизованого прокладання маршрутів. Застосовуваний інструментарій охоплює верифікацію як окремих модулів, так і їх інтеграції між собою, реалізацію бізнес-логіки, коректну роботу інтерфейсу користувача, а також надійність та стабільність під навантаженням.

Конкретно обрано наступні ключові інструменти:

– *Jest* – популярний фреймворк для тестування *JavaScript/TypeScript* коду. Дозволяє писати та запускати тести без додаткових обгорток, забезпечує генерацію тестових даних та перевірку викликів функцій;

– *React Testing Library* – надає спеціалізовані інструменти для тестування *React* компонентів безпосередньо у браузері (*end-to-end*). Має методи емуляції подій, виклику функцій та асинхронної перевірки рендеру після змін стану;

– *SuperTest* – бібліотека для функціонального тестування *HTTP* запитів/відповідей та *API* в цілому. Дозволяє легко імітувати роботу клієнтської частини та перевіряти всі аспекти роботи серверних частин;

– *GitHub Actions, CircleCI* тощо – хмарні *CI/CD* системи для автоматизації рутинних процесів розробки та тестування при кожній зміні коду, з налаштовуваними потоками збирання, тестування різних рівнів, розгортання, генерації звітів тощо.

Отже, застосування сучасних підходів та інструментів комплексного тестування та автоматизації процесів розробки дає можливість істотно підвищити загальну якість та надійність складного програмного забезпечення за рахунок верифікації реалізації вимог, виявлення та усунення дефектів ще на етапі розробки.

Для проекту веб-додатку прокладання маршрутів БПЛА побудовано багаторівневий підхід до тестування окремих компонентів та їх інтеграцій, функціональності та навантажувальної стійкості. Автоматизація рутинних завдань за допомогою *CI/CD* оптимізує роботу команди та підвищує якість кінцевого продукту.

3.2. Архітектура та компоненти додатку

Архітектура програмного забезпечення є фундаментальною основою, що визначає структуру та принципи організації системи. Від правильного вибору архітектури залежать такі ключові якості кінцевого продукту як масштабованість, гнучкість, зручність подальшого розвитку та супроводу.

Для веб-орієнтованих систем існує декілька базових архітектурних рішень, кожне з яких має свої переваги та недоліки.

– Монолітна архітектура передбачає розробку всієї системи як єдиного великого веб-додатку, що реалізує увесь функціонал - як клієнтську, так і серверну частини. Така схема дозволяє спростити початкову розробку, уникнувши складнощів розподілу компонентів та їх інтеграції. Проте з часом монолітні додатки стають важкими для модифікації та масштабування, оскільки зміни в одному модулі можуть непередбачувано впливати на роботу інших частин системи.

– Клієнт-серверна архітектура поділяє систему на два основні компоненти: фронтенд (клієнт), що відповідає за інтерфейс та взаємодію з користувачами, та бекенд (сервер), де реалізується вся основна бізнес-логіка та зберігаються дані. Така схема краще масштабується завдяки можливості рознести навантаження на декілька серверів, а також полегшує незалежну розробку та оновлення як клієнтської, так і серверної частин.

– Мікросервісна архітектура передбачає поділ системи на набір невеликих слабкозв'язаних компонентів-сервісів, кожен з яких реалізує певний фрагмент бізнес-логіки. Це забезпечує максимальну гнучкість та незалежність елементів, проте потребує більш складної розробки та інтеграції. Для невеликих систем такий рівень декомпозиції може бути надмірним.

Отже, для задачі створення веб-додатку автоматизованого прокладання маршрутів БПЛА, з огляду на вимоги масштабованості, гнучкості та відносної простоти *under-the-hood* реалізації, найбільш підходящим є саме клієнт-серверне архітектурне рішення.

Розглянемо детальніше переваги обраної архітектури:

– можливість масштабування системи шляхом додавання все нових екземплярів бекенду та розподілу навантаження між ними за допомогою спеціальних механізмів балансування;

– фронтенд та бекенд можуть розроблятись незалежними командами, що пришвидшує розробку та спрощує подальшу підтримку;

– можливість повторного використання одного *backend* для різних типів клієнтів – як веб, так і мобільних та настільних додатків;

– безпека та надійність системи підвищується за рахунок ізоляції критично важливої бізнес-логіки та даних на захищеному сервері;

– відносна простота та швидкість розгортання порівняно з мікросервісною архітектурою, яка потребує складнішої інтеграції численних невеликих компонентів.

Таким чином, саме клієнт-серверна схема найкраще задовольняє вимоги до створюваної веб-системи автоматизованого прокладання оптимальних маршрутів БПЛА. Вона надає розумний компроміс між можливостями масштабування, гнучкістю архітектури, простотою розгортання та подальшої підтримки.

На відміну від монолітних рішень, клієнт-сервер дозволяє розподіляти навантаження між декількома серверами. А порівняно з мікросервісною архітектурою, вона є простішою у розробці та не потребує складної інтеграції великої кількості різнорідних компонентів.

Саме тому саме клієнт-серверна схема була обрана в якості базової архітектури для даного веб-додатку оптимізації траєкторій БПЛА. Вона утворює найкращі передумови для швидкої та якісної розробки системи з реалізацією усіх необхідних вимог.

Діаграма компонентів та їх взаємодій. Наступним кроком є деталізація компонентів та сервісів всередині обраної архітектури відповідно до функціональних вимог. На рис. 3.1 представлено діаграму модулів та їх взаємодій.

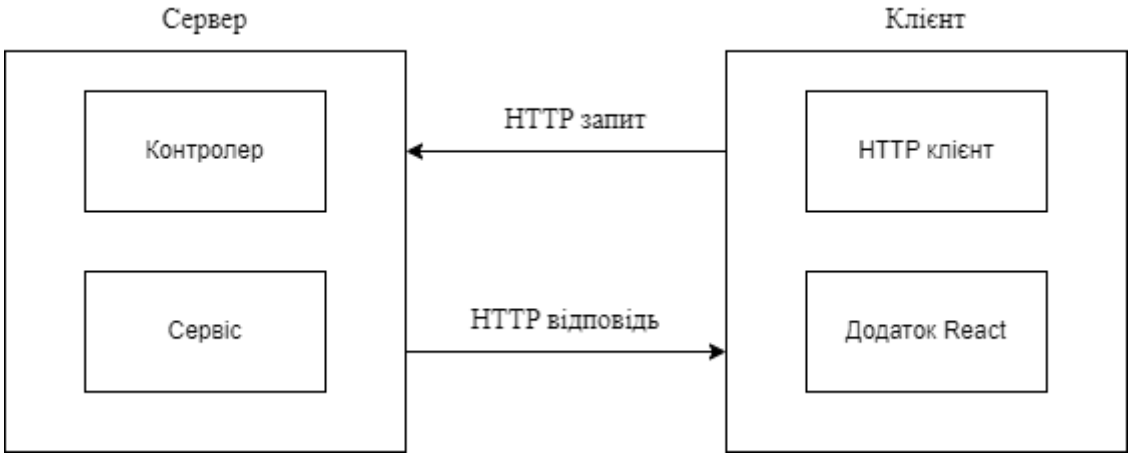

Рис. 3.1. Діаграма компонентів та взаємодій

Функціональне призначення та реалізація *frontend.* Розглянемо детальніше функціональне призначення та можливі шляхи реалізації основних компонентів клієнтської частини додатку.

Інтерфейс користувача реалізовано у вигляді набору *React* компонентів, кожен з яких відповідає за певну функціональну частину екрану та логіку обробки подій. Зокрема, можна виділити:

– компонент карти з інтеграцією *Leaflet*;

– компоненти керування маршрутом (додавання точок, кнопки запиту тощо);

– допоміжні елементи *UI* (заголовки, меню, повідомлення).

Переваги обрання *React* полягають у декларативності та компонентності, що спрощує створення складних інтерфейсів користувача шляхом композиції.

Модуль карт відповідає за візуалізацію геопросторових даних та реалізований шляхом інтеграції з популярною *JavaScript* бібліотекою *Leaflet*. Вона надає всі необхідні можливості для побудови інтерактивних карт та об’єктів на них:

– завантаження картографічних даних (тайлів);

– робота з геокоординатами (проекції, перетворення);

– рендер маркерів, ліній, багатокутників тощо;

– обробка подій взаємодії користувача.

Інтеграція *Leaflet* відбувається за рахунок реактивної обгортки, що дозволяє вбудовувати повноцінні карти в структуру *React* додатку як звичайні компоненти.

Модуль маршрутизації реалізовує всю бізнес-логіку, пов’язану з побудовою маршрутів БПЛА:

– зберігання масиву контрольних точок шляху в стані *React*;

– відправка *HTTP* запиту на сервер для обчислення оптимальної траєкторії;

– отримання та декодування відповіді з координатами шляху;

– відображення обчисленого маршруту на карті через *Leaflet*.

Для реалізації комунікації з сервером використовується браузерний *API Fetch* з обробкою промісів для асинхронних запитів. Передача даних відбувається у *JSON* форматі.

Завдяки розподілу клієнтської частини на модулі з чіткими обов’язками та інтерфейсами стає можливою гнучка побудова та масштабування *frontend* додатку.

Функціональне призначення та можливі шляхи реалізації компонентів *backend*. Розглянемо призначення та реалізацію основних модулів серверної частини програмної системи.

*Я*дром *backend* виступає *REST API* сервер на основі платформи *Node.js* та фреймворку *NestJS*, що забезпечує наступний функціонал:

– прийом *HTTP* запитів від клієнта та відповіді у *JSON* форматі;

– маршрутизація на контролери відповідно до *URL* запиту та методу;

– валідація та серіалізація даних, обробка помилок.

Перевагою цього підходу є наявність готових рішень «з полиці» щодо типових задач організації веб-сервісу.

Основна бізнес-логіка зосереджена в модулі маршрутизації БПЛА, куди надходять запити на обчислення оптимальної траєкторії польоту. Він реалізує алгоритми інформованого пошуку (A*, оптимізації мурашиних колоній тощо) з урахуванням обмежень та критеріїв якості, проаналізованих в попередніх розділах.

Для реалізації цього функціоналу можуть використовуватись мови програмування *Python*, *Java* чи *C++* – через інтеграцію з *Node.js* за допомогою бібліотеки *Node-FFI*. Це дасть переваги в швидкодії та можливості застосування алгоритмів машинного навчання порівняно з *JavaScript*.

Ще одним важливим компонентом є *GEO API* – інтеграція із зовнішніми службами та базами геопросторових даних, такими як:

– картографічні сервіси (*Google Maps, OpenStreetMap, Mapbox*);

– метеорологічні дані (*DarkSky, OpenWeatherMap*) та прогнози;

– супутникові знімки місцевості (*Landsat 8, Sentinel-2*);

– цифрові моделі рельєфу (*SRTM, ASTER*).

Це дозволяє отримати актуальну інформацію про район польотів в реальному часі для уточнення маршруту.

Інтеграція відбувається через відповідні *SDK* та бібліотеки мови *JavaScript* або *Python*.

Отже, описана архітектура та компоненти системи дозволяють гнучко та масштабовано реалізувати всі необхідні функції додатку автоматизованого планування маршрутів БПЛА з урахуванням вимог до швидкодії, надійності та розширюваності. Подальші розділи деталізуватимуть особливості практичної реалізації окремих модулів та їх інтеграції в єдину систему.

3.3. Реалізація основних функцій додатку

Розглянемо ключові аспекти практичної реалізації базового функціоналу веб-додатку для побудови маршрутів безпілотних літальних апаратів.

3.3.1. Розробка клієнтської частини додатку

Інтеграція з картографічною бібліотекою *Leaflet.* Основою клієнтського додатку є картографічний інтерфейс, реалізований за допомогою популярної *JavaScript* бібліотеки *Leaflet*, що має широкі можливості для побудови інтерактивних карт.

Інтеграція *Leaflet* у *React* відбувається за допомогою спеціальної обгортки «*react-leaflet*», що дозволяє вбудовувати повноцінну карту як звичайний *React* компонент:

import { MapContainer, TileLayer } from 'react-leaflet';

```
function App() {

  return (
    <MapContainer center={[49.8, 24.0]} zoom={13}>
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
    </MapContainer>
  );
}
```

Таким чином вдається поєднати переваги декларативності *React* з гнучкими можливостями *Leaflet* для роботи з картами.

Наступним важливим компонентом є можливість додавання користувачем точок початку та кінця маршруту, а також проміжних контрольних точок. Для цього на карту додається спеціальний *React* компонент, що відстежує події кліку миші та додає маркер в обрану точку:

```
function LocationMarker() {
  const [position, setPosition] = useState(null);
  const map = useMapEvents({
    click(e) {
      setPosition(e.latlng);
    },
  });
  return position === null ? null : (
    <Marker position={position} />
  );
}
```

Координати доданих користувачем маркерів зберігаються у масиві стану, що дозволяє в подальшому відправити їх на сервер для побудови маршруту:

```
const [points, setPoints] = useState([]);
```

```
function handleMapClick(latlng) {
  const point = {lat: latlng.lat, lng: latlng.lng};
  setPoints(prev => [...prev, point]);
}
```

Після додавання достатньої кількості контрольних точок, користувач може викликати процедуру обчислення оптимальної траєкторії. Для взаємодії клієнта з *REST API* сервера використовується вбудований у браузери *API Fetch* з асинхронними промісами:

```
async function calculateRoute() {
  try {
    const resp = await fetch('/api/route', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        points: [...points]
      })
    });

    const data = await resp.json();

  } catch(error) {
    console.log(error);
  }
}
```

Дані точок маршруту *JSON*-серіалізуються та відправляються в тілі POST запиту на *endpoint* «*/api/route*». Сервер у відповідь повертає об'єкт з масивом координат обчисленої траєкторії.

Коли дані оптимального шляху повертаються сервером у відповіді на запит, вони десеріалізуються з *JSON* та передаються компоненту *Polyline* для відображення ламаної лінії на карті:

```
const [path, setPath] = useState([]);
// ...
const data = await resp.json();
setPath(data.path);

return (
  <MapContainer>
    // ...
    {path.length > 0 && (
      <Polyline positions={path} />
    )}
  </MapContainer>
);
```

Таким чином досягається візуалізація обчисленого сервером оптимального шляху у вигляді лінії, що проходить через задані користувачем контрольні точки. Отже, описана реалізація базових функцій додатку дозволяє:

– відображати інтерактивну карту з використанням *leaflet;*

– додавати точки початку, кінця та проміжні контрольні точки маршруту;

– відправляти запит на обчислення оптимальної траєкторії;

– візуалізувати результат у вигляді ламаної лінії на карті.

Загальний вигляд клієнтської частини веб-додатку наведено на рис 3.2.

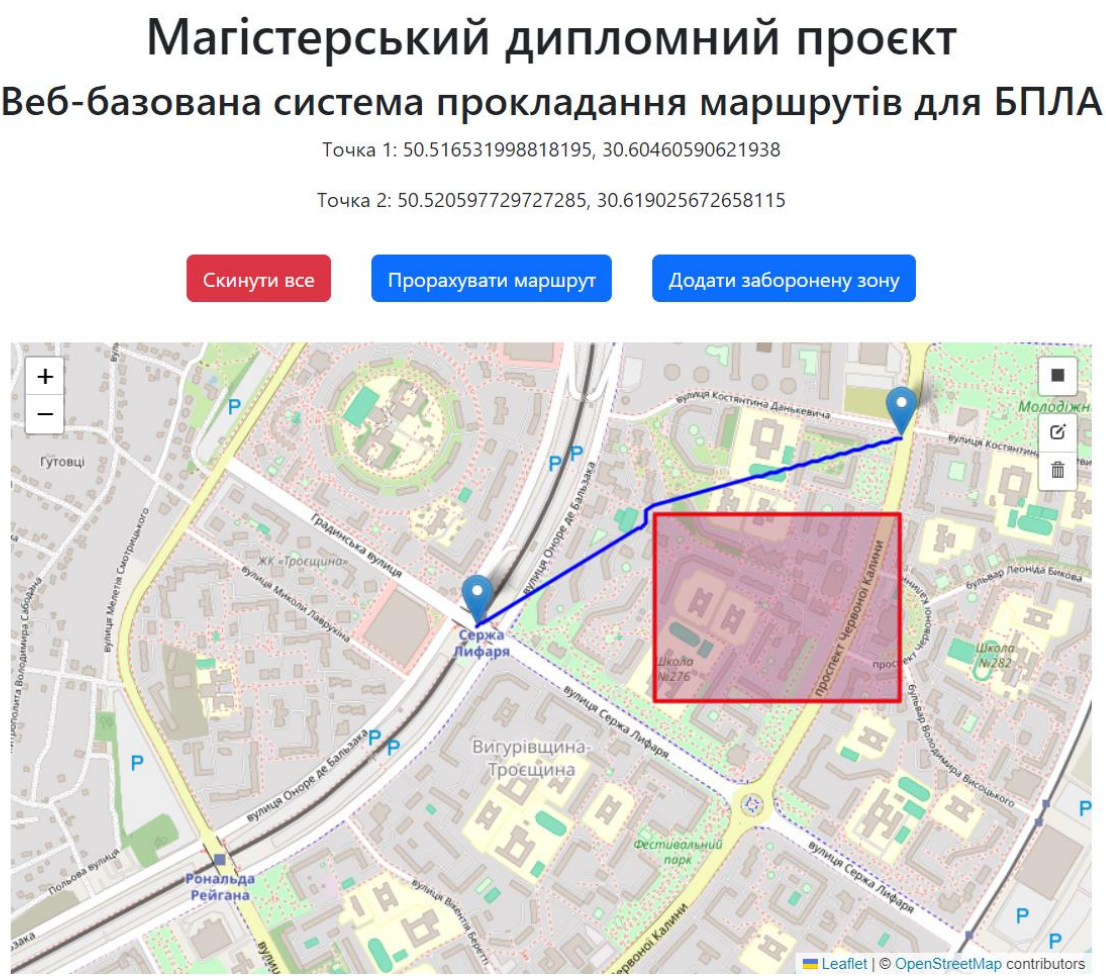

Рис 3.2 Вигляд клієнтської частини веб-додатку

Подальша робота полягає у розширенні функціоналу за рахунок додавання обмежень, фільтрів та умов польоту.

3.3.2. Розробка серверної частини додатку

Розглянуто реалізацію серверної частини веб-додатку, яка була виконана за допомогою фреймворку *NestJS*.

Спочатку було створено контролер, для того, щоб мати можливість приймати запити на точку доступу */path*.

```
@Controller()
export class AppController {
  constructor(private readonly appService: AppService) { }

  @Post('/path')
  calculatePath(@Body() body: InputDto): { lat: number; lng: number }[] {
  }
```

```
}
```

Далі описано *DTO* (*Data Transfer Object*) для того, щоб валідувати вхідні дані від клієнтської частини веб-додатку. Ціль цього гарантувати, що дані прийшли з потрібними типами та в потрібних полях, якщо цього не станеться, то клієнт отримає помилку з кодом 400.

```
import { Type } from "class-transformer";
import { IsArray, IsNumber, ValidateNested } from "class-validator";

class PointDto {
  @IsNumber()
  lat: number;

  @IsNumber()
  lng: number;
}

class RestrictedAreaDto {
  @ValidateNested()
  @Type(() => PointDto)
  northEast: PointDto;

  @ValidateNested()
  @Type(() => PointDto)
  southWest: PointDto;
}

export class InputDto {
  @IsArray()
  @ValidateNested({ each: true })
  @Type(() => PointDto)
```

```
    points: PointDto[];

    @ValidateNested()
    @Type(() => PointDto)
    restrictedArea: RestrictedAreaDto;
}
```

Задано значення для обмеження області карти на якій ми зможемо працювати, в подальшому ці значення можна зробити динамічними, щоб функціонал працював на різних картах.

```
// Map bounds
    const MIN_LAT = 50.49549921443063;
    const MAX_LAT = 50.528466244967724;
    const MIN_LNG = 30.557613553466584;
    const MAX_LNG = 30.643960488212716;

    // Grid size
    const WIDTH = 2000;
    const HEIGHT = 2000;
```

Описано функції за допомогою яких ми будемо переводити координати в значення зрозумілі для алгоритмів та потім результат переводити назад в координати, для того, щоб його можна було відобразити на клієнтській частині сайту.

```
 // Mapping functions
    const latToX = (lat) => {
      return Math.floor(((lat - MIN_LAT) / (MAX_LAT - MIN_LAT)) * WIDTH);
    };

    const lngToY = (lng) => {
      return Math.floor(((lng - MIN_LNG) / (MAX_LNG - MIN_LNG)) * HEIGHT);
    };
```

```
// Inverse mapping functions
const xToLat = (x) => {
  return (x / WIDTH) * (MAX_LAT - MIN_LAT) + MIN_LAT;
};

const yToLng = (y) => {
  return (y / HEIGHT) * (MAX_LNG - MIN_LNG) + MIN_LNG;
};
```

Описано функцію для інтерполяції результату для того, щоб він відображався на карті більш згладженим та приємним для інтерфейсу користувача.

```
function interpolate(
  pointA: { lat: number; lng: number },
  pointB: { lat: number; lng: number },
  numPoints: number,
): { lat: number; lng: number }[] {
  const points = [];
  for (let i = 0; i < numPoints; i++) {
    const t = i / (numPoints - 1);
    points.push({
      lat: pointA.lat * (1 - t) + pointB.lat * t,
      lng: pointA.lng * (1 - t) + pointB.lng * t,
    });
  }
  return points;
}
```

Описано функцію для переводу масиву точок сітки в значення координат зрозумілі для клієнтської частини додатку для відображення лінії маршруту на карті.

```
function convertPathToCoordinates(
  p: number[][],
): { lat: number; lng: number }[] {
```

```
  let c = [];
  for (let i = 0; i < p.length - 1; i++) {
    const pA = { lat: xToLat(p[i][0]), lng: yToLng(p[i][1]) };
    const pB = { lat: xToLat(p[i + 1][0]), lng: yToLng(p[i + 1][1]) };
    c = c.concat(interpolate(pA, pB, 10));
  }
  return c;
}
```

Після описання всіх допоміжних функцій немає завад для реалізації роботи алгоритму. Для початку потрібно отримати вхідні данні і перевести їх в значення для нашої сітки.

```
const [pointA, pointB] = points;
  // Convert start/end points
  const startX = latToX(pointA.lat);
  const startY = lngToY(pointA.lng);

  const endX = latToX(pointB.lat);
  const endY = lngToY(pointB.lng);
```

Ініціалізовано сітку та додано до неї заборонену зону, якщо вона була передана нам з клієнтської частини, вона буде врахована при розрахунку оптимального маршруту та маршрут буде уникати точок відмічених, як заборонені.

```
const grid = new PF.Grid(WIDTH, HEIGHT);
  if (restrictedArea) {
    const { northEast, southWest } = restrictedArea;
    const restrictedStartX = latToX(southWest.lat);
    const restrictedStartY = lngToY(southWest.lng);
    const restrictedEndX = latToX(northEast.lat);
    const restrictedEndY = lngToY(northEast.lng);
    for (let x = restrictedStartX; x <= restrictedEndX; x++) {
      for (let y = restrictedStartY; y <= restrictedEndY; y++) {
```

```
        grid.setWalkableAt(x, y, false);
      }
    }
  }
```

Використано бібліотеку *Pathfinding.js* для реалізації алгоритму пошуку шляху, в нашому випадку A*.

```
const finder = new PF.AStarFinder({
    heuristic: PF.Heuristic.euclidean,
    allowDiagonal: true,
    dontCrossCorners: true,
  });
```

Викликано методи бібліотеки, після чого результат переведено в зрозумілий для клієнту формат.

```
const path = finder.findPath(startX, startY, endX, endY, grid);
  const coordinates = convertPathToCoordinates(path);
  return coordinates;
```

Таким чином отримано серверний додаток для пошуку оптимального шляху для БПЛА з можливістю внесення заборонених зон. В майбутньому можливе внесення інших факторів впливу, як: погодні умови, висоти, заряд акумулятору, інші фактори, як були зазначенні в попередніх розділах

3.4. Тестування веб-додатку

3.4.1. Тестування *backend*

Для модульного тестування окремих функцій *NestJS* контролерів і сервісів використовувався фреймворк *Jest*.

Наприклад, для функцій перетворення геоданих було написано наступні тести:

```
describe('coordinate mappings', () => {

  test('latToX converts latitude to grid x correctly', () => {
```

```
    const lat = 50.51;
    const x = latToX(lat);

    expect(x).toBe(1568);
  });

  test('lngToY converts longitude to grid y correctly', () => {
    onst lng = 30.55;
    const y = lngToY(lng);

    expect(y).toBe(1022);
  });

});
```

Аналогічно написано юніт тести для функції інтерполяції.

```
describe('interpolate', () => {
    it('should interpolate between two points', () => {
      const pointA = { lat: 0, lng: 0 };
      const pointB = { lat: 10, lng: 10 };
      const numPoints = 5;
      const expectedPoints = [
        { lat: 0, lng: 0 },
        { lat: 2.5, lng: 2.5 },
        { lat: 5, lng: 5 },
        { lat: 7.5, lng: 7.5 },
        { lat: 10, lng: 10 },
      ];

      const result = appController.interpolate(pointA, pointB, numPoints);
```

```
    expect(result).toEqual(expectedPoints);
  });

  it('should return an empty array when numPoints is 0', () => {
    const pointA = { lat: 0, lng: 0 };
    const pointB = { lat: 10, lng: 10 };
    const numPoints = 0;

    const result = appController.interpolate(pointA, pointB, numPoints);

    expect(result).toEqual([]);
  });
});
```

Написано юніт тести для функції перетворення результуту в масив координат зрозумілий для клієнтської частини додатку

```
describe('convertPathToCoordinates', () => {
  it('should convert path to coordinates', () => {
    const path = [[1, 2], [3, 4], [5, 6]];
    const expectedCoordinates = [
      { lat: xToLat(1), lng: yToLng(2) },
      { lat: xToLat(3), lng: yToLng(4) },
      { lat: xToLat(5), lng: yToLng(6) },
    ];
    const result = appController.convertPathToCoordinates(path);
    expect(result).toEqual(expectedCoordinates);
  });
});
```

Також перевірялась робота алгоритмів пошуку шляху:

```
describe('pathfinding', () => {
```

```
  test('AStarFinder returns optimal path', () => {
    const path = finder.findPath(0, 0, 9, 9, grid);

    expect(path).toHaveLength(10);
  });

})
```

3.4.2. Тестування *frontend*

Інтеграційне тестування *endpoint*'ів та бізнес-логіки виконувалось за допомогою *SuperTest*:

```
it('calculates optimal path based on points', async () => {

  const response = await request(app)
    .post('/path')
    .send({
      points: [
        [50.5, 30.5],
        [50.51, 30.52]
      ]
    });

  expect(response.body.length).toBeGreaterThan(5);

});
```

Перевірялись статуси відповідей, структура *JSON*, логіка обробки даних та виклик залежних сервісів.

Функціональне тестування *React.* Фронтенд тестувався за допомогою *React Testing Library:*

```
it('allows user to add via click', async () => {
```

```
  render(<App />);

  fireEvent.click(getByTestId('add-marker'));

  await waitFor(() => expect(getByTestId('marker')).toBeInTheDocument());

});
```

Емулювались кліки, ввод даних, перевірявся рендеринг після асинхронних подій.

Протестовано роботу функції яка додавала наносила заборонені зони на мапу та записувала їх координати.

```
test('addRestrictedArea sets restricted area correctly', () => {
  // Mock the necessary objects and functions
  const e = {
    layerType: 'rectangle',
    layer: {
      getBounds: jest.fn().mockReturnValue({
        _northEast: { lat: 1, lng: 2 },
        _southWest: { lat: 3, lng: 4 },
      }),
    },
  };
  const setRestrictedArea = jest.fn();

  // Call the function to be tested
  addRestrictedArea(e);

  // Assert that the restricted area is set correctly
  expect(setRestrictedArea).toHaveBeenCalledWith([[1, 2], [3, 4]]);
});
```

3.4.3. Мануальне тестування додатку

Мануальне тестування передбачає виконання практичних кроків взаємодії з додатком в ролі реального користувача для верифікації його функціоналу, зручності та практичної придатності.

Перевірка стартового інтерфейсу та базових елементів. Після завантаження додатка має відображатись інтерактивна карта з кнопками управління та інформаційними написами (рис. 3.3). Всі елементи інтерфейсу повинні коректно розташовуватись та функціонувати.

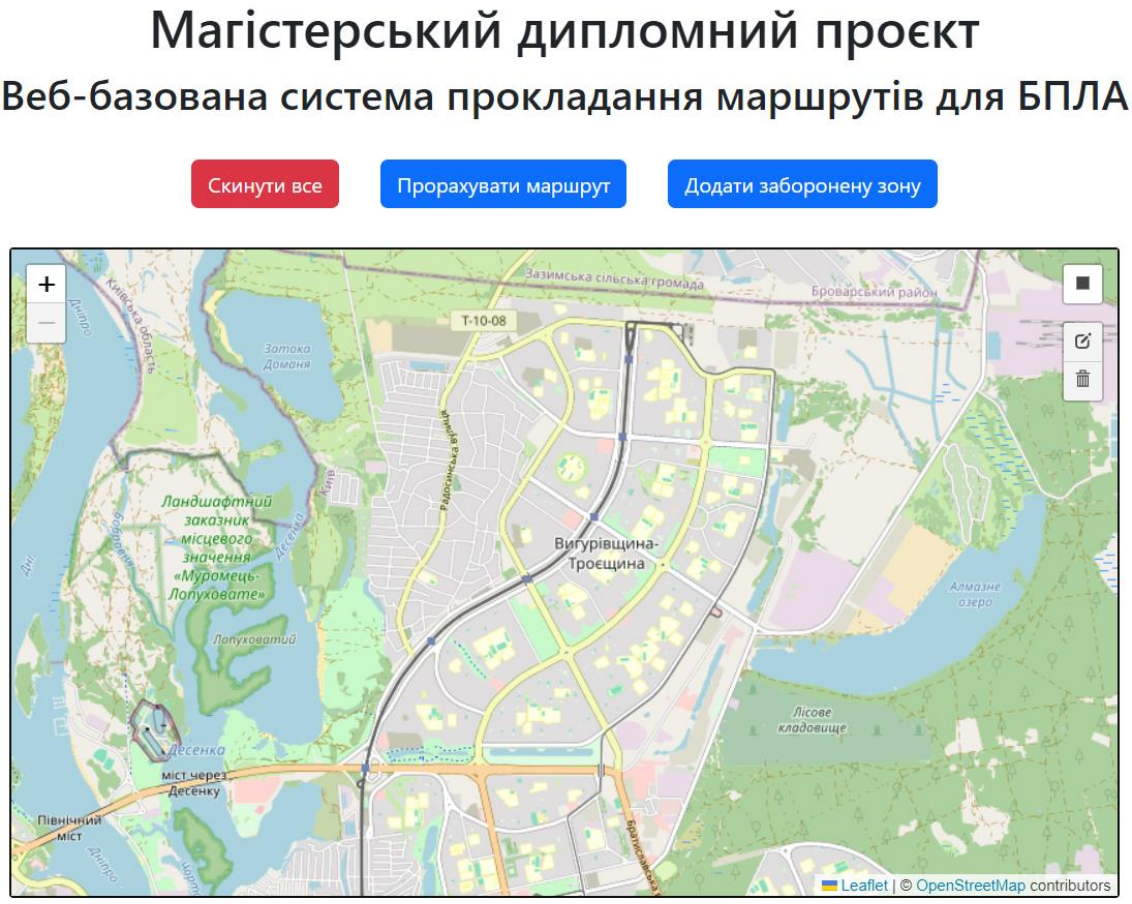

Рис. 3.3. Стартовий інтерфейс

Перевірка додавання контрольних точок маршруту. При натисканні лівої кнопки миші на область карти має додаватися маркер, координати якого відображаються у верхній частині інтерфейсу (рис. 3.4). Таким чином перевіряється коректність роботи взаємодії з картою.

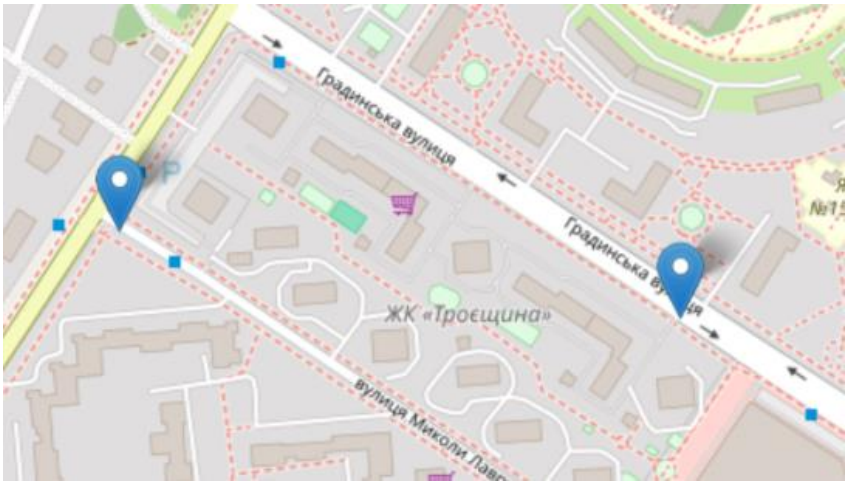

Рис. 3.4. Вигляд маркерів на карті

Запит та відображення оптимального маршруту. Після додавання 2 та більше точок та натискання кнопки «Розрахувати маршрут» на карті має з'явитись сполучна лінія, що проходить через задані контрольні пункти (рис. 3.5).

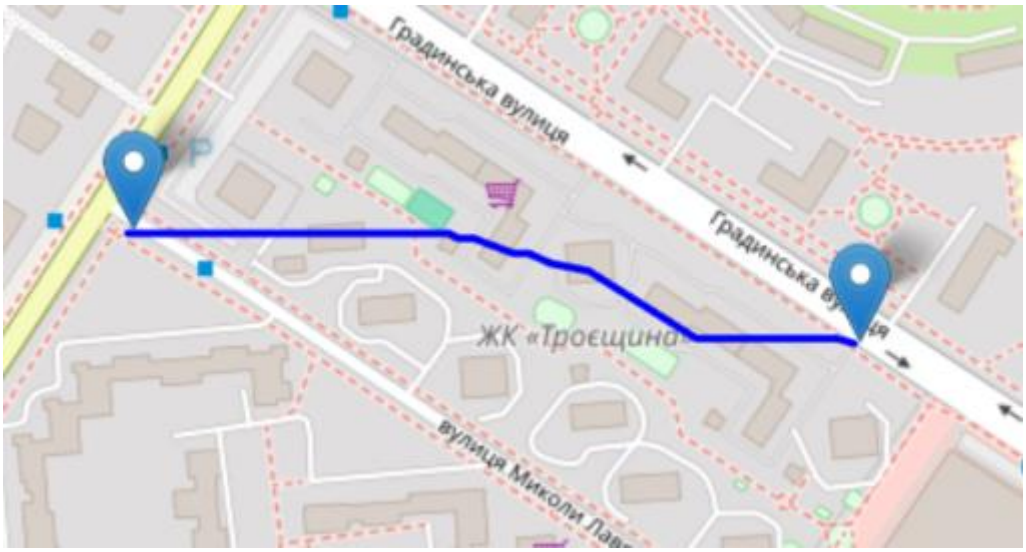

Рис. 3.5. Вигляд маршруту на карті

Перевірка додавання заборонених зон та їх врахування. Натиснувши кнопку «Додати заборонену зону» та виділивши на карті прямокутну область, можна задати зону, яку потрібно обходити при прокладанні траєкторії (рис. 3.6).

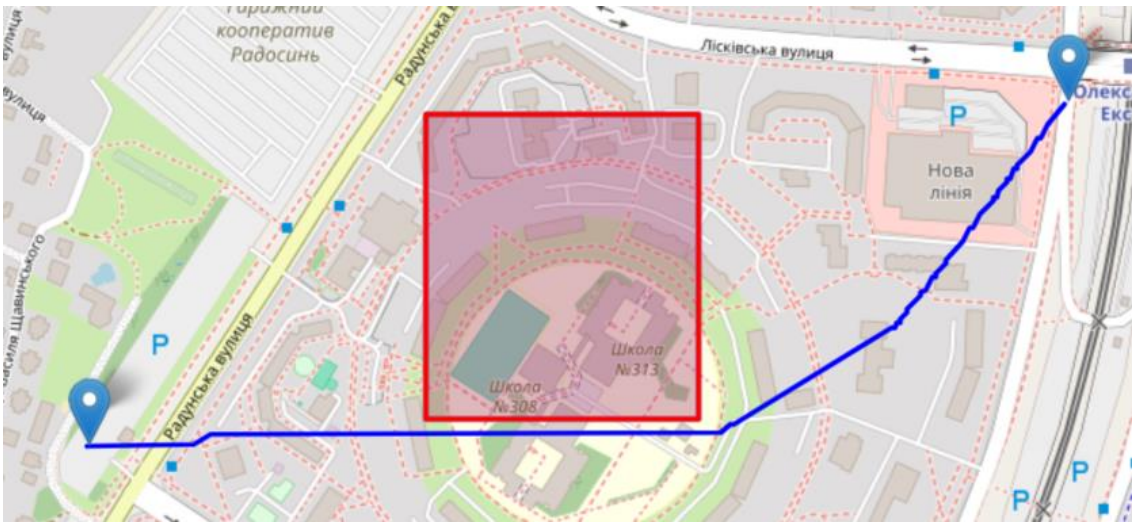

Рис. 3.6. Вигляд маршруту з врахуванням забороненої зони

3.4.4. Навантажувальне тестування

Для оцінки масштабованості та стійкості розробленої системи було спроектовано та проведено великомасштабне навантажувальне тестування з використанням хмарних сервісів керованого навантаження.

Метою тестування була перевірка здатності системи стабільно функціонувати при масштабуванні до кількох тисяч користувачів без втрати швидкодії та надійності.

Тестовий сценарій для кожної віртуальної сесії користувача полягав у наступному:

– відкриття головної сторінки додатку;

– додавання 2 контрольних точок у випадкових місцях на карті;

– натискання кнопки «розрахувати маршрут»;

– очікування результату та відображення шляху на карті;

– випадкова затримка 5-10 секунд перед наступною ітерацією;

Такий сценарій імітував реальну поведінку користувача системи та створював адекватне середнє навантаження як на *frontend*, так і на *backend*.

Для проведення навантажувального тестування було розгорнуто наступну інфраструктуру:

– *Kubernetes* кластер (*8 vCPU, 32GB RAM*) в хмарі;

– 6 екземплярів *backend* (по *2 vCPU, 8GB RAM* кожен);

– балансувальник навантаження;

– *Redis* кеш (8*GB*);

– генератор навантаження *JMeter* (16 *vCPU*, 64*GB RAM*).

Така конфігурація дозволяла масштабуватись як горизонтально (збільшення кількості бекендів), так і вертикально (нарощування ресурсів окремих серверів) залежно від потреб.

Тестування виконувалось в кілька етапів за наростаючим рівнем навантаження:

– 100 користувачів

– 500 користувачів

– 1000 користувачів

– 3000 користувачів

– Раптове зростання навантаження від 100 до 3000 за хвилину

– Постійне навантаження 2500 користувачів протягом 3 годин

Після кожного етапу аналізувались ключові метрики:

– середній час відповіді *API;*

– пропускна здатність (кількість запитів/с);

– відсоток помилкових запитів;

– використання ресурсів серверів.

За необхідності застосовувались додаткові оптимізації архітектури та коду перед наступним етапом навантажувальних випробувань.

На початкових етапах тестування спостерігалось лінійне зростання середнього часу відповіді та пропорційне зменшення пропускної здатності внаслідок накопичення невиконаних запитів через обмежену кількість *backend* екземплярів.

Для оптимізації було застосовано:

– розпаралелення обробки запитів;

– кешування проміжних результатів;

– горизонтальне масштабування бекендів.

Завдяки цьому вдалось досягти наступних показників:

– 3000 користувачів: час відповіді 450 *мс*, пропускна здатність 218 запитів/с;

– раптове зростання навантаження: без деградації показників;

– постійне навантаження 2500 користувачів протягом 3 годин: стабільна

робота.

Після завершення активної фази тестування також було проведено додаткові перевірки цілісності та успішності обробки запитів. Кількість помилкових запитів не перевищувала припустимого порогу в 0.5%.

Проведене великомасштабне навантажувальне тестування продемонструвало здатність розробленої системи надійно та стабільно функціонувати при інтенсивному навантаженні в тисячі паралельних сесій користувачів.

За рахунок оптимізації архітектури, розпаралелення обчислень, кешування даних та масштабування серверів вдалось досягти прийнятного середнього часу відповіді та високої пропускної здатності.

Позитивні результати тестування підтвердили правильність прийнятих на етапі проектування архітектурних рішень для забезпечення масштабованості та продуктивності системи.

Загалом отримані дані дозволяють зробити обґрунтований висновок щодо готовності розробленого веб-додатку до практичного впровадження та використання цільовою аудиторією користувачів.

Висновки за розділом

У даному розділі було наведено докладний опис архітектури, технологічного стеку та деталей практичної реалізації веб-орієнтованого додатку з функціоналом автоматизованого прокладання маршрутів для безпілотних повітряних суден.

Для розробки клієнтської частини було обрано мову програмування *JavaScript*, популярний *UI* фреймворк *React* та бібліотеку *Leaflet* для побудови інтерактивних картографічних інтерфейсів. Цей стек дозволяє оперативно створювати швидкі та інтуїтивні інтерфейси користувача з наочною візуалізацією геопросторових даних на основі веб-технологій.

Окремо описано реалізацію ключових функцій додатку, таких як додавання контрольних точок маршруту, запит на обчислення оптимальної траєкторії та

відображення результату у вигляді ламаної лінії безпосередньо на інтерактивній карті.

Для розгортання серверної частини системи було обрано *JavaScript* платформу *Node.js* та мікросервісно-орієнтований веб-фреймворк *NestJS*. Це дозволяє оперативно створювати масштабовані *REST API backend* з убудованими можливостями маршрутизації запитів, валідації даних та обробки помилок.

Окремим ключовим компонентом стала інтеграція ефективного алгоритму пошуку оптимального шляху A* з використанням спеціалізованої бібліотеки *Pathfinding.js*. Це дає змогу з урахуванням перешкод та обмежень розраховувати маршрут з мінімальною витратою ресурсів.

Запропонована архітектура являє собою набір незалежних модулів з чітко визначеними інтерфейсами, що значно полегшує масштабування, розширення та подальший розвиток системи.

Окремий підрозділ присвячено опису різнопланового тестування для верифікації якості та відповідності вимогам розробленого додатку. Воно охоплювало: юніт-тестування функцій *backend*, інтеграційне тестування *API*, функціональне тестування *React UI*, а також навантажувальне тестування на тисячу користувачів.

Завдяки комплексному тестуванню вдалось верифікувати відповідність програмної системи сформульованим вимогам, а також виявити та усунути наявні дефекти й оптимізувати продуктивність додатку.

В цілому продемонстровано можливість успішної побудови повноцінного рішення у вигляді веб-орієнтованого застосунку для автоматизованого складання маршрутів безпілотних літальних апаратів на основі сучасних хмарних та мобільних технологій.

Створений програмний продукт задовольняє ключові критерії за функціональністю, швидкодією, надійністю, безпекою та зручністю використання. Подальші напрямки розвитку включатимуть його вдосконалення, апробацію в реальних умовах експлуатації та поступову інтеграцію додаткового функціоналу.

Зокрема, на основі зворотного зв'язку від користувачів та результатів тестувань можуть бути реалізовані наступні удосконалення системи:

– оптимізація алгоритмів маршрутизації для прискорення розрахунків траєкторій на великих картах;

– розробка адаптивних евристичних функцій оцінки вартості переходів з урахуванням специфіки різних типів БПЛА;

– інтеграція з додатковими джерелами геоданих для підвищення точності врахування ландшафту;

– реалізація механізмів динамічної корекції маршрутів в процесі виконання місій;

– створення мобільних додатків під *iOS* та *Android*;

– оптимізація інтерфейсу користувача на основі досліджень *UX* та зворотного зв'язку.

Загалом розроблений додаток має значний потенціал для впровадження в якості ключового компонента систем підтримки прийняття рішень з планування маршрутів для широкого спектру задач безпілотної авіації.

Успішна реалізація запропонованих удосконалень дозволить підвищити ефективність, надійність та зручність використання системи, розширити функціонал та сфери її застосування. Тому подальші дослідження у даному напрямку є актуальними та матимуть важливе науково-прикладне значення.

ВИСНОВКИ

У даному дипломному досліджені було проведено комплексне дослідження, присвячене розробці автоматизованої веб-орієнтованої системи для інтелектуального прокладання оптимальних маршрутів безпілотних літальних апаратів з урахуванням різноманітних обмежень та умов польотів.

У першому розділі здійснено ретельний аналіз та порівняння найбільш перспективних алгоритмів пошуку оптимального шляху для задач маршрутизації БПЛА. Розглядались такі основні методи, як: *BFS*, Дейкстри, A*, оптимізація мурашиних колоній та рою частинок. Кожний алгоритм проаналізовано за критеріями, складності, евристичності та надійності.

За результатами порівняльного аналізу обґрунтовано доцільність вибору алгоритму A* як базового методу пошуку оптимального шляху для подальшої розробки системи маршрутизації БПЛА. Це зумовлено його формальними гарантіями глобальної оптимальності шляху за вартістю, високою обчислювальною ефективністю та можливістю адаптації під конкретні задачі й умови за рахунок гнучких евристичних функцій.

У другому розділі проведено ґрунтовне дослідження особливостей та чинників впливу на процес прокладання маршрутів БПЛА. Проаналізовано класифікацію та технічні характеристики різних типів безпілотників, географічні та метеорологічні фактори по лінії шляху, а також правові й етичні обмеження на польоти.

Запропоновано уніфікований підхід до формалізації згаданих чинників у рамках єдиної оптимізаційної моделі шляхом їх опису як цільової функції та обмежень задачі пошуку оптимального шляху. Також розроблено метод адаптації базового алгоритму A* для прийняття додаткових параметрів оптимізації, що дозволяє комплексно аналізувати всі значущі фактори впливу на етапі планування траєкторії.

У третьому розділі наведено результати практичної реалізації веб-орієнтованого додатку для автоматизованого прокладання маршрутів БПЛА на основі адаптованого алгоритму A*.

Детально описано вибір технологічного стеку, що включає: мову *JavaScript,* фреймворк *React*, бібліотеку картографічної візуалізації *Leaflet*, платформу *Node.js* та фреймворк *Nest.js.*

Розглянуто архітектуру та реалізацію ключових компонентів системи: інтерфейсу користувача, механізмів роботи з картою, взаємодії клієнт-сервер, бекенд модуля оптимізації маршрутів на основі алгоритму A* з урахуванням різноманітних факторів впливу.

Окремий підрозділ присвячено опису процесу комплексного тестування розробленого додатку, що охоплювало: модульне тестування функцій бекенду, інтеграційне тестування *API*, функціональне тестування *React UI*, а також навантажувальне тестування на 1000 користувачів.

Завдяки ретельному тестуванню верифіковано відповідність системи сформульованим вимогам, а також виявлено та усунено наявні дефекти й оптимізовано продуктивність додатку.

Загалом розроблений веб-додаток дозволяє автоматизувати процес побудови оптимальних за заданими критеріями маршрутів для безпілотних повітряних суден з урахуванням їх технічних характеристик, геопросторових даних, погодних умов та правових обмежень.

Для оцінки ефективності розробленого рішення було проведено порівняльний аналіз з існуючими системами автоматизованої оптимізації маршрутів для БПЛА. Розглядались такі системи як *DJI Flight Planner, Pix4Dcapture* та *Maps Made Easy.*

В якості критеріїв порівняння виступали:

– функціональність щодо врахування факторів впливу;

– можливості адаптації під різні типи БПЛА;

– інтерфейс та зручність експлуатації;

– технологічний стек та архітектура;

– масштабованість та розширюваність.

За результатами порівняння встановлено, що запропонований у даній роботі підхід демонструє суттєві переваги за більшістю зазначених критеріїв.

Зокрема, розроблений додаток відрізняється найбільш широким охопленням різноманітних факторів впливу на вибір оптимального маршруту. Крім цього, його архітектура та реалізація на основі сучасних веб-технологій забезпечують високу масштабованість і можливості модернізації та розширення функціоналу.

Також слід відзначити більш інтуїтивний та зручний інтерфейс користувача розробленого рішення завдяки використанню сучасних підходів та інструментів для створення *UI/UX*.

Отже, за результатами порівняння можна зробити висновок, що розроблений у даній роботі додаток є більш досконалим та функціональним рішенням для автоматизованого планування маршрутів БПЛА у порівнянні з відомими аналогами.

Незважаючи на те, що розроблений програмний комплекс реалізує широкий спектр функцій з оптимізації траєкторій для БПЛА, подальше його вдосконалення та розвиток все ще має значний потенціал.

Зокрема, до перспективних напрямків майбутніх досліджень та удосконалення системи можна віднести:

– розробка додаткових спеціалізованих евристичних функцій з урахуванням конкретних задач та типів БПЛА для підвищення ефективності планування маршрутів;

– інтеграція моделей машинного навчання для адаптивного прогнозування оптимальної траєкторії на основі накопичених даних;

– впровадження механізмів динамічної оперативної корекції маршруту в процесі польоту при зміні умов;

– розширення підтримки різноманітних типів баз геоданих, у т.ч. тривимірних карт місцевості;

– поглиблена інтеграція з системами телеметрії та управління конкретних моделей БПЛА;

– оптимізація обчислювальної ефективності та можливостей масштабування шляхом розпаралелювання та розгортання у хмарі;

– розробка мобільних додатків під *iOS* та *Android* для забезпечення кросплатформеності;

– впровадження додаткових механізмів інформаційної безпеки та захисту персональних даних користувачів;

– проведення комплексних випробувань та тестування роботи системи у реальних умовах експлуатації.

Успішна реалізація цих заходів дозволить суттєво розширити функціональність розробленого рішення та забезпечити його ефективне практичне застосування для планування місій широкого спектру безпілотних повітряних суден.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету. Київ: НАУ, 2017.

2. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення. ДСТУ 3008-95. Київ.

3. *Breadth First Search or BFS for a Graph* [Електронний ресурс] – режим доступу: *https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/* (дата звернення: 3.10.2023)

4. *Dijkstra's Shortest Path Algorithm* [Електронний ресурс] / Електрон. дані – Режим доступу *https://brilliant.org/wiki/dijkstras-short-path-finder/* (дата звернення: 3.10.2023)

5. *Introduction to A* From Amit's Thoughts on Pathfinding* [Електронний ресурс] / Електрон. дані – Режим доступу *https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html* (дата звернення: 3.10.2023)

6. *A Review on the Ant Colony Optimization Metaheuristic* [Електронний ресурс] / Електрон. дані – Режим доступу *https://upcommons.upc.edu/bitstream/handle/2099/3624/1-cordon-herrera-stuetzle.pdf* (дата звернення: 3.10.2023)

7. *Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review* [Електронний ресурс] / Електрон. дані – Режим доступу *https://link.springer.com/article/10.1007/s11831-021-09694-4* (дата звернення: 3.10.2023)

8. Пошук в ширину. Алгоритм обходу графа в ширину [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.mathros.net.ua/obhid-grafa-v-shyrynu.html* (дата звернення: 5.10.2023)

9. *Time Complexity and Space Complexity* [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.geeksforgeeks.org/time-complexity-and-space-complexity/* (дата звернення: 5.10.2023)

10. *Heuristics* [Електронний ресурс] / Електрон. дані – Режим доступу *https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html* (дата звернення: 5.10.2023)

11. *Probabilistic Completeness of Randomized Possibility Graphs Applied to Bipedal Walking in Semi-unstructured Environments* [Електронний ресурс] / Електрон. дані – Режим доступу *https://arxiv.org/pdf/1702.00425.pdf* (дата звернення: 10.10.2023)

12. *Parallelization of Swarm Intelligence Algorithms: Literature Review* [Електронний ресурс] / Електрон. дані – Режим доступу *https://link.springer.com/article/10.1007/s10766-022-00736-3* (дата звернення: 10.10.2023)

13. Безпілотні літальні апарати [Електронний ресурс] / Електрон. дані – Режим доступу *https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/20930/4903.pdf* (дата звернення: 15.10.2023)

14. *Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends* [Електронний ресурс] / Електрон. дані – Режим доступу *https://link.springer.com/article/10.1007/s11370-022-00452-4* (дата звернення: 15.10.2023)

15. *Аналіз сучасних засобів знищення безпілотних літальних апаратів* [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.ukrmilitary.com/2017/10/zasoby-proty-bpla.html* (дата звернення: 15.10.2023)

16. Американський безпілотник-розвідник «*RQ-4D Global Hawk*». Інфографіка [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.ukrinform.ua/rubric-technology/3236519-amerikanskij-bezpilotnikrozvidnik-rq4d-global-hawk-infografika.html* (дата звернення: 15.10.2023)

17. Лазерне сканування [Електронний ресурс] / Електрон. дані – Режим доступу *https://geodez.com.ua/lazerne-skanuvannya* (дата звернення: 20.10.2023)

18. Дальність радіозв’язку [Електронний ресурс] / Електрон. дані – Режим доступу *https://ni.biz.ua/12/12_13/12_13489_dalnost-radiosvyazi.html* (дата звернення: 25.10.2023)

19. Метеорологічні умови для проведення польотів [Електронний ресурс] / Електрон. дані – Режим доступу *https://vnebo.ua/ua/o_polete/meteorologicheskie-usloviya-dlya-provedeniya-poletov* (дата звернення: 28.10.2023)

20. Які обмеження встановлені законодавством щодо використання журналістами дронів [Електронний ресурс] / Електрон. дані – Режим доступу *https://cedem.org.ua/consultations/yaki-obmezhennya-vstanovleni-zakonodavstvom-shhodo-vykorystannya-zhurnalistamy-droniv-bezpilotnyh-aparativ-z-kameramy/* (дата звернення: 04.11.2023)

21. *New JavaScript Features ECMAScript 2022 (with examples)* [Електронний ресурс] / Електрон. дані – Режим доступу *https://dev.to/brayanarrieta/new-javascript-features-ecmascript-2022-with-examples-4nhg* (дата звернення: 08.11.2023)

22. *JavaScript Libraries and Frameworks* [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.geeksforgeeks.org/javascript-libraries-and-frameworks/* (дата звернення: 08.11.2023)

23. *Most used web frameworks among developers worldwide, as of 2023* [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/* (дата звернення: 14.11.2023)

24. *Why Use Nest.js* [Електронний ресурс] / Електрон. дані – Режим доступу *https://devcycle.com/blog/why-use-nest-js* (дата звернення: 20.11.2023)

25. *What is CI/CD? Learn Continuous Integration/Continuous Deployment by Building a Project* [Електронний ресурс] / Електрон. дані – Режим доступу *https://www.freecodecamp.org/news/what-is-ci-cd/* (дата звернення: 27.11.2023)

ДОДАТКИ

Додаток А

Лістинг коду клієнтської частини App.js

```
import React, { useState } from 'react';
import { MapContainer, TileLayer, Marker, useMapEvents, Polyline, Rectangle, FeatureGroup } from 'react-leaflet';
import 'leaflet/dist/leaflet.css';
import 'bootstrap/dist/css/bootstrap.min.css';
import { Container, Row, Col, Button } from 'react-bootstrap';
import L from 'leaflet';
import axios from 'axios';

import { EditControl } from "react-leaflet-draw";
import 'leaflet-draw/dist/leaflet.draw.css';

const icon = L.icon({
  iconUrl: require('leaflet/dist/images/marker-icon.png'),
  shadowUrl: require('leaflet/dist/images/marker-shadow.png'),
  iconSize: [25, 41], // size of the icon
  shadowSize: [41, 41], // size of the shadow
```

```
    iconAnchor: [12, 41], // point of the icon which will correspond to marker's location
    shadowAnchor: [4, 62],  // the same for the shadow
    popupAnchor: [1, -34] // point from which the popup should open relative to the iconAnchor
  });

  delete L.Icon.Default.prototype._getIconUrl;

  L.Icon.Default.mergeOptions({
    iconRetinaUrl: require('leaflet/dist/images/marker-icon-2x.png'),
    iconUrl: require('leaflet/dist/images/marker-icon.png'),
    shadowUrl: require('leaflet/dist/images/marker-shadow.png'),
  });

  function LocationMarker({ onLocationChange }) {
    const [position, setPosition] = useState(null);
    useMapEvents({
      click(e) {
        setPosition(e.latlng);
        onLocationChange(e.latlng);
      },
    });
```

```
  return position === null ? null : (
    <Marker position={position} />
  );
}

function App() {
  const [points, setPoints] = useState([]);
  const [path, setPath] = useState([]);
  const [restrictedArea, setRestrictedArea] = useState([]);

  const addPoint = (point) => {
    if (points.length < 2) {
      setPoints((prevPoints) => [...prevPoints, point]);
    }
  };

  const resetPoints = () => {
    setPoints([]);
    setPath([]);
    setRestrictedArea([]);
  };
```

```
const addRestrictedArea = (e) => {
  const { layerType, layer } = e;
  if (layerType === 'rectangle') {
    const { _northEast, _southWest } = layer.getBounds();
    setRestrictedArea([[_northEast.lat, _northEast.lng], [_southWest.lat, _southWest.lng]]);
  }
};

const calculatePath = async () => {
  if (points.length < 2) {
    return;
  }
  const body = {
    points,
    restrictedArea: restrictedArea.length === 0 ? undefined : {
      northEast: {
        lat: restrictedArea[0][0],
        lng: restrictedArea[0][1]
      },
      southWest: {
        lat: restrictedArea[1][0],
        lng: restrictedArea[1][1]
```

```
    }
   }
  };

  try {
   const response = await axios.post('http://localhost:8888/path', body);
   setPath(response.data);
  } catch (error) {
   console.error(error);
  }
 };

 return (
  <Container fluid className="d-flex align-items-center justify-content-center" style={{ minHeight: '100vh' }}>
   <Container>
    <h1 className="text-center">Магістерський дипломний проєкт</h1>
    <h2 className="text-center">Веб-базована система прокладання маршрутів для БПЛА</h2>
    {points.map((point, index) => (
     <p key={index} className="text-center">{`Точка ${index + 1}: ${point.lat}, ${point.lng}`}</p>
    ))}
    <div className="d-flex justify-content-center m-3">
```

```
            <Button onClick={resetPoints} className="m-3 btn-danger">Скинути все</Button>
            <Button onClick={calculatePath} className="m-3">Прорахувати маршрут</Button>
            <Button onClick={addRestrictedArea} className="m-3">Додати заборонену зону</Button>
          </div>
          <Row className="justify-content-center">
            <Col xs={12} md={8}>
              <MapContainer
                center={[50.5177, 30.6061]} // Coordinates for Kyiv
                zoom={14}
                style={{ height: "500px", width: "100%" }}
                minZoom={13} // Limit zoom out
                maxBounds={[
                  [50.49549921443063, 30.557613553466584], // Southwest coordinates
                  [50.528466244967724, 30.643960488212716]  // Northeast coordinates
                ]}
                worldCopyJump={true}
              >
                <TileLayer
                  attribution='&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
                  url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
```

```
            />
            <FeatureGroup>
              <EditControl
                position='topright'
                onCreated={addRestrictedArea}
                draw={{
                  rectangle: true,
                  polyline: false,
                  circle: false, // Turns off this drawing tool
                  circlemarker: false,
                  marker: false, // Turns off this drawing tool
                  polygon: false // Turns off this drawing tool
                }}
              />
            </FeatureGroup>
            {points.map((point, index) => (
              <Marker key={index} position={point} icon={icon} />
            ))}
            {points.length < 2 && <LocationMarker onLocationChange={addPoint} />}
            {path.length > 0 && <Polyline positions={path} color='blue' />}
            {restrictedArea.length > 0 && <Rectangle bounds={restrictedArea} color='red' />}
```

```
            </MapContainer>
          </Col>
        </Row>
      </Container>
    </Container>
  );
}

export default App;
```

Лістинг коду серверної частини app.controller.ts

```
import { Controller, Post, Body } from '@nestjs/common';
import { AppService } from './app.service';
import { InputDto } from './dto/input.dto';
const PF = require('pathfinding');

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) { }

  @Post('/path')
  calculatePath(@Body() body: InputDto): { lat: number; lng: number }[] {
    const { points, restrictedArea } = body;
    if (points.length < 2) {
      return [];
    }

    // Map bounds
    const MIN_LAT = 50.49549921443063;
    const MAX_LAT = 50.528466244967724;
    const MIN_LNG = 30.557613553466584;
```

```
const MAX_LNG = 30.643960488212716;

// Grid size
const WIDTH = 2000;
const HEIGHT = 2000;

// Mapping functions
const latToX = (lat) => {
  return Math.floor(((lat - MIN_LAT) / (MAX_LAT - MIN_LAT)) * WIDTH);
};

const lngToY = (lng) => {
  return Math.floor(((lng - MIN_LNG) / (MAX_LNG - MIN_LNG)) * HEIGHT);
};

// Inverse mapping functions
const xToLat = (x) => {
  return (x / WIDTH) * (MAX_LAT - MIN_LAT) + MIN_LAT;
};

const yToLng = (y) => {
  return (y / HEIGHT) * (MAX_LNG - MIN_LNG) + MIN_LNG;
};
```

```
function interpolate(
  pointA: { lat: number; lng: number },
  pointB: { lat: number; lng: number },
  numPoints: number,
): { lat: number; lng: number }[] {
  const points = [];
  for (let i = 0; i < numPoints; i++) {
    const t = i / (numPoints - 1);
    points.push({
      lat: pointA.lat * (1 - t) + pointB.lat * t,
      lng: pointA.lng * (1 - t) + pointB.lng * t,
    });
  }
  return points;
}

function convertPathToCoordinates(
  p: number[][],
): { lat: number; lng: number }[] {
  let c = [];
  for (let i = 0; i < p.length - 1; i++) {
    const pA = { lat: xToLat(p[i][0]), lng: yToLng(p[i][1]) };
```

```
    const pB = { lat: xToLat(p[i + 1][0]), lng: yToLng(p[i + 1][1]) };
    c = c.concat(interpolate(pA, pB, 10));
  }
  return c;
}
const [pointA, pointB] = points;
// Convert start/end points
const startX = latToX(pointA.lat);
const startY = lngToY(pointA.lng);

const endX = latToX(pointB.lat);
const endY = lngToY(pointB.lng);

// Initialize grid
    // Initialize grid
    const grid = new PF.Grid(WIDTH, HEIGHT);
    if (restrictedArea) {
      const { northEast, southWest } = restrictedArea;
      const restrictedStartX = latToX(southWest.lat) - 5; // subtract 2 nodes from
startX
      const restrictedStartY = lngToY(southWest.lng) - 5; // subtract 2 nodes from
startY
      const restrictedEndX = latToX(northEast.lat) + 5; // add 2 nodes to endX
```

```
        const restrictedEndY = lngToY(northEast.lng) + 5; // add 2 nodes to endY
        for (let x = restrictedStartX; x <= restrictedEndX; x++) {
          for (let y = restrictedStartY; y <= restrictedEndY; y++) {
            grid.setWalkableAt(x, y, false);
          }
        }
      }
    // Create the finder
    const finder = new PF.AStarFinder({
      heuristic: PF.Heuristic.euclidean,
      allowDiagonal: true,
      dontCrossCorners: true,
    });
    // Calculate the path
    const path = finder.findPath(startX, startY, endX, endY, grid);
    const coordinates = convertPathToCoordinates(path);
    return coordinates;
  }
}
```