МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

"_____"_____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних

Виконавець: Максим КОСЕНКО

Керівник: Ольга СУПРУН

Нормоконтролер: Євгеній ТУПОТА

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Олександр ЛИТВИНЕНКО

«_____»_____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

Косенка Максима Олексійовича

1. Тема кваліфікаційної роботи: «Програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією використанням оперативних даних»
затверджена наказом ректора від «28» серпня 2023 р. № 1494/ст
2. Термін виконання роботи: з 2 жовтня 2023 р. по 31 грудня 2023 р.
3. Вхідні дані до роботи: технічна документація, тестові дані, програмні продукти
4. Зміст пояснювальної записки: вступ, аналіз існуючих систем для оптимізації маршрутів, побудова схеми оптимізації маршрутів, програмний модуль генерації тестових даних, тестування та оцінка результатів, висновки.
5. Перелік обов'язкового графічного (ілюстрованого) матеріалу:
   1) життєвий цикл маршруту;
   2) головний інтерфейс при роботі модуля;
   3) структура хмарної БД *Firebase;*
   4) звітність в додатку;
   5) діаграма класів;
   6) схема алгоритму проходження всіх маршрутів.

6. Календарний план

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проаналізувати існуючі системи оптимізації маршрутів та сформувати основні вимоги до програмного модуля	02.10.2023 – 10.10.2023	
2	Визначити структуру та розробити програмний модуль для оптимізації маршрутів	11.10.2023 – 01.11.2023	
3	Розробка розділу 1: Аналіз існуючих систем для оптимізації маршрутів	02.11.2023 – 13.11.2023	
4	Розробка розділу 2: Побудова схеми оптимізації маршрутів	14.11.2023 – 20.11.2023	
5	Розробка розділу 3: Проектування програмного модуля	21.11.2023 – 01.12.2023	
6	Розробка розділу 4: Тестування та оцінка результатів	02.12.2023 – 12.12.2023	
7	Оформлення пояснювальної записки	13.12.2023 – 17.12.2023	
8	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 Ольга СУПРУН

Завдання прийняв до виконання Максим КОСЕНКО

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних»: 79 сторінок, 7 рисунків, 3 таблиці, 25 використаних джерел.

ЕФЕКТИВНІСТЬ ВИКОРИСТАННЯ ЛІТАКІВ, АЛГОРИТМ ОПТИМІЗАЦІЇ МАРШРУТІВ, ПРОГРАМНИЙ МОДУЛЬ.

Об’єкт дослідження дипломної роботи – транспортна логістика в авіаційній галузі, її обмеження та принципи роботи.

Предмет дослідження дипломної роботи – програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних.

Мета дипломної роботи – дослідження та розробка програмного модулю побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних.

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

Здійснено огляд існуючих систем оптимізації маршрутів; проаналізовано їх переваги та недоліки; досліджено процес оптимізації маршрутів та підвищення ефективності використання літаків; здійснено огляд існуючих методів оптимізації маршрутів; реалізовано мінімізацію та максимізацію цільових функцій, на основі яких обираються маршрути у створеному програмному.

Матеріали дипломної роботи рекомендується використовувати при розробці *BI* систем в авіаційні галузі, дослідженні алгоритмів оптимізації маршрутів та у навчальному процесі фахівців з системного програмування та тестування, а також у комерційних авіакомпаніях.

Прогнозні припущення про розвиток об’єкту та предмету дослідження – зручний програмний модуль для побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних. Модуль має дозволяє забезпечити зручний доступу до даних та візуалізацію результатів для різних рівнів управління авіакомпанією та полегшить постійний моніторинг показників та оновлення стратегій на основі нових даних і розвитку ситуацій. Логіка оптимізації має бути інкапсульована від кінцевого користувача, при цьому досить простою для подальшого розширення функціоналу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

BI (*Business Intelligence*) – системи аналізу даних про рейси, перевезення та фінансову інформацію для прийняття стратегічних рішень

MVC (*Model View Controller*) – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення

ECC (*Elliptic Curve Cryptography*) – асиметричний алгоритм шифрування, заснований на використанні еліптичних кривих

AES (*Advanced Encryption Standard*) – симетричний алгоритм блочного шифрування, прийнятий як американський стандарт шифрування урядом США

БД (База даних) – сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами

BLOC (*Business Logic Component*) – шаблон проєктування, який відділяє логіку роботи додатку від графічного представлення

ЗМТЗ – задача маршрутизації транспортних засобів

ВСТУП

Актуальність теми. Сучасне життя дуже важко уявити без авіаційної галузі: пасажирські перевезення, вантажні перевезення, патрулювання територій, гуманітарні місії та десятки інших застосувань вже є невід’ємною частиною нашого життя. Авіація дуже стара галузь, яка бере свій початок аж в кінці 18 століття, коли люди реалізували перші польоти на повітряних кулях, глайдерах та інших літаючих об’єктах.

В 1903 році було розроблено перший контрольований моторизований літак, який був проривом в галузі авіації. Однак, перші літаки мали виключно військове призначення, тобто були частиною військової авіації, та використовувались для спостереження, зв'язку та бойових дій.

Ближче до середини 20 століття, відбулись перші трансатлантичні та трансконтинентальні авіапольоти, які відкрили міжнародну цивільну авіацію.

Не зважаючи на довгу історію розвитку, та постійне покращення наявних літальних апаратів, оптимізація витрат є однією з ключових та найбільш актуальних проблем у авіаційній галузі.

Існує безліч речей, які можна оптимізувати: маршрути літаків, планування та розподіл пального, викиди CO_2, бортовий персонал, обробку багажу і пасажирів, тощо. Раніше всі ці параметри розраховувались людьми і мали досить велику похибку, яка призводила до зменшення показників ефективності. З часом певні розрахунки почали делегувати машинам, а в наш час майже повністю автоматизовано процеси різних розрахунків.

На сьогоднішній день існують наступні автоматизовані системи управління рухом:

– *Air Traffic Control Automation Systems*: ці системи допомагають управлінцям повітряним рухом координувати рух літаків, враховуючи їхні маршрути, висоту і швидкість. Вони включають системи автоматичного визначення позицій та розподілу радіочастот;

– *Flight Management Systems*: автоматизують навігацію, управління швидкістю і висотою літака, а також розраховують оптимальні маршрути;

– *Autopilot Systems*: дозволяють авіакомпаніям автоматично керувати літаками під час крейсерського польоту і при заходженні на посадку;

– *Maneuverability of inventories*: програмні рішення для планування і виконання ремонтних робіт та обслуговування літаків;

– *Inventory and logistics management*: використовуються для ведення обліку запасів, замовлення та управління логістикою авіакомпанії;

– *Health and Usage Monitoring Systems*: використовуються для неперервного моніторингу стану літаків і виявлення потенційних проблем;

– *Aviation Safety Management Systems*: забезпечують відстеження і аналіз інцидентів та нещасних випадків для підвищення безпеки авіаперевезень;

– *Risk Management Systems*: допомагають авіакомпаніям оцінювати та керувати ризиками в операціях;

– *Business Intelligence Systems*: аналізують дані про рейси, пасажирів і фінансову інформацію для прийняття стратегічних рішень;

– *Reporting Tools*: допомагають авіакомпаніям створювати звіти для внутрішнього управління та звітування перед регулюючими органами;

– *Baggage Handling Systems*: автоматизовані системи для обробки та відстеження багажу пасажирів;

– *Passenger Service Systems*: управляють резервaціями, квитками, реєстрацією та обслуговуванням пасажирів.

В цій роботі детально розглянуто існуючі системи *Business Intelligence*, які відповідають за аналіз даних про рейси, для побудови оптимальних маршрутів і прийняття стратегічних рішень, та запропоновано власну таку систему. Досліджено її переваги і недоліки, та зроблено висновки щодо можливості використання в різних цілях.

У основі програмного модулю побудови оптимального розкладу польотів є розробка системи *BI*. Основні інструменти *BI* можна розділити на дві групи: інструменти панелі управління та інструменти бізнес логіки.

Інструменти панелі управління містять в собі візуалізацію ключових метрик, таких як пунктуальність, завантаженість рейсів і витрати на пальне. Панель управління також дозволяє змінити певні вхідні дані, аби бізнес логіка спрацювала більш специфічно в конкретній ситуації. Останньою зоною відповідальності панелі управління є виготовлення звітів та аналітика для виявлення тенденцій та аномалій. Панель управління дозволяє забезпечити зручний доступу до даних та візуалізацію результатів для різних рівнів управління авіакомпанією та полегшить постійний моніторинг показників та оновлення стратегій на основі нових даних і розвитку ситуації.

Інструменти бізнес логіки виконують дві основні функції: використання алгоритмів прогнозування для розробки оптимальних розкладів. Формування стратегій ціноутворення та визначення і вимірювання ключових показників ефективності, для оцінки виконання стратегічних цілей авіакомпанії. Бізнес логіка в основному використовує лінійне програмування і оптимізаційні алгоритми з використанням математичних моделей, для визначення оптимальних маршрутів і розкладів та сценарний аналіз, для врахування різних сценаріїв, таких як зміни у погоді або попиті, для прийняття найкращих рішень.

Аналіз даних про рейси та побудова оптимальних маршрутів є дуже важливими для підвищення ефективності авіаційної діяльності, зменшення витрат і покращення якості обслуговування пасажирів. *BI*-системи дозволяють авіакомпаніям зробити інформовані рішення на основі об'єктивних даних і оптимізувати свою діяльність.

Мета і завдання виконання кваліфікаційної роботи. Метою кваліфікаційної роботи є дослідження та розробка програмного модулю побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних.

Для досягнення поставленої мети було поставлено наступні завдання:

– розробити систему *BI*, а саме інструментів панелі управління та інструменти бізнес логіки, які на основі початкових даних для маршрутів, виконують їх аналіз, розбиваючи маршрути на окремі сегменти;

– визначити оптимальні шляхи для зменшення споживання пального кожним літаком, та підвищення прибутковості перевезень;

– провести аналіз використання пасажирських сидінь, вантажного обладнання і пального на борту літака.

Об’єктом дослідження даної роботи є транспортна логістика в авіаційній галузі, її обмеження та принципи роботи.

Предметом дослідження кваліфікаційної роботи є програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних.

Методи дослідження. У ході роботи було використано ряд методів дослідження для тестування та оцінки ефективності розробленого програмного модуля для оптимізації маршрутів в авіаційній сфері. Методи дослідження включають в себе функціональне тестування, тестування продуктивності, тестування стабільності та надійності, а також тестування в реальних умовах використання.

Ці методи дослідження дозволять оцінити ефективність та продуктивність розробленого програмного модуля в різних умовах. Функціональне тестування дозволяє виявити покращення у часі проходження маршрутів. Тестування продуктивності визначає, наскільки ефективно модуль опрацьовує різні обсяги даних. Тестування стабільності виявляє деякі недоліки у роботі програмного модуля. Тестування в реальних умовах підтверджує придатність модуля до практичного використання авіакомпаніями.

Наукова новизна отриманих результатів. Зростання пасажиропотоку, зміни в геополітичному середовищі, а також вплив глобальних факторів, створюють потребу у нових стратегіях та інструментах. Оскільки алгоритми аналогічних систем не є загальнодоступними, можна вважати, що запропонований алгоритм є загальнодоступною унікальною основою для розробки та імплементації різних алгоритмів оптимізації маршрутів, які можуть бути протестовані в програмному модулі. Програмний модуль містить певні передумови, на які будуть накладатись додаткові обмеження, створюючи нові унікальні алгоритми.

Практичне значення отриманих результатів. Розроблено програмний модуль для оптимізації маршрутів в авіаційній сфері. Даний модуль може використовуватись як у авіаційній сфері, так і студентами у навчальних цілях. Програмний модуль має унікальний алгоритм вибору маршрутів та їх розподілення між літаками. Також модуль дозволяє вплинути на вибір маршрутів в процесі симуляції польотів, та отримати звітності про пройдені маршрути. Модуль може бути корисним для авіакомпаній, що не використовують систем оптимізації маршрутів, або роблять це за допомогою людського ресурсу, він дозволить зменшити кількість часу необхідну для побудови розкладу.

Особистий внесок випускника. Автором дипломної роботи було здійснено аналіз існуючих систем оптимізації маршрутів; досліджено структури та процеси у системах оптимізації маршрутів авіакомпаній; запропоновано алгоритм оптимізації маршрутів для підвищення ефективності використання літаків авіакомпанією; вивчено основні принципи та підходи для оптимізації маршрутів та підвищення ефективності використання літаків; проведено аналіз існуючих систем генерації маршрутів для літаків; вивчено особливості алгоритмів оптимізації та зроблено опис задачі для мінімізації та максимізації цільових функцій, за результатами якого створено програмний модуль побудови оптимального розкладу польотів та ефективного використання літаків авіакомпанією з використанням оперативних даних.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ СИСТЕМ ДЛЯ ОПТИМІЗАЦІЇ МАРШРУТІВ

1.1. Аналіз сучасних методів та стратегій для оптимізації маршрутів

Логістика – це наука, що досліджує оптимальне управління матеріальними та іншими супутніми потоками у будь-якій галузі людської діяльності. У сучасний період часу логістика стала однією з найбільш важливих наук і практик у сферах не лише економіки, але й в інших галузях, таких як політика, містобудування, інженерія, екологія та інші. Логістика фокусується на оптимізації функціонування систем, що базуються на потоках. Ці потоки можуть включати як традиційні розуміння слова «потоки» (такі, як нафтопроводи, газопроводи та інші комунікації), так і транспортні потоки, потоки ресурсів у виробництві та інформаційні потоки в мережах.

Оптимізація маршрутів – це важлива задача управління логістикою та транспортними системами, яка полягає в пошуку найбільш ефективного шляху для переміщення ресурсів або виконання послуг. Це може включати в себе розподіл товарів, пасажирські перевезення, експрес-доставку та інші види транспортних операцій. Оптимізація маршрутів стоїть перед завданням зменшення часу в дорозі, витрат пального, оптимізації використання транспортних засобів та підвищення загальної ефективності транспортно-логістичних процесів [3].

Оптимізація маршрутів у сфері авіатранспорту є ключовим елементом управління повітряним рухом та логістикою авіаперевезень [4]. Ця задача передбачає вибір оптимального шляху для польоту, мінімізацію витрат пального, оптимізацію коротших часових інтервалів між пунктами та максимально ефективне використання повітряного простору. Оптимізація може бути складним завданням, оскільки вона включає в себе ряд різних елементів. Оптимізований план польоту повинен не тільки враховувати правильну фізику (тобто характеристики літака і погодні умови), але й обмеження на маршрут, встановлені органами управління повітряним рухом, і всі відповідні регуляторні обмеження. Математична природа цих обмежень і загальний

обсяг розрахунків роблять цю задачу складною навіть за сучасними стандартами оптимізації. Деякі з рівнянь, що описують поведінку, є нелінійними і неперервними, а стан літака є динамічним (тобто залежить від того, як літак потрапив у певну точку, а не тільки від того, де він знаходиться). Як наслідок, для одного польоту потрібно виконати від десятків до сотень тисяч окремих розрахунків.

Важливість оптимізації маршрутів в авіатранспорті обумовлена наступним:

– мінімізація витрат пального: оптимізація маршрутів в авіатранспорті дозволяє зменшити витрати пального, що є ключовим фактором для авіакомпаній в умовах зростання цін на паливо;

– оптимізація транзитних зон: визначення оптимальних маршрутів дозволяє уникати транзитних зон та зменшити необхідність зупинок та змін напрямку;

– мінімізація часу в дорозі: шляхи, оптимізовані для авіатранспорту, дозволяють мінімізувати час подорожі, забезпечуючи швидкі та ефективні перевезення;

– зменшення затримок та очікувань: оптимізація маршрутів допомагає уникнути та зменшити затримки через оптимальне управління повітряним рухом;

– мінімізація конфліктів повітряного руху: оптимізовані маршрути дозволяють уникати конфліктів та підвищити безпеку повітряного руху;

– оптимізація використання повітряного простору: ефективне використання повітряного простору забезпечує оптимальне розміщення та рух повітряних суден.

Оптимізація авіатранспортних перевезень означає застосування методів та технологій, які надають можливість точно розраховувати час керування маршрутами та пов'язаними із ними витратами. Це включає в себе оптимальне розподілення ресурсів, мінімізацію часу на маршруті та ефективне використання повітряного простору. Поняття оптимізації вантажних перевезень в авіації полягає у постійному вдосконаленні системи перевезення, яка включає в себе доставку, завантаження/розвантаження та інші аспекти перевезень. Забезпечуючи постійні покращення, авіапідприємства можуть ефективно виконувати замовлення клієнтів, зменшуючи витрати та забезпечуючи максимальний рівень обслуговування [5].

Оптимізація авіатранспортних перевезень стає стратегічним інструментом для авіакомпаній, що дозволяє досягати високої ефективності в галузі перевезень та забезпечує конкурентні переваги на ринку авіаційних послуг. Найкращий маршрут для польоту залежить від фактичних умов для кожного польоту. До них відносяться прогнозовані вітер і температура повітря, кількість корисного вантажу, а також часові витрати в цей день. Часові витрати є особливо динамічними і залежать від величини корисного навантаження, а також від розкладу та експлуатаційних обмежень для екіпажу і літака. Задача маршрутизації транспортних засобів (ЗМТЗ або *VRP*, від англійської *Vehicle Routing Problem*) – це завдання визначення оптимальних маршрутів для транспортування вантажів від пунктів відправлення до пунктів призначення, огляду об'єктів і т.д. Основною метою цих задач є визначення набору маршрутів для обслуговування клієнтів, використовуючи певну кількість транспортних засобів, у заданому середовищі [6].

Транспортна задача в контексті авіатранспорту визначає алгоритмічний підхід до розв'язання завдань лінійного програмування або інших методів для визначення оптимального плану перевезень. В основі цих задач лежать змінні, які представляють точки – від постачальника до споживача (клієнта). Основною метою вирішення таких завдань є зниження витрат і максимальна оптимізація транспортної діяльності авіапідприємства. Наприклад, в сфері авіаперевезень компанії можуть використовувати інформацію про погодні умови та завантаженість аеропортів для того, щоб заздалегідь скорегувати маршрути своїх літаків. Інформованість, ефективність витрат, розрахунок оптимальних маршрутів та інші технології оптимізації дозволяють швидко доставляти вантаж із збереженням початкового вигляду та вартості.

Для оптимізації маршрутів використовуються наступні алгоритми оптимізації:

– алгоритм Дейкстри є методом пошуку найкоротших шляхів в графах з вагами ребер, використовується для знаходження найменшого шляху від стартової вершини до всіх інших вершин у направленому графі. Широко використовується в транспортних системах для визначення найбільш ефективних шляхів між різними пунктами;

– алгоритм A* комбінує в собі елементи алгоритмів пошуку в ширину та алгоритмів пошуку за найменшою вартістю. Використовується для пошуку оптимальних шляхів в графах, враховуючи евристичні оцінки вартості проходження через кожну вершину. Ефективно використовується в системах *GPS* та навігаційних додатках для визначення найшвидших та оптимальних маршрутів;

– генетичні алгоритми імітують природний відбір та еволюційний процес для пошуку оптимальних рішень. Створюють нові «особини» шляхом комбінації генетичного матеріалу кращих індивідів. Використовуються для пошуку найкращих маршрутів у транспортних мережах з урахуванням різних обмежень та умов;

– жадібні алгоритми приймають локально оптимальні рішення на кожному кроці з надією, що це призведе до глобально-оптимального рішення. Кожен вибір робиться на основі миттєвого оптимуму. Використовуються для найшвидшого вибору наступної вершини маршруту, може застосовуватися в ситуаціях, де глобальна оптимальність не є критичною [7];

– алгоритми імітації відпалу моделюють тепловий процес відпалу металу, де система «охолоджується» з високої енергії до стабільного стану. Використовуються для глобального пошуку оптимальних рішень, особливо в задачах з багатьма змінними та обмеженнями;

– алгоритми імітації відпалу є евристичним методом глобальної оптимізації, який використовує концепцію фізичного процесу відпалу металу. Цей алгоритм може використовуватися для вирішення задач оптимізації, особливо в тих випадках, коли необхідно знайти глобальний мінімум у функції цілісного обмеження.

Однією з найпростіших ЗМТЗ є *Travelling Salesman Problem* (*TSP*). Згадана задача полягає у пошуку найкоротшого Гамільтонового циклу в графі. Іншими словами, агент повинен пройти через всі вершини графа рівно один раз та повернутися до початкової вершини, пройшовши мінімальну відстань або витративши мінімальний час [8]. Існує різноманіття варіантів задачі комівояжера, таких як обмеження по вантажопідйомності, «часові вікна», альтернативні динамічні депо, повернення та доставка товарів, використання різних видів транспорту,

періодична маршрутизація і т.д. На рис. 1.1 представлена одна з можливих ієрархій класифікації цих задач.

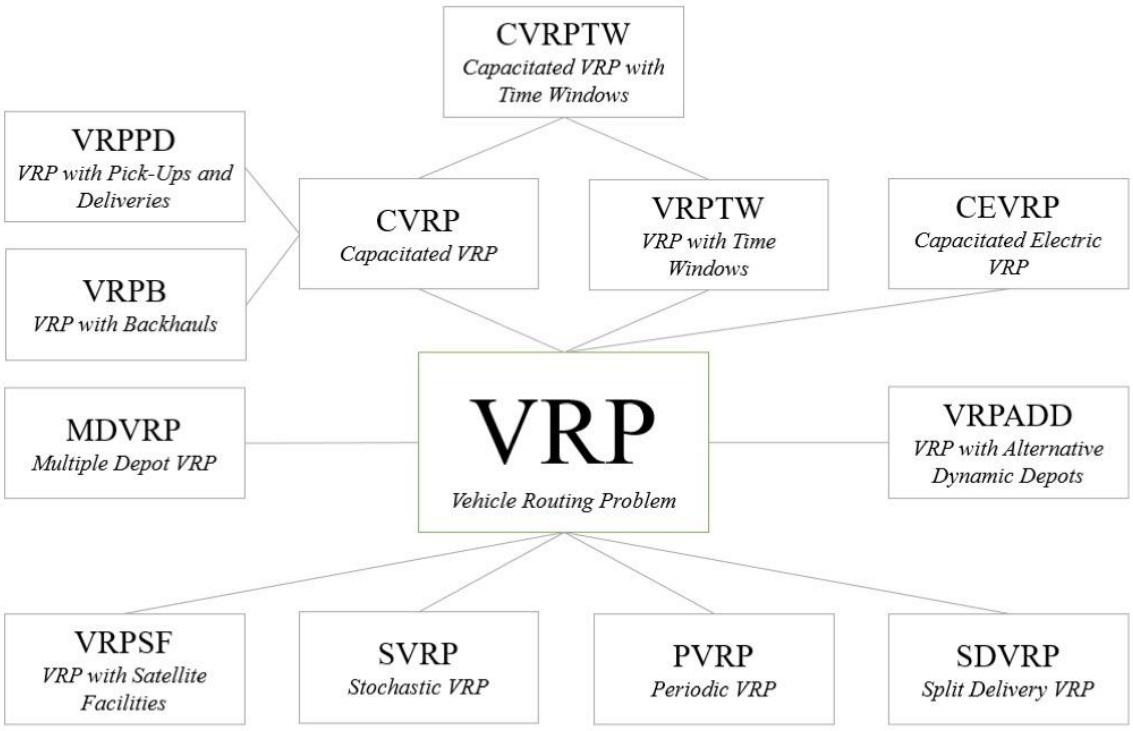

Рис. 1.1. Ієрархія основних класів задач комівояжера

Розглянемо більш детально кожен клас задач, які вказані на рис. 1.1. ЗМТЗ з обмеженнями на вантажопідйомність (*Capacitated VRP, CVRP*). Додатковою умовою такої задачі є обмеження на вагу вантажу для кожного маршруту, яке не повинно перевищувати певну визначену межу допустимого об'єму, який має бути доставлений до конкретної точки. Основною метою задачі є зменшення загальних витрат на транспортування. Задача маршрутизації з обмеженнями на вантажопідйомність можна розширити шляхом додавання додаткових умов, що призводить до появи нових класів задач [9].

Задача маршрутизації транспортних засобів з доставкою та поверненням товарів (*VRP with Pick-Ups and Deliveries, VRPPD*) відрізняється від класичної задачі транспортної маршрутизації тим, що клієнти на точках призначення можуть повертати товари, які транспортний засіб, у свою чергу, повинен повернути назад до депо. Однією з умов повернення товару є достатня кількість місця, щоб уникнути

перенавантаження транспортного засобу, тобто необхідно враховувати обмеження на вантажопідйомність [10]. Такі задачі стають складними у вирішенні через низку проблем, таких як ймовірність неефективного використання транспорту, зростання вартості маршрутів та загальної кількості транспортних засобів у депо. Зазвичай для спрощення враховують додаткові обмеження, наприклад, обмеження обміну товарами між точками доставки або обмеження відвідування точок доставки лише один раз [11]. Ще одним способом спрощення є припущення, що транспортний засіб спочатку роздає всі товари, а лише потім починає забирати товари з точок доставки (*VRP with Backhauls, VRPB*). Мета задачі полягає в мінімізації вартості перевезень і розміру флоту транспортних засобів. Одним із обмежень є те, що сумарна вага товарів, які потрібно доставити або забрати від клієнтів, не повинна перевищувати вантажопідйомність транспортного засобу в будь-якій точці маршруту.

Задача маршрутизації транспортних засобів зі зворотними перевезеннями (*VRP with Backhauls, VRPB*) відрізняється від задачі *VRPPD* (*Vehicle Routing Problem with Pickups and Deliveries*) тим, що в *VRPB* спочатку всі товари повинні бути доставлені, і лише потім можуть відбуватися повернення від клієнтів. У *VRPPD* важливо враховувати, що всі товари, які повертають клієнти, повинні поміщатися у транспортний засіб. Однак у випадку *VRPB* спершу проводиться доставка всіх товарів, а потім здійснюється можливість повернень. Ця особливість пов'язана з тим, що більшість транспортних засобів зазвичай завантажуються ззаду або зверху, утворюючи стек. Тому перегрупування вантажу під час кожної доставки є економічно неефективним.

Задача управління транспортними засобами з «часовими вікнами» (*VRP with Time Windows, VRPTW*) стикається з обмеженням, що вимагає доставки товару протягом конкретного часового інтервалу. У таких випадках встановлені верхні та нижні межі часу. Перевищення верхньої часової межі призводить до неприйняття товару або накладення штрафів. З іншого боку, якщо товар доставлено раніше ніж нижня часова межа, він має очікувати. Загальна мета полягає в мінімізації вартості перевезень, розміру транспортного парку та часу очікування. Ці задачі спрямовані на

визначення оптимального часу виїзду транспортного засобу з депо для уникнення зайвих простоїв [12].

Розвиток задачі *VRPTW* представлений вирішенням її за допомогою зворотних передач та часових вікон (*VRP with Backhauls and Time Windows, VRPBTW*). Ця задача враховує вантажопідйомність, повернення товарів, часові вікна і вирішується за допомогою гібридного метаевристичного алгоритму з метою мінімізації загальної відстані перевезень [13].

Задача ЗМТЗ з обмеженою вантажопідйомністю та часовими вікнами (*Capacitated VRP with Time Windows, CVRPTW*) включає в себе урахування двох важливих обмежень під час планування маршрутів транспортних засобів. По-перше, враховується максимальна вантажопідйомність кожного транспортного засобу, щоб гарантувати ефективне використання ресурсів. По-друге, встановлюються часові вікна для доставки товарів клієнтам, що визначають допустимий час для виконання кожного завдання.

ЗМТЗ з декількома депо (*Multiple Depot VRP*, *MDVRP*) передбачає, що існують кілька базових пунктів обслуговування (депо), і клієнти розподілені між ними [14]. Основна мета – знайти оптимальні маршрути для транспортних засобів, які мають обслуговувати клієнтів з різних депо, мінімізуючи загальні витрати перевезень та оптимізуючи розмір парку транспортних засобів [15].

ЗМТЗ з можливістю дозавантаження (*VRP with Satellite Facilities, VRPSF*) розглядає ситуацію, коли транспортний засіб може бути дозавантажений під час виконання маршруту, без повернення в депо. Це можливо завдяки використанню додаткових транспортних засобів, які можуть доставляти товари безпосередньо на маршруті [16]. Метою є мінімізація витрат на перевезення визначеної кількості товарів протягом фіксованого часу. Важливо враховувати, що хоча це може забезпечити більшу ефективність у визначений час, це також може призвести до збільшення загальних витрат через використання додаткових транспортних засобів [17].

Задача з випадковими даними у сфері перевезень (*Stochastic Vehicle Routing Problem, SVRP*) характеризується тим, що деякі елементи задачі випадковим чином

визначаються для даного класу транспортних задач [18]. Випадковість проявляється у випадковому визначенні точок доставки як ймовірнісних величин, а також у випадковому визначенні кількості товарів та відстані між точками доставки. Розв'язання таких задач відбувається у два етапи. На першому етапі отримується розв'язок без урахування додаткових змінних. Другий етап включає корекцію отриманого розв'язку на основі випадкових значень. Метою є мінімізація загальної вартості перевезень. У випадку невідомих точок доставки застосовуються додаткові обмеження, які виконуються з заданою ймовірністю або застосовуються моделі корекції при порушенні обмежень. Наприклад, для задачі *SVRP* з поверненням товарів та обмеженням вантажопідйомності транспортних засобів можуть бути застосовані різні методи корекції, такі як повернення до депо при перевантаженні, розвантаження та повернення на маршрут або планування дострокового повернення в депо, навіть якщо транспортний засіб не завантажений повністю. Модель корекції може залежати від кількості товарів і відстані до депо, і транспортний засіб може повертатися до депо, якщо збирання товару від наступного клієнта приведе до перевищення вантажопідйомності.

Задача з періодичною маршрутизацією (*Periodic VRP*, *PVRP*) характеризується розширеним періодом планування від одного до кількох днів. Основна мета полягає в зниженні вартості перевезень і оптимізації розміру автопарку. Додаткові обмеження включають можливість повернення транспортного засобу в депо не в той день, коли він виїхав, і необхідність відвідування кожної точки доставки протягом M-денного періоду щонайменше один раз [19]. Кожен клієнт повинен мати свій запит виконаний за один візит одним транспортним засобом. У випадку, коли період планування (M) дорівнює 1, задача стає класичною *VRP*. Кожна точка доставки в задачі з періодичною маршрутизацією повинна бути відвідана k раз, де $1 \leq k \leq M$. У класичній *PVRP* замовлення клієнта залишається незмінним щодня, тож задачу можна розглядати як оптимізацію груп маршрутів для кожного дня, де маршрути відповідають обмеженням, а загальна вартість перевезень є мінімальною.

Задачі з розділеною доставкою та різним транспортом (*Split Delivery VRP*, *SDVRP*) відрізняються від інших класів завдань тим, що одну точку доставки може

обслуговувати кілька транспортних засобів, якщо це дозволяє зменшити вартість доставки. Типовий приклад – ситуація, коли є значний обсяг товарів для доставки, але обмежена вантажопідйомність транспортних засобів. Мета цих задач – мінімізація витрат на перевезення та оптимізація розміру транспортного парку.

1.2. Вивчення існуючих програмних рішень та їхніх переваг і недоліків

Аналіз існуючих програмних рішень для оптимізації маршрутів авіатранспорту важливий для розуміння того, які можливості доступні та як кожна з них впливає на ефективність та економічність операцій.

Google Maps Platform – це набір інструментів та сервісів, які дозволяють розробникам і підприємствам інтегрувати і використовувати картографічні дані та функціонал *Google Maps* у своїх додатках та веб-сайтах. Включає в себе можливості візуалізації карт, розрахунку оптимальних маршрутів, роботи з геоданими та геолокацією. Застосовується в різних галузях, включаючи транспорт, логістику, туризм, тощо. Цей інструмент надає широкий функціонал для взаємодії з географічною інформацією, надає можливості відображення інтерактивних карт на веб-сайтах та в додатках, дозволяє розраховувати оптимальні маршрути для користувачів з врахуванням різних параметрів, таких як трафік, вид транспорту та інші, надає засоби перетворення адреси на географічні координати та навпаки, забезпечує можливості інтеграції з іншими сервісами, такими як *Google Places API,* для отримання додаткової інформації.

Основні переваги *Google Maps Platform*:

– широкий функціонал: має величезний набір функцій для візуалізації та роботи з геоданими;

– зручна інтеграція: легко інтегрується в додатки та веб-сайти за допомогою *API*.

– оновлення в реальному часі: забезпечує актуальні дані про трафік та інші параметри в режимі реального часу;

– глобальне покриття: доступний для використання в різних регіонах.

До недоліків можна віднести наступне:

– вартість: використання деяких функцій може бути дороговартістним;

– залежність від Інтернету: потребує наявності Інтернет-з'єднання для повного функціонування;

– обмежена персоналізація: обмежені можливості вносити значні зміни в інтерфейс або функціонал.

Google Earth Engine – це обчислювальна платформа, розроблена *Google*, яка спрямована на аналіз та обробку великих обсягів геопросторових даних. В основному використовується для наукових та дослідницьких цілей, зокрема для вивчення змін на Землі, в екології та кліматі. *Google Earth Engine* дозволяє користувачам обробляти та аналізувати великі обсяги геопросторових даних, такі як зображення високої роздільності, дані з датчиків зондування Землі, надає інструменти для вивчення та виявлення змін на поверхні Землі протягом часу та дозволяє науковцям та дослідникам спільно працювати над проектами, обмінюючись та аналізуючи геопросторові дані.

Основні переваги *Google Earth Engine*:

– обробка великих даних: здатність обробляти та аналізувати великі обсяги геопросторових даних;

– аналіз змін: можливість вивчати та виявляти зміни на Землі протягом часу;

– глобальний доступ: можливість аналізувати дані з різних частин світу;

– інструменти дослідження: забезпечує інструменти для ведення наукових досліджень та екологічного моніторингу.

До недоліків можна віднести наступне:

– складність використання: необхідність певного рівня експертизи у вивченні та використанні;

– залежність від обчислювальних ресурсів: для великих обсягів даних може знадобитися потужне обчислювальне обладнання;

– нецільовий для комерційних додатків: призначений для науковців і дослідників, не завжди ідеальний для комерційного використання.

Flight Planning Software тримає пілота в курсі всіх відповідних погодних умов, які змінюються і розвиваються з кожним годиною. Заздалегідь знаючи, в яку погоду прямує літак, пілот може приймати виважені рішення щодо положення літака і, таким чином, економити паливо. *Flight Planning Software* дозволяє планування маршрутів з урахуванням аеронавігаційних обмежень та інших факторів, що важливі для авіаційних подій. *ForeFlight* є прикладом *Flight Planning Software*, і являє собою інтегроване програмне рішення для планування та виконання польотів, призначеним для пілотів із загальної авіації та комерційного льотного складу. Це додаток, який надає пілотам засоби для оптимізації різних аспектів їхніх польотних операцій, починаючи від планування маршруту до виконання польоту та навігації. *ForeFlight* дозволяє пілотам визначити оптимальний маршрут для їхніх польотів, враховуючи погодні умови, обмеження повітряного простору та інші фактори; забезпечує інтерактивні картографічні дані для навігації в режимі реального часу, включаючи інформацію про аеропорти, повітряний простір, навігаційні точки; надає доступ до детальної інформації про погоду, включаючи метеорологічні умови на різних висотах, погодні карти та прогнози. *ForeFlight* допомагає пілотам ефективно планувати використання пального, враховуючи різні аспекти, такі як вага пального, дистанція та умови польоту та забезпечує можливість підключення до різних додаткових приладів та обладнання для полегшення виконання польоту. Дані та карти оновлюються в реальному часі, що дозволяє пілотам мати актуальну інформацію під час польоту.

Основні переваги *ForeFlight*:

– комплексність: *ForeFlight* надає всебічний набір інструментів для всіх етапів польоту;

– інтуїтивний інтерфейс: легкий у використанні, що дозволяє швидко освоїти всі функції;

– оновлення та підтримка: регулярні оновлення додають нові функції та покращують продукт.

До недоліків можна віднести наступне:

– вартість: *ForeFlight* є платним сервісом, що може бути вартими коштами;

– залежність від технічних засобів: ефективне використання може вимагати підключення до певного обладнання та технічних засобів;

– нецільовий для комерційної авіації.

SkyDemon – це програмне забезпечення для планування польотів, що спеціалізується на авіаційній навігації. Розроблене для використання в авіаційній галузі, це додаток, який надає пілотам та авіаторам засоби для вивчення маршрутів, розрахунку планів польотів та навігації під час польоту. *SkyDemon* надає доступ до детальних авіаційних карт та навігаційних даних. Додаток включає в себе інтерактивні карти, що дозволяють пілотам детально вивчати маршрути та аеропорти; забезпечує інструменти для складання та оптимізації маршрутів польоту з урахуванням погодних умов та інших факторів; надає можливість розрахунку часу польоту, відстані та паливних резервів. Автоматична генерація оптимального маршруту з врахуванням зон обмеження та погодних умов покращує ефективність планування та зменшує час, необхідний для підготовки до польоту; забезпечує актуальну інформацію під час польоту, включаючи позначення точок на маршруті, зони обмеження та дані про аеропорти. *SkyDemon* має можливість інтеграції з *GPS* та іншим обладнанням на борту літака, а також підтримує підключення до різних типів авіоніки для полегшення навігації. Загалом, *SkyDemon* є комплексним інструментом для планування та навігації в авіаційній галузі, надаючи пілотам зручні та потужні інструменти для безпечного та ефективного польоту.

Основні переваги *SkyDemon*:

– детальна авіаційна інформація: надання високоякісну та актуальну інформацію для авіаторів, включаючи навігаційні карти та дані про польоти;

– потужні інструменти планування польотів: *SkyDemon* пропонує інтуїтивно зрозумілі та ефективні інструменти для складання та оптимізації маршрутів;

– автоматичне планування: можливість автоматичного планування оптимальних маршрутів з врахуванням обмежень та умов польоту;

– навігаційна підтримка у реальному часі: забезпечення пілотів актуальною інформацією під час польоту, сприяючи прийняттю правильних рішень;

– інтеграція з *GPS* та авіонікою: підтримка інтеграції з *GPS* та іншим обладнанням, що полегшує навігацію на борту літака.

До недоліків можна віднести наступне:

– вартість: *SkyDemon* може бути дороговартістним продуктом, особливо для користувачів, які не використовують всі його функції;

– залежність від технічної інфраструктури: використання *GPS* та інших технічних засобів може бути обмежено або вимагати додаткових обладнань;

– підтримка функцій може залежати від регіону: деякі функції можуть бути обмежені або недоступні в певних регіонах.

Загально, *SkyDemon* є популярним інструментом у своєму класі, але вибір варто робити з урахуванням конкретних потреб та вимог користувача.

FlightAware – це платформа, яка спеціалізується на відстеженні польотів та надає різноманітні послуги для авіаційного спільноти. Платформа надає інформацію про польоти в реальному часі, історію польотів, а також інші сервіси для авіаційних професіоналів та ентузіастів.

Основні переваги *FlightAware*:

– оновлення в реальному часі: *FlightAware* забезпечує актуальну інформацію про рух польотів у реальному часі, включаючи поточне положення, швидкість та висоту літаків;

– глобальне покриття: платформа відстежує польоти в різних частинах світу, що робить її корисною для міжнародних подорожей та авіаперевезень;

– повідомлення про зміни: *FlightAware* надсилає повідомлення про зміни у розкладі польотів, включаючи затримки, скасування та інші події;

– широкий функціонал: окрім відстеження польотів, платформа має додаткові функції, такі як погодні дані, картографія, інструменти планування подорожей тощо.

До недоліків можна віднести наступне:

– неспрямований на планування маршрутів: *FlightAware* в першу чергу спрямований на надання інформації про рух польотів, а не на оптимізацію маршрутів для авіаційного транспорту;

– не забезпечує детальні метеодані: хоча надає загальні погодні дані, вони можуть бути обмеженими у порівнянні із спеціалізованими метеосервісами.

Jeppesen FliteDeck Pro – це електронний планшетний планер для пілотів, розроблений компанією *Jeppesen*, яка входить до складу *Boeing*. Цей інструмент призначений для покращення навігації та управління польотами в цифровому форматі. Він інтегрується з системами електронного керування польотами та надає доступ до електронних навігаційних карт, процедур та іншої інформації для пілотів.

Основні переваги *Jeppesen FliteDeck Pro*:

– електронні навігаційні карти: забезпечення доступу до актуальних та детальних електронних карт для польотів;

– інтеграція з системами електронного керування: покращення управління польотами та обміну даними;

– оновлення в реальному часі: можливість отримання оновлень карт та інформації в реальному часі під час польотів;

– підтримка технічних даних літака: надання технічних даних та параметрів літака, що полегшує планування та виконання маршрутів;

– зберігання журналів польотів: це дозволяє пілотам вести електронні журнали польотів та взаємодіяти з ними.

До недоліків можна віднести наступне:

– вартість: використання *Jeppesen FliteDeck Pro* може бути дороговартістним, особливо для індивідуальних пілотів;

– обмежена доступність для загальної авіації: програма орієнтована переважно на великі авіакомпанії та пілотів комерційної авіації, тож може бути надто потужним для загальної авіації або невеликих операторів;

– залежність від технічних засобів: вимагає сумісного обладнання та технічного забезпечення для повного функціонування.

Вибір програмного забезпечення для оптимізації маршрутів авіатранспорту повинен бути обґрунтованим і здійснюватися враховуючи конкретні потреби та обмеження користувача чи компанії. Кожен інструмент має свої унікальні переваги

та недоліки, і важливо зрозуміти, які саме аспекти є ключовими для конкретного використання.

1.3. Використання оперативних даних у системах керування авіаційним транспортом

Оперативні дані в авіаційному транспорті представляють собою інформацію, яка збирається та оновлюється в режимі реального часу під час експлуатації літаків та пов'язаних процесів. Вони є ключовим елементом для прийняття обґрунтованих рішень та ефективного керування авіаційним транспортом.

Класифікація оперативних даних включає різні аспекти:

– технічні дані літаків. Інформація про стан обладнання літаків, параметри двигунів, системи безпеки, та інші технічні характеристики;

– метеодані. Дані про погодні умови на маршруті польоту, включаючи інформацію про вітер, турбуленції, температуру, та інші метеорологічні фактори;

– інформація про трафік. Дані щодо розташування та руху інших літаків, що перебувають у повітрі, для уникнення зіткнень та планування оптимальних маршрутів;

– дані про пасажиропотік. Інформація про кількість та тип пасажирів, обсяг багажу, що дозволяє авіакомпаніям оптимізувати обслуговування та ресурси;

– дані технічного обслуговування. Інформація про технічний стан літаків та проведені роботи з обслуговування.

Розуміння цих категорій оперативних даних є важливим для розробки та впровадження систем керування авіаційним транспортом, оскільки вони надають комплексну картину процесів у галузі авіаційної діяльності та їхніх взаємозв'язків.

Динамічні системи керування в авіації включають у себе різноманітні процеси, що змінюються в часі, від планування рейсів та обслуговування літаків до координації пасажирських та вантажних операцій. Нижче розглянуто роль та значення оперативних даних у цих системах.

Оперативні дані надають інформацію для швидкого та обґрунтованого прийняття рішень в реальному часі. Наприклад, врахування метеоумов, трафіку та технічного стану літаків для оптимізації маршрутів польоту в реальному часі. Оперативні дані дозволяють ефективно використовувати ресурси авіакомпаній, включаючи повітряний флот, пасажирські потоки та персонал. Оперативні дані грають ключову роль у забезпеченні безпеки польотів та технічної надійності літаків. Наприклад, моніторинг стану двигунів та систем літаків для завчасного виявлення потенційних проблем та запобігання аваріям. Неменш важливі оперативні дані для координації та оптимізації руху повітряного транспорту, особливо в зоні аеропортів та їхніх оточуючих просторах, а також взаємодіють з іншими технічними та оперативними системами для забезпечення комплексного керування повітряним простором. Наприклад, моніторинг та регулювання потоків літаків для зменшення заторів та затримок, та інтеграція даних з системами трафіку, метеорологічними службами та системами безпеки.

Загальна мета збору та використання оперативних даних у динамічних системах керування полягає в тому, щоб забезпечити оптимальну ефективність, безпеку та ресурсозбереження в авіаційній діяльності. Технічні засоби для збору оперативних даних включають в себе різноманітні сенсори, датчики та системи зв'язку, розташовані на літаках, аеропортах та інших ключових точках. Системи моніторингу в режимі реального часу здатні отримувати, аналізувати та передавати дані в автоматизованому режимі, забезпечуючи операторам та системам керування необхідну інформацію. Обробка оперативних даних вимагає використання різноманітних методів та алгоритмів, що забезпечують високу швидкість та точність обчислень. Автоматизовані системи використовують алгоритми для фільтрації шуму, визначення виняткових ситуацій, прогнозування та оптимізації. Машинне навчання та штучний інтелект можуть застосовуватися для аналізу великих обсягів даних та виявлення патернів.

З сучасним розвитком технологій, використання хмарних систем, великих даних (*Big Data*), технологій Інтернету речей (*IoT*) та блокчейну стає все більш розповсюдженим у зборі та обробці оперативних даних. Ці технології дозволяють

збирати та обробляти великі обсяги даних в режимі реального часу, підвищуючи ефективність та точність управління. Успішна обробка оперативних даних вимагає інтеграції різних систем та підсистем. Впровадження стандартів та протоколів обміну даними дозволяє різним компонентам системи ефективно взаємодіяти та забезпечує системну злагодженість. Застосування стандартів безпеки є критичним аспектом для забезпечення конфіденційності та цілісності даних. Розробка ефективних архітектур систем збору та обробки даних важлива для забезпечення продуктивності та швидкодії. Використання високоефективних апаратних рішень та оптимізованих алгоритмів може значно поліпшити продуктивність систем. Технічні засоби та підходи для збору оперативних використовуються для реалізації систем, які дозволяють збирати, обробляти та аналізувати оперативні дані в авіаційному транспорті. Вони забезпечують операторам та менеджерам необхідну інформацію для ефективного керування та прийняття рішень в реальному часі.

Обробка оперативних даних в авіаційному транспорті вимагає застосування різноманітних методів та алгоритмів для забезпечення точності, швидкості та ефективності обчислень. Нижче розглянуто ключові методи та алгоритми, які застосовуються в цьому контексті.

– фільтрація даних: фільтрація даних є ключовим етапом в обробці оперативних даних. Методи фільтрації використовуються для виявлення та видалення шумів, артефактів або неправильних вимірювань;

– прогнозування: алгоритми прогнозування використовуються для апроксимації майбутніх значень на основі історичних даних. У контексті авіаційного транспорту, це може включати прогнозування метеорологічних умов, пасажирських потоків, або траєкторій літаків;

– кластеризація та класифікація: алгоритми кластеризації та класифікації використовуються для групування або класифікації оперативних даних за певними ознаками. В авіаційному контексті це може включати класифікацію типів літаків, аналізу безпеки, чи визначення ступенів технічного стану;

– аналіз часових рядів: для оперативних даних, які відображаються в часі, використовують алгоритми для аналізу часових рядів. Це може бути важливим для виявлення тенденцій, сезонності та інших патернів;

– оптимізація та маршрутизація: для оптимального використання ресурсів та ефективності маршрутів, використовують алгоритми оптимізації та маршрутизації.

1.4. Висновки до розділу

У першому розділі було розглянуто різноманітні методи та стратегії для оптимізації маршрутів у транспортних системах, а також підкреслюється важливість ефективного управління рухом та логістичними процесами.

Розглянуті алгоритми включають в себе класичні та сучасні підходи, які дозволяють вирішувати завдання оптимізації в різних галузях, включаючи автомобільний та авіатранспорт, логістику та доставку.

Методи оптимізації маршрутів, такі як алгоритм Дейкстри, A*, генетичні алгоритми, жадібні алгоритми та алгоритми імітації відпалу, надають різні підходи до вирішення складних задач управління транспортними потоками. Кожен з них має свої переваги та області застосування, що визначає їхню ефективність в конкретних сценаріях. ЗМТЗ, така як задача комівояжера, є типовим прикладом оптимізаційної задачі, де важливою є здатність знаходження найбільш оптимального маршруту для ефективного пересування агента або транспортного засобу. Різноманітні варіації ЗМТЗ, включаючи обмеження по вантажопідйомності, часові вікна, різні види транспорту та інші умови, вказують на велику варіабельність цих задач і необхідність застосування різних стратегій для їх ефективного вирішення.

Розвиток та застосування методів оптимізації маршрутів в сучасних транспортних системах сприяє підвищенню продуктивності, ефективності та економічності у сфері транспорту та логістики. Вдосконалення цих методів відкриває нові можливості для подолання викликів, пов'язаних із ростом обсягів транспортних потоків та ускладненням логістики.

РОЗДІЛ 2
ПОБУДОВА СХЕМИ ОПТИМІЗАЦІЇ МАРШРУТІВ

Однією з ключових проблем, які стикаються авіакомпанії в сучасному світі, є необхідність оптимізації маршрутів та ефективного використання ресурсів для підвищення конкурентоспроможності та зниження експлуатаційних витрат. Побудова ефективної схеми оптимізації маршрутів є складним завданням, яке вимагає врахування різноманітних факторів, таких як погодні умови, обсяг пасажиропотоку, наявність альтернативних аеропортів та інші аспекти. У цьому розділі буде розглянуто процес розробки схеми оптимізації маршрутів, включаючи визначення критеріїв ефективності, вибір алгоритмів оптимізації, та створення програмного коду інструменту, який дозволить авіакомпаніям автоматизувати та поліпшити процес управління маршрутами.

2.1. Визначення початкових умов маршрутів для схеми оптимізації

В основі маршрутів лежить три ключових елементи: літаки, аеропорти та вантаж або пасажири яких треба перевезти. Кожен із цих елементів необхідно розглядати як окремий об’єкт зі своїми властивостями. Це допоможе у розробці алгоритму та при перенесенні алгоритму в код, адже у програмуванні такий підхід називається об’єктно-орієнтованим.

Об’єкт літака – модель даних, яка мусить містити наступну публічну інформацію, визначену на початку роботи програми:

- назва літака;
- позиція в якій літак знаходиться до початку руху;
- маршрут за яким літак буде рухатись;
- технічний стан літака;
- кількість пального;
- вартість літака.

Окрім цього, є також приховані дані, які не є видимими для користувача та інкапсулюють логіку роботи літака: перевірка на досягнення проміжної точки та поточна позиція відносно початку та кінця маршруту, які змінюються в процесі проходження маршруту.

Аеропорти містять лише інформацію про назву, та позицію, яка репрезентується координатами *x* та *y*.

Вантаж для перевезення містить в собі інформацію про аеропорти, з яких він вирушає та куди має потрапити, а також пріоритетність перевезення, кількість та ціну за одиницю. Пріоритетність перевезення є важливим параметром, який буде впливати на схему оптимізації таким чином, що навіть найбільш підходящий та прибутковий маршрут літака буде ігноруватись, якщо існує перевезення з вищим пріоритетом. На основі цих трьох моделей даних, створюються модель маршруту, та передається в об’єкт літака. Модель маршруту містить наступні дані:

– точки початку та кінця;
– проміжна точка маршруту;
– прибутковість маршруту.

На основі цих даних, формуються обмеження та правила, за якими можливі перевезення, та їхня подальша оптимізація. Для літака існують наступні обмеження:

– літак не може знаходитись поза координатною площиною визначеною для маршрутів;

– літак не може знаходитись поза аеропортом на момент початку роботи симуляції польоту;

– літак обмежений в пальному, тобто якщо кількість пального менша, ніж потрібно на маршрут плюс половина цієї кількості на екстрені випадки, політ не можливий;

– літак має певну кількість місця для транспортування і не може взяти більше ресурсів для транспортування, ніж об’єм цього місця;

– літак має кілька станів, і в разі якщо літак пошкоджений – маршрутом має опрацювати інший літак.

Якщо ж літак має критичні пошкодження під час маршруту – він має зупинитись в найближчому доступному аеропорту.

Оскільки цей проєкт лише симулює роботу справжньої системи, в обмеження маршрутів було додано кілька пунктів, які зменшують реалістичність, а також зменшують складність системи. Це дозволило створити її за менший період часу, а також допомогло більш очевидно продемонструвати оптимізацію.

Список обмежень маршрутів:

– ресурси для транспортування можуть бути взяті лише в повному обсязі, за один політ, одним літаком;

– маршрут з критичним пріоритетом має відбутись раніше, ніж маршрут з нижчим пріоритетом;

– маршрут має точку початку і кінця, тобто літак обов’язково має відвідати ці дві точки;

– наступний маршрут визначається лише після завершення поточного;

– якщо літак має пройти маршрут, але не знаходиться в його початковій точці, стартова точка маршруту стає проміжною точкою, а поточна позиція літака – стає початковою точкою;

– симуляція вважається закінченою, якщо більше немає доступних маршрутів

– список маршрутів визначається на основі ресурсів транспортування, але для спрощення системи, не можливо визначити ресурс транспортування, який не може бути перевезений;

– список маршрутів не обмежений і не фіналізований, він може змінюватись, для отримання різних тестових результатів.

2.2. Математична модель задачі

Для того, аби виконати поставлену задачу, необхідно спочатку розробити математичну модель, що дозволить формалізувати різноманітні аспекти процесу оптимізації маршрутів, враховуючи різні вхідні параметри. Це дозволяє отримати точні та передбачувані результати в залежності від обраної моделі. Математичні

моделі дозволяють вам визначити оптимальні рішення, враховуючи обмеження та вимоги. Математичні моделі допомагають знайти баланс між різними факторами, такими як час, витрати пального, кількість пересадок та інші. За допомогою математичних моделей можливо легко моделювати різні сценарії та визначати, які зміни в параметрах можуть впливати на результати оптимізації маршрутів. Це дозволить проводити аналіз чутливості та вдосконалювати стратегії. Хоча розробка математичної моделі може здаватися складною, однак при використанні оптимізаційних алгоритмів вона може значно зменшити обчислювальні витрати порівняно з прямими методами. Не менш важливим є той факт, що математичні моделі можуть бути легко інтегровані з іншими інформаційними системами та програмним забезпеченням, що використовується в авіакомпаніях.

Для того, аби розробити математичну модель, необхідно пройти через загальний підхід створення моделі, що базується на теорії оптимізації та лінійному програмуванні:

– вибір основних параметрів, які впливають на ефективність маршрутів. Це може включати витрати пального, час подорожі, кількість пересадок, обсяг пасажиропотоку та інші;

– формування цільової функції яку треба максимізувати чи мінімізувати. Наприклад, це може бути мінімізація витрат пального, мінімізація часу подорожі чи максимізація прибутковості;

– накладання обмежень, які обмежують варіанти оптимізації. Це може включати обмеження бюджету, максимальну кількість літаків, мінімальну кількість перевезених пасажирів та інші обмеження;

– введення змінних рішень, які представляють варіанти маршрутів та рішення. Наприклад, кількість рейсів між двома містами або час затримки для кожного літака;

– математична формалізація, під час якої використовуються математичні формули для вираження цільової функції та обмежень. Зазвичай використовують лінійне програмування чи інші методи оптимізації для максимізації чи мінімізації цільової функції;

– отримання розв’язку, використовуючи відповідні методи оптимізації для отримання оптимальних значень змінних рішення, що відповідають оптимальним маршрутам.

При виборі основних параметрів було обрано наступні:

– пріоритетність маршруту;
– прибутковість маршруту;
– можливість опрацювання маршруту літаком;
– позиція літака відносно перевезення.

Пріоритетність маршруту, це значення, за яким маршрути розкладаються по відповідним спискам пріоритетів. В програмному модуль реалізовано наступні пріоритети: *Critical, High, Mid, Low.*

Списки розташовані в порядку зростання, тобто маршрут з вищою пріоритетністю буде завжди виконаний раніше, ніж маршрут з нижчим пріоритетом, навіть якщо його прибутковість вища. Це обумовлено тим, що система має на меті наблизитись до реального світу, де якісь медичні перевезення органів або транспортування тяжкого хворого має більшу цінність ніж прибуток з перевезення якогось вантажу. Прибутковість маршруту визначається як різниця прибутку від маршруту та затрат на нього. Прибуток маршруту визначається досить просто, це добуток кількості одиниць ресурсу що перевозиться та вартості за одиницю ресурсу. Витрати на маршрут складаються з суми витрат на пальне та витрат на обслуговування. Середня витрата пального вантажним літаком – 6 *л* на 1 *км*. Оскільки зона польотів визначена як координатна площина, де максимальне значення *x* координати дорівнює 30 одиниць, а *y* – 12 одиниць, вважається, що 1 одиниця відстані на цій площині – дорівнює 44 *км*. Таким чином, зважаючи на дистанцію польоту, витрати на пальне визначаються як 264 *л* на одиницю відстані помножені на умовну ціну пального рівну 100 *грн* за 1 *л*. Ціна на обслуговування включає в себе вартість ремонту за пройдений кілометраж, та виплату коштів обслуговуючому персоналу, що визначається як 1000 *грн* на одиницю відстані. Не менш важливим фактором є сама відстань маршруту, адже вона може змінюватись, залежно від

позиції в якій знаходиться літак на початку маршруту. Алгоритм намагається обрати літаку найбільш прибутковий маршрут з точки в якій він знаходиться, однак якщо такого маршруту немає, літак отримує найбільш прибутковий маршрут з решти маршрутів, з урахуванням витрат на переміщення в точку початку маршруту.

Можливість опрацювання маршруту літаком, визначається за сукупністю змінних. За обмеженнями нашої інформаційної системи, перевезення не може бути розбито на кілька етапів, тобто літак не може опрацювати маршрут, в якому міститься більша кількість вантажу ніж його вантажопідйомність. Окрім цього, існує обмеження на кількість палива в літаку, його кількість має бути в два рази більша, ніж необхідно на дорогу в обидві сторони за маршрутом.

Позиція літака впливає те, який маршрут буде обрано. Інформаційна система створена таким чином, що якщо існують маршрути з однаковою пріоритетністю, то буде обрано найбільш прибутковий маршрут з точки, в якій літак знаходиться. Тобто маршрути які мають більшу прибутковість, але знаходяться в іншій точці, будуть проігноровані. Таке обмеження суттєво зменшує час, за який всі маршрути проходяться.

Конвертуємо описані текстом параметри в параметри для математичної моделі [20]. Міста між якими відбувається маршрут позначимо як i та j, k – точка, що проміжним пунктом на маршруті. Точка k буде використана, якщо літак не знаходиться в точці з якої починається маршрут. Параметр k буде прирівняний до параметру j, параметр j буде прирівняний параметру i, а параметр i – буде рівний точці в якій літак знаходиться.

Параметри для математичної моделі:

- x_{ij} – маршрут між містами i та j;
- x_{ijk} – маршрут між містами i та j, при якому літак не знаходиться в точці i;
- O – пріоритет маршруту;
- P – прибутковість маршруту;
- H – можливість опрацювати маршрут;

Перенесемо параметри в набір обмежень. Для цього необхідно ввести бінарну змінну y_{ij}, яка приймає значення 1, якщо маршрут пройдено, і 0 відповідно, якщо маршрут не був пройдений.

$$\sum_{i,j} O_{ij} * y_{i,j} = 4; \quad (2.1)$$

$$y_{ij} \geq y_{lm}; \quad (2.2)$$

$$H_{ij} = 1. \quad (2.3)$$

для всіх *i,j,k,l,* де $O_{ij}<O_{lm}$.

Обмеження (2.1 – 2.3), означає суму пріоритетів маршрутів, які були виконані. Якщо ця сума дорівнює 4, то це вказує на те, що виконані маршрути мають в сумі пріоритет 4, тобто вони відносяться до маршрутів верхнього рівня пріоритету. Це використовується як умова для забезпечення виконання маршрутів верхнього рівня пріоритетності в рамках оптимізаційного моделювання. Після проходження всіх маршрутів *critical* рівня, обмеження набуває значення 3. Значення цього обмеження поступово зменшується, доки не буде виконано маршрути зі всіх рівнів пріоритетів. Обмеження (2.2) гарантує, що маршрут від l до m, який означає нижчий рівень пріоритетності, не може бути виконаний, поки не буде виконаний маршрут від *i* до *j*, який є верхнім рівнем пріоритетності. Забезпечивши ці обмеження, можливо ефективно моделювати виконання маршрутів з різними рівнями пріоритетності. Обмеження (2.3) означає, що літак може опрацювати маршрут.

$$\sum_{i,j} P_{ij} * y_{i,j}; \quad (2.4)$$

$$\sum_{i,j} t_{ij} * x_{i,j}; \quad (2.5)$$

Сформуємо цільові функції, які будемо мінімізувати або максимізувати. Цільова функція (2.4), означає що прибуток від маршруту *ij* має бути максимальним, задача роботи – максимізувати її. Цільова функція (2.5) має бути мінімізована, і означає, що всі маршрути мають бути пройдені за мінімальну кількість часу, з урахуванням накладених обмежень на вибір маршрутів.

На основі всіх наведених вище даних, формується оптимізаційна задача, рішення якої буде знаходити програмний модуль. Ця оптимізаційна задача включає максимізацію прибутковості маршрутів і мінімізацію часу польоту, з урахуванням пріоритетності маршрутів, можливості опрацювання маршрутів літаком, а також витрат на пальне та обслуговування. Залежно від різних стартових наборів даних, рішення будуть відрізнятись, однак всі вони мусять задовольняти систему.

2.3. Розробка алгоритму оптимізації

У цьому розділі увага зосереджується на процесі створення ефективного алгоритму, для вирішення поставленої задачі оптимізації маршрутів та підвищення ефективності використання літаків авіакомпанією. На основі створеної математичної моделі, необхідно визначити пріоритетність обмежень, перенести їх в програмний код, а також визначити алгоритм роботи керування програмним модулем. Цей розділ є критичним етапом в розробці програмного модулю, оскільки ефективність та точність алгоритму визначатимуть успішність розв'язання оптимізаційної задачі.

Розглянемо опис розробленого алгоритму, включаючи кроки його виконання, основні критерії вибору маршрутів та механізми прийняття рішень. Особлива увага буде приділена методам оптимізації роботи алгоритму для досягнення високої продуктивності та точності.

У програмному модулі, варто розглянути два основних сценарії: до запуску симуляції та після.

На етапі запуску симуляції, в додаток вже введені початкові дані про аеропорти, літаки, маршрути, та обрано, чи використовується алгоритм оптимізації. При цьому сама симуляція почне працювати, лише після натискання кнопки пуск, яка перевірить стан перемикача використання алгоритму, та на основі його значення запустить конкретний ланцюг функцій. Програма пройде по списку літаків і вибере перший підходящий маршрут, якщо він існує, для кожного з них. Оскільки маршрут може набувати значення *null*, відсутність маршруту сприймається за задовільний результат, який не викликає збоїв в роботі додатку. Цей алгоритм відображено на рис. 2.1.

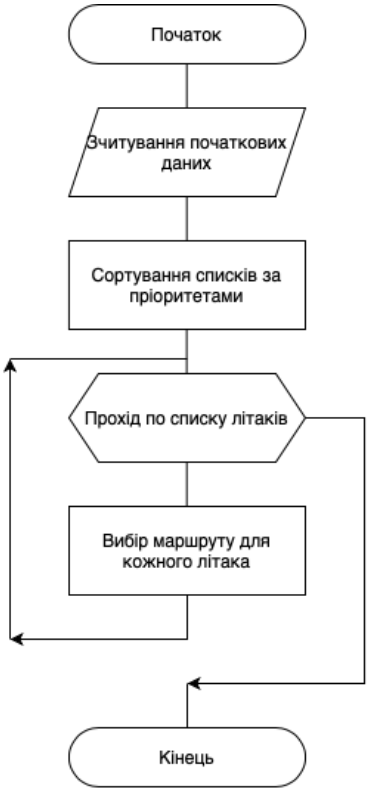

Рис. 2.1. Схема алгоритму вибору початкових маршрутів для літаків

В процесі симуляції, літаки проходять маршрути, і обирають наступний, коли потрапляють в кінцеву точку маршруту. Ця логіка відображена на рис 2.2.

Рис. 2.2. Схема алгоритму вибору нового маршруту літаком

Вибір оптимального маршруту підпорядковується наведеним в попередньому розділі обмеженням та правилам, і проілюстрований на рис. 2.3 та рис 2.4. На рис. 2.3 відображено яким чином, проходимо по списку необхідних маршрутів.

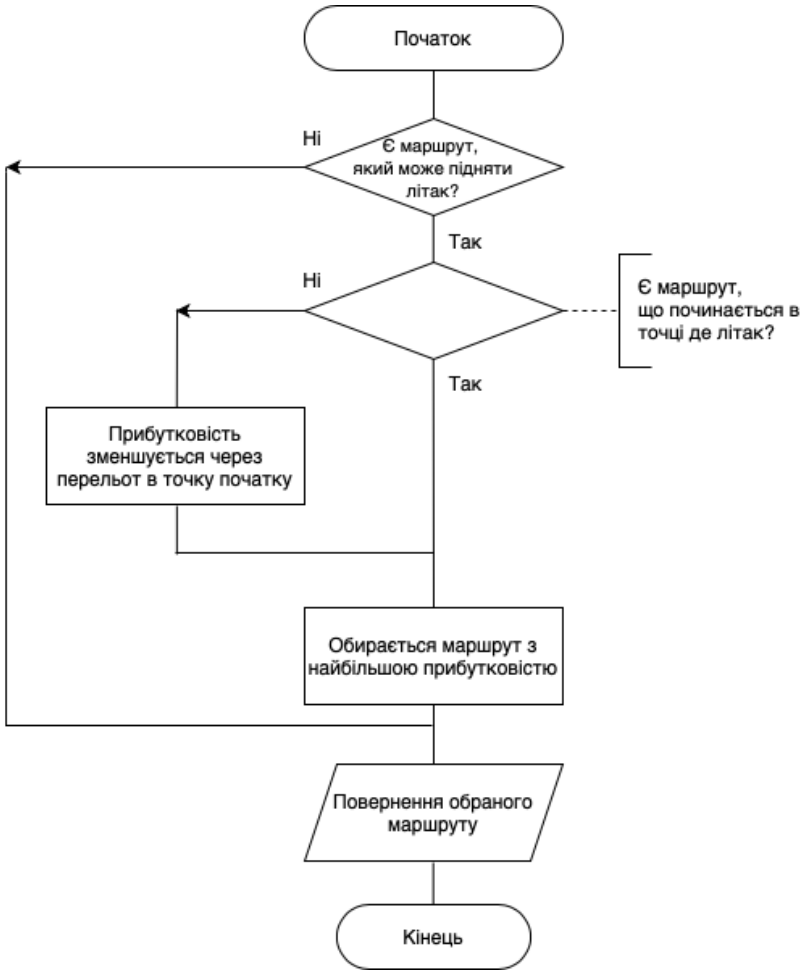

Рис. 2.3. Схема алгоритму вибору маршруту, що підходить найбільше

На рис. 2.4 проілюстровано, як саме визначається, чи підходить літаку конкретний маршрут.

Детальніше алгоритм роботи програмного модуля описано в розділі 3, під час реалізації алгоритму у програмному коді.

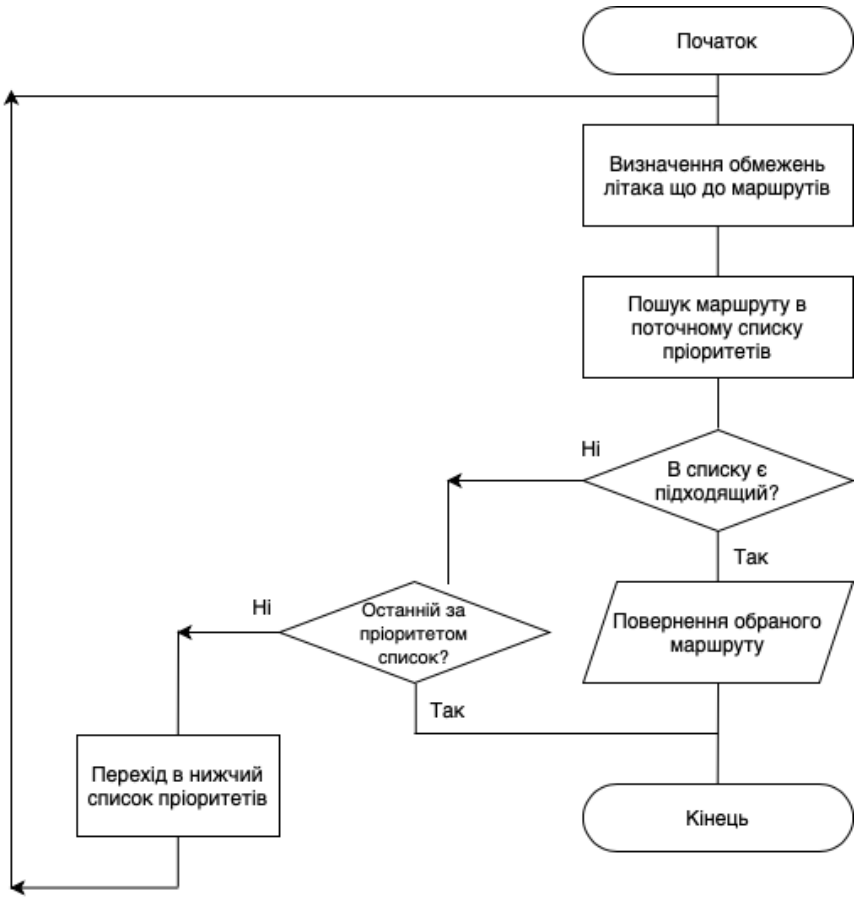

Рис. 2.4. Схема алгоритму вибору нового маршруту

Розбиття глобального алгоритму на алгоритми роботи кожного підмодулю, полегшує подальшу розробку програмного модуля, та дозволяє краще зрозуміти загальну ідею.

2.4. Висновки до розділу

У другому розділі було досліджено принципи, підходи та методи, спрямовані на розробку оптимізаційного алгоритму для вирішення завдання оптимізації маршрутів та підвищення ефективності використання літаків авіакомпанією. Основна увага була сконцентрована на визначенні початкових умов для маршрутів та створенні математичної моделі, щоб відобразити основні аспекти оптимізації. Зазначено, що розроблений алгоритм має сприяти ефективній роботі авіакомпанії, оптимізуючи вибір маршрутів та використання літаків. При цьому прибутковість не

зводиться в абсолют, а система покладається на інші ключові фактори, для підвищення реалістичності системи.

Важливим етапом стала розробка математичної моделі, яка враховує об'єктиви мінімізації і максимізації та обмежень, що виникають з особливостей завдання. Уточнення функції для мінімізації, таких як час польотів, відображено як спроба покращити ефективність системи перевезень. Максимізація функції прибутковості, дозволить підвищити коефіцієнт ефективності використання літаків авіакомпанією. Алгоритм оптимізації був ретельно розроблений та розділений на алгоритми кожного із підмодулів для полегшення впровадження та забезпечення модульності усього програмного модулю.

Результатом цього розділу є створення основи для подальшого вдосконалення та використання розробленого алгоритму в практичних умовах авіакомпанії. Проведені аналіз та розробка є важливим етапом в дослідженні, що сприятиме подальшій оптимізації та вдосконаленню системи управління маршрутами авіакомпанії. На основі цього розділу будуються наступні етапи дослідження та розробки, де будуть проведені тестування розробленого алгоритму та визначені його переваги та недоліки.

РОЗДІЛ 3
ПРОГРАМНИЙ МОДУЛЬ ГЕНЕРАЦІЇ ТЕСТОВИХ ДАНИХ

У даному розділі вивчається важливий аспект розробки програмного модуля, спрямованого на оптимізацію маршрутів та підвищення ефективності використання літаків у діяльності авіакомпанії. Проєктування цього модуля вимагає глибокого аналізу та зосередження на кількох ключових аспектах, серед яких варто виділити архітектуру програми, функціональні вимоги, впровадження алгоритму оптимізації, додатковий функціонал та графічну реалізацію.

Під час проектування архітектури враховувалися вимоги до ефективності та масштабованості, з метою забезпечення оптимального функціонування програмного продукту в умовах реального використання. Функціональні вимоги детально розкривалися для забезпечення повноцінного функціоналу програмного модуля, що відповідає потребам авіакомпанії.

Реалізація алгоритму оптимізації стала ключовим етапом, де важливим було вибрати оптимальний підхід та забезпечити його ефективну інтеграцію у програмний модуль. Додатковий функціонал впроваджувався з метою покращення користувацького досвіду та розширення можливостей програми.

Графічна реалізація програмного модуля була уважно розроблена, здійснюючи виважений підхід до використання графічних елементів та забезпечуючи зручність взаємодії для кінцевого користувача.

3.1. Опис архітектури та функціональних вимог до програмного модуля

Основним архітектурним рішенням для додатку було обрано архітектуру *MVC*, яка є популярним підходом у розробці програмного забезпечення, який спрощує управління кодом та полегшує розподіл обов'язків у програмному продукті. У випадку даного програмного модуля, вибір архітектури *MVC* свідчить про важливий підхід до організації коду та структури програми.

Основні складові архітектури *MVC* включають:

– *Model*: представлення даних та бізнес-логіки програми. Модель відповідає за обробку та зберігання даних, а також за виконання основних операцій;

– *View*: відображення інформації користувачу та відповідає за візуальне подання даних. Вид отримує інформацію від моделі та відображає її користувачеві;

– *Controller*: контролер відповідає за обробку введення користувача, взаємодію з моделлю та оновлення вигляду. Він служить посередником між моделлю та виглядом.

За допомогою архітектури *MVC* програмний модуль буде добре організованим та легко розширюваним. Модель відокремлена від представлення, що дозволяє робити зміни в інтерфейсі користувача без впливу на бізнес-логіку, і навпаки. Контролер відповідає за взаємодію між моделлю та видом, а їх розділення полегшує тестування та підтримку коду.

Загалом, використання архітектури *MVC* сприяє покращенню читабельності коду, його модульності та загальної ефективності програмного модуля.

Розроблений програмний модуль має на меті не лише надавати зручний інтерфейс для визначення маршрутів, але й автоматизувати процес їх оптимізації. Використання локальної бази даних дозволяє зберігати та обробляти інформацію на пристрої користувача, забезпечуючи швидший доступ та можливість роботи в автономному режимі. Одночасно хмарна база даних гарантує їх доступність з будь-якого пристрою та можливість автоматичної синхронізації, що є важливим аспектом для забезпечення єдиної точки доступу та оновлення даних у реальному часі.

Процес синхронізації даних включає в себе автоматичну взаємодію між пристроями користувача та базами даних, забезпечуючи єдність інформації. Це важливо для уникнення розбіжностей між різними версіями даних та підтримання актуальності інформації. Застосування сучасних методів синхронізації забезпечує надійність та ефективність цього процесу.

Використання шифрування даних є ключовим елементом для забезпечення конфіденційності та безпеки інформації. Шифрування застосовується до всіх чутливих даних, включаючи маршрути, особисті дані та фінансову інформацію. Це

перешкоджає несанкціонованому доступу та гарантує, що лише авторизовані користувачі мають доступ до конфіденційної інформації.

Можливість працювати в автономному режимі та мати доступ до даних через хмару розширює функціональність програмного модуля. Це особливо корисно в умовах обмеженого часу, коли користувачі можуть вносити зміни у свої дані навіть поза зонами покриття мережі Інтернет.

Впровадження системи моніторингу та журналювання подій є важливим етапом для забезпечення безпеки. Система автоматично виявляє та реєструє події, такі як невдачі входу, зміни в правах доступу чи вразливості в системі. Це допомагає оперативно виявляти та вирішувати проблеми безпеки, забезпечуючи стабільну та надійну роботу програмного модуля.

Швидкодія є визначальною характеристикою програмного модуля для оптимізації маршрутів та управління авіакомпанією. Швидка реакція на запити користувачів та оптимізована обробка даних грають ключову роль у забезпеченні ефективності та зручності використання. Потужні алгоритми оптимізації маршрутів мають працювати швидко та ефективно, дозволяючи користувачам отримати оптимальні рішення у найкоротший термін.

Багатоплатформенність є необхідною умовою для максимальної досяжності аудиторії. Розробка додатку, який працює на платформах *Android, iOS, macOS* та *Windows*, гарантує, що користувачі з різних пристроїв та операційних систем матимуть доступ до всіх функціональних можливостей. Це важливо для забезпечення універсальності та комфорту використання, дозволяючи користувачам вибирати пристрій за їхніми власними вподобаннями та потребами. Такий підхід до розробки не лише сприяє широкому розповсюдженню програмного модуля, але і робить його доступним для різних груп користувачів, незалежно від їхньої технічної оснащеності та обраної платформи. Це дозволяє забезпечити максимальну користь від використання додатку працівниками авіакомпаній.

3.2. Технологія роботи програмного модуля генерації тестових даних

Однією з ключових особливостей є використання шаблону *Singleton* для створення єдиного екземпляру класу *AlgorithmUtil*. Це гарантує, що весь функціонал, пов'язаний з оптимізацією маршрутів, доступний через єдину точку управління, сприяючи покращенню керованості та уникненню конфліктів.

Модуль широко використовує принцип поліморфізму та інкапсуляції, використовуючи приватні методи та класи для зберігання списків аеропортів, літаків та інших ресурсів. Це дозволяє підтримувати внутрішню структуру даних відокремленою від зовнішнього доступу, забезпечуючи конфіденційність та безпеку.

Ключова роль у модулі відводиться методам *generateStartRoutes* та *generateStartRoutesWithoutAlgorithm*, які відповідають за початкову генерацію маршрутів. Вони враховують пріоритети та умови для вибору оптимальних маршрутів для літаків, використовуючи різні списки пріоритетів. При чому один з них повертає маршрути з урахуванням функцій оптимізації, а інший без. Функція *generateStartRoutes* є верхньорівневою функцією з генерацією початкових маршрутів для літаків у контексті оптимізації авіаперевезень. Розглянемо в деталях, як вона виконує цю задачу:

– підготовка даних: функція отримує на вхід списки аеропортів, ресурсів транспортування та літаків;

– організація даних за пріоритетом: розділення ресурсів транспортування на чотири категорії за пріоритетом: *critical*, *high*, *mid*, та *low*. Це полегшує вибір та оптимізацію маршрутів враховуючи їхні пріоритети;

– генерація маршрутів для літаків: для кожного літака відбувається вибір наступного маршруту за допомогою методу *getNextRoute*. Залежно від пріоритету, використовується відповідний список ресурсів для вибору маршруту;

– додавання та оновлення інформації: визначений наступний маршрут додається до списку згенерованих маршрутів *generatedRoutes*. Літак також оновлюється, додаючи новий маршрут до свого списку та літак додається до списку активних літаків;

– видалення використаних ресурсів: згенерований маршрут видаляється зі списку ресурсів, таким чином вони більше не беруть участь в подальших розрахунках.

Лістинг коду функції *generateStartRoutes*:

```
Void generateStartRoutes(
  List<Airport> airports,
  List<TransportationResource> transportationResources,
  List<Aircraft> planes,
) {
  _airports.addAll(airports);
  _criticalPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.critical));
  _highPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.high));
  _midPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.mid));
  _lowPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.low));
  planes.forEach((plane) {
    AircraftRoute? nextRoute = getNextRoute(plane);
    if (nextRoute != null) {
      removeFromList(nextRoute);
      _generatedRoutes.add(getAdditionalTransitionIfNeeded(plane, nextRoute));
      plane.aircraftRoutes
          .add(getAdditionalTransitionIfNeeded(plane, nextRoute));
      _aircrafts.add(plane);
    }});
}
```

Загальною метою функції є ефективне розподілення ресурсів для максимізації прибутку авіакомпанії та оптимізації маршрутів для літаків, забезпечуючи врахування пріоритетів і умов використання ресурсів. Особливу увагу слід звернути на методи, що визначають та обробляють пріоритети маршрутів. Це реалізовано шляхом використання методів, таких як *sortByPriority* та *getProfit*, що забезпечують відповідне сортування та обчислення прибутку.

Функція *getNextRoute* відповідає за вибір наступного маршруту для конкретного літака враховуючи різні пріоритети ресурсів та розташування літака на момент вибору. Розглянемо основні етапи цієї функції:

– визначення початкової позиції літака: визначається позиція, з якої літак розпочинає свій маршрут. Це може бути базова позиція літака чи кінцева точка останнього маршруту;

– вибір за пріоритетом: вибір ресурсів транспортування для вибору наступного маршруту залежно від пріоритету ресурсів *critical*, *high*, *mid*, та *low*;

– перевірка сумісності: перевірка сумісності доступних ресурсів з літаком за обсягом перевезення;

– оновлення списків: після вибору наступного маршруту, відповідний ресурс транспортування видаляється зі списку, щоб уникнути його повторного використання.

– повернення результату: повертає об'єкт *aircraftroute*, який представляє собою вибраний маршрут для літака.

Важливо зауважити, що функція *getNextRoute* може повернути *null*, це означає що для літака не лишилось маршрутів які він може взяти. Тим не менш, функція *getNextRoute* дозволяє системі ефективно вибирати наступні маршрути для літаків враховуючи пріоритети та умови використання ресурсів, що допомагає досягти оптимізації авіаперевезень.

Лістинг коду функції *getNextRoute*:

```
AircraftRoute? getNextRoute(Aircraft plane) {
  AirportPosition routeStartPosition =
      plane.baseAircraftPosition.airportPosition;
```

```
if (_criticalPriorityRoutes.isNotEmpty) {
  try {
    return getAircraftRouteFromTransportationResource(
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _criticalPriorityRoutes));
  } catch (e) {
    print("No suitable objects in _criticalPriorityRoutes list");
  }
} else if (_highPriorityRoutes.isNotEmpty) {
  try {
    return getAircraftRouteFromTransportationResource(
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _highPriorityRoutes));
  } catch (e) {
    print("No suitable objects in _highPriorityRoutes list");
  }
} else if (_midPriorityRoutes.isNotEmpty) {
  try {
    return getAircraftRouteFromTransportationResource(
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _midPriorityRoutes));
  } catch (e) {
    print("No suitable objects in _midPriorityRoutes list");
  }
} else if (_lowPriorityRoutes.isNotEmpty) {
  try {
    return getAircraftRouteFromTransportationResource(
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _lowPriorityRoutes));
  } catch (e) {
```

```
    print("No suitable objects in _lowPriorityRoutes list");
  }}
 return null;
}
```

Ця функція містить в собі функції нижнього рівня: *getFirstSuitableTransportation* та *getAircraftRouteFromTransportationResource*, задачі яких знайти перший підходящий маршрут зі списку пріоритету, та перетворити його з формату *TransportationResource* в *AircraftRoute*.

Лістинг коду функції *getFirstSuitableTransportation*:

```
TransportationResource getFirstSuitableTransportation(
    AirportPosition routeStartPosition,
    int planeSpaceAmount,
    List<TransportationResource> prioritizedList) =>
 prioritizedList.firstWhere(
    (element) =>
      element.startPoint.airportPosition == routeStartPosition &&
      didPlaneCanPickup(planeSpaceAmount, element.amount),
    orElse: () => prioritizedList.firstWhere((element) =>
      didPlaneCanPickup(planeSpaceAmount, element.amount)));
```

Лістинг коду функції *getAircraftRouteFromTransportationResource*:

```
AircraftRoute getAircraftRouteFromTransportationResource(
  TransportationResource transportation) {
 return AircraftRoute(
  name:
    "Route From${transportation.startPoint.name}to${transportation.endPoint.name}",
  startPoint: transportation.startPoint,
  endPoint: transportation.endPoint,
  routeProfit: transportation.costPerUnit * transportation.amount,
  routePriority: transportation.priority,
 );
```

При цьому низькорівнева функція *getFirstSuitableTransportation*, виконує перевірки чи є маршрут, який може почати літак, з точки в якій знаходиться літак. Якщо маршруту з точки, в якій знаходиться літак, то шукається перший маршрут який може бути виконаний. Оскільки маршрути відсортовані за пріоритетністю та прибутковістю – такий маршрут буде найбільш оптимальним. Важливо зазначити, що модуль використовує мову програмування *Dart* останньої версії та дотримується сучасних практик програмування, що сприяє зручності управління та розвитку коду.

3.3. Розробка додаткового функціоналу програмного модуля

В розвитку додаткового функціоналу було реалізовано три ключові складові: шифрування даних, використання локальної та хмарної баз даних, а також формування звітностей.

Для забезпечення безпеки конфіденційної інформації використовуються два сучасних шифрувальних алгоритми: *ECC* та *AES*.

ECC відповідає за генерацію ключів: для кожного користувача генерується унікальна пара ключів - приватний та публічний ключі, використовуючи еліптичні криві. Також еліптична крива дозволяє ефективно використовувати алгоритми підпису та шифрування, забезпечуючи високий рівень захисту.

AES використовує симетричний метод шифрування, де один і той же ключ використовується для як шифрування, так і дешифрування. Це забезпечує ефективність та швидкість обробки даних. Застосування *AES* забезпечує надійний рівень захисту конфіденційної інформації в додатку.

Для безперебійної роботи додатку реалізовано дві БД: локальна та хмарна. Дані з локальної БД потрапляють в хмарну з певною періодичністю, і зберігаються там лише для формування довготривалих статистичних даних, розвантажуючи таким чином додаток. Синхронізація даних між локальною та хмарною базами даних реалізована з використанням технологій *Firebase* та *Hive*.

Хмарна БД *Firebase*: *Firebase Realtime Database* дозволяє автоматично синхронізувати дані в реальному часі, надаючи однаковий стан даних на всіх

пристроях. Кожна зміна в базі даних автоматично синхронізується з іншими підключеними пристроями, що гарантує однорідність даних.

Локальна БД *Hive* дозволяє зручно зберігати дані локально на пристрої користувача, полегшуючи доступ до інформації без необхідності постійного Інтернет-з'єднання. Користувач може працювати з додатком в автономному режимі, а підключившись до мережі, дані будуть синхронізовані з хмарною базою даних.

Формування звітностей в додатку реалізовано для надання користувачам інформативних даних про оптимізацію маршрутів та використання ресурсів. Для формування звітностей використовуються дані про здобуті маршрути, витрати, прибуток та інші ключові показники, які систематично збираються та аналізуються. Інформація відображається у вигляді графіків, діаграм та інших інтерактивних засобів візуалізації, що робить звітні дані зрозумілими та доступними для аналізу. Користувачі можуть експортувати або друкувати звіти для подальшого використання або обміну інформацією. Такий підхід до формування звітностей не лише надає користувачам повний обсяг аналізу даних, але й полегшує процес прийняття рішень та вдосконалення стратегій авіаперевезень.

3.4. Графічна реалізація програмного модуля

Підготовка до графічної реалізації програмного модуля з використанням фреймворку *Flutter* включає кілька ключових кроків. Було ретельно вивчено вимоги до графічного представлення програмного модуля. Визначено, які функції повинен виконувати інтерфейс, і які дані відображатимуться на екрані. Функції було описано в попередніх розділах, за основу можна взяти інтерфейси конкурентних програм. Для розробки було обрано фреймворк *Flutter* у комбінації з мовою програмування *Dart* для створення графічної частини [21]. Це обумовлено тим, що такий додаток має бути доступним на максимальній кількості платформ, а найбільш популярним на сьогодні багатоплатформним рішенням є *Flutter* [22].

Для створення дизайну графічного інтерфейсу, було використано інструменти для проектування інтерфейсу, такі як *Figma* або *Adobe XD*. При розробці графічного

інтерфейсу враховуються вимоги користувача та використовуються стандарти *Flutter* щодо розміщення елементів.

Починаючи розробку, створено базову структуру проекту *Flutter*. Визначено екрани, віджети та папки для кожної частини програми. Відбувається розробка власних віджетів, які можна повторно використовувати в різних місцях програми. Використовуються вбудовані та сторонні *Flutter* віджети для швидкого створення елементів інтерфейсу [23]. Це може включати кнопки, тексти, списки, карти, тощо. Забезпечується взаємодія графічного інтерфейсу з даними, що надходять з іншої частини програми. Використовується підхід *bloc,* та пакет *Provider* для керування станом додатку [24]. Застосовуються анімації та ефекти переходів для покращення взаємодії користувача. *Flutter* надає багато можливостей для створення плавних анімацій.

Далі розробляється адаптація до різних платформ, в ході якої перевіряється, що графічна реалізація працює коректно на різних платформах, особливо на *Windows* та *MacOS*. Це відбувається з використанням адаптивного дизайну, аби додатком можна було скористатись і на мобільних платформах. Графічний інтерфейс тестується на різних етапах розробки, виправляються помилки та недоліки, щоб забезпечити правильну роботу програми [25]. Створюється документація для графічної реалізації, включаючи опис функцій, використані технології та скріншоти інтерфейсу. У графічній реалізації існує кілька пунктів, розробка яких заслуговує більш детального обговорення. У основі зовнішнього вигляду, лежить макет схожий на сервіс *FlightRadar*, з зеленою супутниковою мапою на фоні та жовтими літаками для гарної видимості, зображений на рис. 3.1.

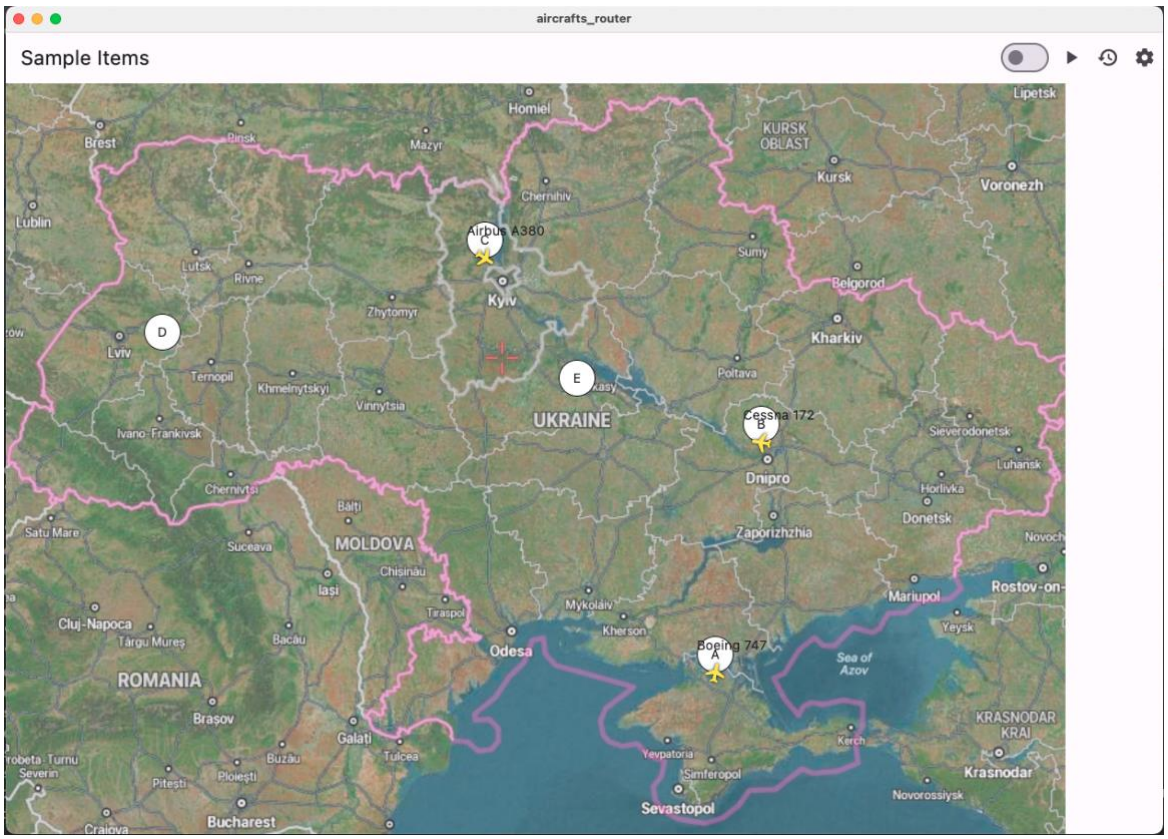

Рис. 3.1. Головне вікно додатку

Аеропорти представлені у вигляді точок для зручності. Клас аеропорт реалізований наступним чином:

```
        class Airport {
  Airport({
    required this.name,
    required this.airportPosition,
    required this.fuelAmount,
    required this.totalAircraftAmount,
  });
  final String name;
  ///Airport position on a flat surface (x, y)
  final AirportPosition airportPosition;
  double fuelAmount;
  int totalAircraftAmount;
```

```
  int currentAircraftAmount = 0;
}
```

При цьому клас *AirportPosition*, приймає в себе координати *x* в діапазоні від 0 до 30 та у від 0 до 12. Ці координати трансформують в відстань на екрані, множачи координату на 50. Це накладає певні обмеження щодо відображення, оскільки в настільних додатках можна змінити розмір вікна, а ця формула не має динамічного змінення. Це можна виправити, використовуючи дані з *MediaQuery.of(context)*, однак вставляти такі обчислення на рівень бізнес логіки є поганою практикою. Цей недолік не є суттєвим, але розмір вікна додатку не бажано змінювати.

На рис. 3.2 зображено екран модифікації даних про літаки. Ця функція є ключовою, адже залежно від змін в цих моделях даних, можливі зміни в проходженні маршрутів. Перехід на вікно з налаштуваннями відбувається при натисканні кнопки в правому куті екрану. Ця кнопка доступна на протязі всієї роботи додатку, однак дані про ці зміни сприймаються лише при запуску симуляції, тобто зміна даних літаків під час польотів ні на що не вплине.

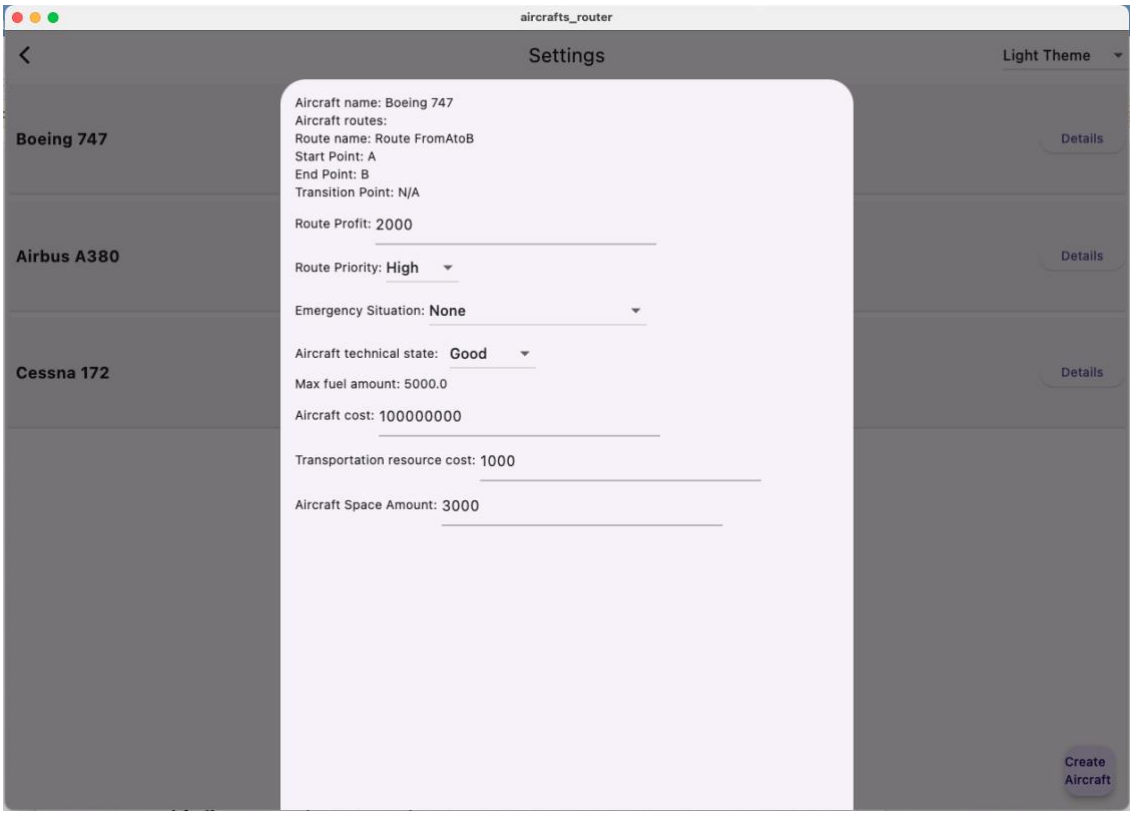

Рис. 3.2. Модифікація даних літаків

Не менш важливою є реалізація іконок літаків на маршрутах. На рис. 1 видно, що літаки націлені на аеропорт, в який вони будуть летіти. Це реалізовано як статична іконка, розміщена в поточній позиції літака, обгорнута віджетом *Transform.rotate,* який розвертає її в потрібному напрямку.

Лістинг коду іконки літака:

```
return Positioned(
  left: currentPosition.dx,
  top: currentPosition.dy,
  child: Transform.rotate(
    angle: rotationAngle,
    child: IconButton(
      onPressed: () =>
          context.read<SelectedItemCubit>().selectAircraft(widget.aircraft),
      icon: const Icon(
        Icons.airplanemode_active,
        color: Colors.yellow,
        shadows: [Shadow(blurRadius: 2)],
      ), ), ),);
```

Для того щоб визначити кут, на який треба повернутись літаку, використовується функція *calculateRotationAngle()*. Змінні *dx* та *dy* представляють різницю між відповідними координатами цільової і поточної позицій. Ці значення визначають відстань по горизонталі та вертикалі між двома точками.

Використовуючи *math.atan2(dy, dx)*, функція обчислює арктангенс відношення *dy* до *dx*. Це дозволяє визначити кут між вектором, спрямованим від поточної позиції до цільової. Кут, отриманий з арктангенсу, керується властивостями тригонометричного кола, тому його необхідно коригувати для відображення правильного кута повороту. Додається *math.pi* / 2, і результуючий кут відображає поворот за годинниковою стрілкою, де *math.pi* – значеня π, з стандартної бібліотеки *math*.

Перевіряється, чи отриманий кут менший за нуль. Якщо так, то до нього додається 2 * *math.pi*. Це робиться для нормалізації кута, забезпечуючи, що він завжди буде в межах від 0 до 2 * *math.pi*. Остаточний обчислений кут повороту повертається як результат функції.

Лістинг коду функції для знаходження напрямку літака:

```
double calculateRotationAngle() {
  final dx = targetPosition.dx - currentPosition.dx;
  final dy = currentPosition.dy - targetPosition.dy;
  double angle = math.atan2(dy, dx);
  angle = -angle + math.pi / 2;
  return angle < 0 ? angle + 2 * math.pi : angle;
}
```

3.5. Висновки до розділу

У цьому розділі розглянуто проектування програмного модуля для оптимізації маршрутів та підвищення ефективності використання літаків авіакомпанією.

Для досягнення ефективності та надійності програмного модуля, була використана архітектура *MVC*. Ця архітектура дозволяє чітко визначити ролі різних компонентів, забезпечуючи модульність та розширюваність.

Функціональні вимоги включають в себе здійснення оптимізації маршрутів для авіакомпанії, використовуючи дані з локальної та хмарної баз даних. Синхронізація даних, застосування шифрування для захисту конфіденційності, а також можливість роботи в автономному режимі та моніторинг подій для виявлення проблем безпеки також є важливими частинами функціональних вимог.

Алгоритм оптимізації було ретельно розроблено та реалізовано з використанням мови програмування та підходів, які найкраще відповідають поставленим завданням. Процес реалізації включав в себе визначення початкових умов маршрутів, розробку математичної моделі, а також деталізацію алгоритму на дрібні частини для забезпечення ефективності та чіткості коду.

Додатковий функціонал включає в себе використання локальної та хмарної баз даних, що дозволяє зберігати та отримувати дані в режимі реального часу. Також, шифрування даних забезпечує конфіденційність та безпеку інформації, а реалізація моніторингу та журналювання подій дозволяє вчасно виявляти та вирішувати проблеми безпеки.

Графічна реалізація програмного модуля використовує фреймворк *Flutter* для створення інтуїтивно зрозумілого та естетичного інтерфейсу користувача. Взаємодія з користувачем оптимізована для зручного використання функцій модуля, забезпечуючи при цьому зручність та ефективність роботи.

В цілому, проектування програмного модуля було виконано з урахуванням найкращих практик розробки програмного забезпечення, а реалізація включає в себе високоякісний код, гнучкість та можливість масштабування для подальшого розвитку.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1. Проведення експериментів для тестування та оцінки ефективності розробленого програмного модуля

У цьому розділі проведено експерименти для тестування та оцінки ефективності розробленого програмного модуля, спрямовані на детальний аналіз його функціональності та визначення його продуктивності в різних умовах. Даний розділ включає в себе огляд мети експериментів, методології їх проведення та очікуваних результатів.

Основною метою проведення експериментів є оцінка функціональності та продуктивності розробленого програмного модуля з точки зору його здатності ефективно оптимізувати маршрути в умовах реального використання. Важливо визначити, наскільки добре модуль впорався із завданнями оптимізації та чи відповідає вимогам авіакомпаній.

Експерименти будуть включати в себе використання реальних даних та сценаріїв, що відображають різні ситуації в авіаційній сфері. Для оцінки продуктивності та швидкості оптимізації будуть вимірюватися часові показники роботи модуля при різних обсягах даних та складності маршрутів.

Очікується, що розроблений програмний модуль демонструватиме високу ефективність у вирішенні завдань оптимізації маршрутів. Результати експериментів надають об'єктивну оцінку функціональних можливостей модуля та допомогають виявити можливі області для подальшого вдосконалення.

Кількість тестів повинна бути достатньою для вичерпного покриття функціональності та характеристик модуля. Пропонується провести не менше 5 різних тестових сценаріїв, які включають в себе різноманітні умови використання та вхідні дані. Кількість тестів продуктивності може бути збільшена для більш точного визначення меж ефективності модуля. Було пропрацьовано наступні тестові сценарії.

Функціональне тестування сценарій 1: Оптимізація маршрутів. Проведення тестів на основні функції модуля, перевірка його здатності ефективно оптимізувати маршрути відповідно до вхідних даних. Стартовий набір даних відображено в табл. 4.1.

При виконанні такого тестування було отримано наступний результат: без використання алгоритму оптимізації час проходження маршрутів по черзі зайняв 21 секунду. Проходячи відсортовані маршрути за алгоритмом – 16.5 секунд. Таким чином було зафіксовано покращення часу за який проходяться маршрути, рівне 22%.

Таблиця 4.1

Список початкових умов для тестового сценарію №1

Літаки	**Аеропорти**	**Перевезення**
name: "*Boeing* 747", *baseAircraftPosition*: *airports*[0], *transportSpaceAmount*: 300,	*name*: "*A*", *airportPosition*: *AirportPosition*(15, 12), *fuelAmount*: 40, *totalAircraftAmount*: 3,	*startPoint*: *airports*[0], *endPoint*: *airports*[1], *amount*: 200,*costPerUnit*: 10, *priority*: *Priority.high*,
name: "*Airbus A*380", *baseAircraftPosition*: *airports*[2], *transportSpaceAmount*: 400,	*name*: "*B*", *airportPosition*: *AirportPosition*(16, 7), *fuelAmount*: 40, *totalAircraftAmount*: 2,	*startPoint*: *airports*[1], *endPoint*: *airports*[2], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
name: "*Cessna* 172", *baseAircraftPosition*: *airports*[1], transportSpaceAmount: 20000,	*name*: "*C*", *airportPosition*: *AirportPosition*(10, 3), *fuelAmount*: 40, *totalAircraftAmount*: 1,	*startPoint*: *airports*[3], *endPoint*: *airports*[4], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
	name: "*D*", *airportPosition*: *AirportPosition*(3, 5),	*startPoint*: *airports*[4], *endPoint*: *airports*[2],

Закінчення таблиці 4.1

1	**2**	**3**
	fuelAmount: 40, *totalAircraftAmount*: 3	*amount*:100, *costPerUnit*:100, *priority*: *Priority.high*,
	name: "*E*", *airportPosition*: *AirportPosition*(12, 6), *fuelAmount*: 40, *totalAircraftAmount*: 2,	*startPoint*: *airports*[2], *endPoint*: *airports*[1], *amount*: 50, *costPerUnit*: 100, *priority*: *Priority.critical*,
		startPoint: *airports*[0], *endPoint*: *airports*[4], *amount*: 300, *costPerUnit*: 10, *priority*: *Priority.mid*,
		startPoint: *airports*[0], *endPoint*: *airports*[3], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
		startPoint: *airports*[3], *endPoint*: *airports*[0], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
		startPoint: *airports*[2], *endPoint*: *airports*[3], *amount*: 200, *costPerUnit*: 5, *priority*: *Priority.mid*,
		startPoint: *airports*[1], *endPoint*: *airports*[4], *amount*: 400, *costPerUnit*: 1, *priority*: *Priority.mid*,

Тестування продуктивності сценарій 2: вимірювання часових показників оптимізації при різних обсягах вхідних даних. Для цього було додано нові дані, до існуючих в табл 4.1. Список було оновлено даними з табл. 4.2.

При виконанні такого тестування було отримано наступний результат: без використання алгоритму оптимізації час проходження маршрутів по черзі зайняв 47 секунд. Проходячи відсортовані маршрути за алгоритмом – 39 секунд. Таким чином, було зафіксовано покращення часу за який проходяться маршрути, рівне 17%.

Таблиця 4.2

Додатковий список початкових умов для тестового сценарію №2

Аеропорти	**Перевезення**
name: "*F*", *airportPosition*: *AirportPosition*(10, 12), *fuelAmount*: 40, *totalAircraftAmount*: 3,	*startPoint*: *airports*[5], *endPoint*: *airports*[6], *amount*: 200, *costPerUnit*: 10, *priority*: *Priority.high*,
name: "*G*", *airportPosition*: *AirportPosition*(16, 1), *fuelAmount*: 40, *totalAircraftAmount*: 2,	*startPoint*: *airports*[6], *endPoint*: *airports*[7], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
name: "*H*", *airportPosition*: *AirportPosition*(4, 8), *fuelAmount*: 40, *totalAircraftAmount*: 1,	*startPoint*: *airports*[8], *endPoint*: *airports*[9], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
name: "*I*", *airportPosition*: *AirportPosition*(9, 0), *fuelAmount*: 40, *totalAircraftAmount*: 3	*startPoint*: *airports*[9], *endPoint*: *airports*[7], *amount*: 100, *costPerUnit*: 100, *priority*: *Priority.high*,
name: "*J*", *airportPosition*: *AirportPosition*(6, 12),	*startPoint*: *airports*[7], *endPoint*: *airports*[6]

Закінчення таблиці 4.2

1	**2**
fuelAmount: 40, *totalAircraftAmount*: 2,	*amount*: 50, *costPerUnit*: 100, *priority*: *Priority.critical*,
	startPoint: *airports*[5], *endPoint*: *airports*[9], *amount*: 300, *costPerUnit*: 10, *priority*: *Priority.mid*,
	startPoint: *airports*[5], *endPoint*: *airports*[8], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
	startPoint: *airports*[8], *endPoint*: *airports*[5], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
	startPoint: *airports*[7], *endPoint*: *airports*[8], *amount*: 200, *costPerUnit*: 5, *priority*: *Priority.mid*,
	startPoint: *airports*[6], *endPoint*: *airports*[9], *amount*: 400, *costPerUnit*: 1, *priority*: *Priority.mid*,

Тестування продуктивності сценарій 3: оцінка продуктивності при оптимізації складних маршрутів з урахуванням великої кількості зупинок. Для цього було додано нові дані, до існуючих в табл. 4.1 та табл. 4.2. Обидва списки перевезень було модифіковано, в кожен маршрут було додано додаткову пересадку. Оновлені дані

відображені в табл. 4.3, та на основі них проведено два тестування: з перевезеннями з першої колонки та з сумою перевезень з двох колонок.

При виконанні такого тестування було отримано наступний результат: без використання алгоритму оптимізації час проходження 10 маршрутів по черзі зайняв 1 хвилину 54 секунди. Проходячи відсортовані 10 маршрутів за алгоритмом – 1 хвилина 42 секунди. Без використання алгоритму оптимізації час проходження 20 маршрутів по черзі зайняв 3 хвилини 07 секунд. Проходячи відсортовані 20 маршрутів за алгоритмом – 2 хвилини 45 секунд. Таким чином було зафіксовано покращення часу за який проходяться маршрути, рівне 11% при 10 маршрутах та 12% при 20 маршрутах.

Таблиця 4.3

Змінені перевезення з попередніх таблиць

Перевезення	Перевезення
startPoint: *airports*[0], *endPoint*: *airports*[1], *transitionPoint*: *airports*[2], *amount*: 200, *costPerUnit*: 10, *priority*: *Priority.high*,	*startPoint*: *airports*[5], *endPoint*: *airports*[6], *transitionPoint*: *airports*[7], *amount*: 200, *costPerUnit*: 10, *priority*: *Priority.high*,
startPoint: *airports*[1], *endPoint*: *airports*[2], *transitionPoint*: *airports*[3], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,	*startPoint*: *airports*[6], *endPoint*: *airports*[7], *transitionPoint*: *airports*[8], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
startPoint: *airports*[3], *endPoint*: *airports*[4], *transitionPoint*: *airports*[2], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,	*startPoint*: *airports*[8], *endPoint*: *airports*[9], *transitionPoint*: *airports*[7], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
startPoint: *airports*[4], *endPoint*: *airports*[2], *transitionPoint*: *airports*[1], *amount*: 100, *costPerUnit*: 100,	*startPoint*: *airports*[9], *endPoint*: *airports*[7], *transitionPoint*: *airports*[6], *amount*: 100, *costPerUnit*: 100,

Закінчення таблиці 4.3

1	**2**
priority: *Priority.high*,	*priority*: *Priority.high*,
startPoint: *airports*[2], *endPoint*: *airports*[1], *transitionPoint*: *airports*[0], *amount*: 50, *costPerUnit*: 100, *priority*: *Priority.critical*,	*startPoint*: *airports*[7], *endPoint*: *airports*[6], *transitionPoint*: *airports*[5], *amount*: 50, *costPerUnit*: 100, *priority*: *Priority.critical*,
startPoint: *airports*[0], *endPoint*: *airports*[4], *transitionPoint*: *airports*[1], *amount*: 300, *costPerUnit*: 10, *priority*: *Priority.mid*,	*startPoint*: *airports*[5],*endPoint*: *airports*[9], *transitionPoint*: *airports*[6], *amount*: 300, *costPerUnit*: 10, *priority*: *Priority.mid*,
startPoint: *airports*[0], *endPoint*: *airports*[3], *transitionPoint*: *airports*[4], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,	*startPoint*: *airports*[5], *endPoint*: *airports*[8], *transitionPoint*: *airports*[9], *amount*: 400, *costPerUnit*: 10, *priority*: *Priority.mid*,
startPoint: *airports*[3], *endPoint*: *airports*[0], *transitionPoint*: *airports*[1], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,	*startPoint*: *airports*[8], *endPoint*: *airports*[5], *transitionPoint*: *airports*[6], *amount*: 300, *costPerUnit*: 5, *priority*: *Priority.mid*,
startPoint: *airports*[2], *endPoint*: *airports*[3], *transitionPoint*: *airports*[4], *amount*: 200, *costPerUnit*: 5, *priority*: *Priority.mid*,	*startPoint*: *airports*[7], *endPoint*: *airports*[8], *transitionPoint*: *airports*[9], *amount*: 200, *costPerUnit*: 5, *priority*: *Priority.mid*,
startPoint: *airports*[1], *endPoint*: *airports*[4], *transitionPoint*: *airports*[0], *amount*: 400, *costPerUnit*: 1, *priority*: *Priority.mid*,	*startPoint*: *airports*[6], *endPoint*: *airports*[9], *transitionPoint*: *airports*[5], *amount*: 400, *costPerUnit*: 1, *priority*: *Priority.mid*,

Тестування стабільності та надійності сценарій 4: тестування стійкості модуля до можливих збоїв та його можливості відновлення після відмов. Результати

тестування показали, що модуль не захищений від збоїв. Існує історія вибору маршрутів, однак в разі поломки додатку, продовжити з місця поломки не вийде. Також тестування показало, що маршрути генеруються однаково кожен раз, якщо початкові умови не змінні. Додаток показав досить високий рівень працездатності, однак можливості протестувати його на слабшому апаратному забезпеченні не було.

Тестування у реальних умовах сценарій 5: проведення тестів на основі реальних сценаріїв використання авіакомпаніями, враховуючи різні ситуації та обмеження.

В додатку реалізовано перебудування маршрутів, в разі трьох екстрених ситуацій: поломка літака, погана погода на маршруті та проблема з перевезенням. Поломка літака перебудовує всі маршрути таким чином, аби літак міг сісти в найближчий аеропорт, звільнивши місце для літака, якщо потрібно. Погана погода на маршруті виконує перебудову конкретного маршруту, додаючи в нього проміжну точку, через яку буде пройдено, для уникнення небезпечної точки. Проблема з перевезенням змінює маршрут таким чином, що літак повертається в точку з якої вилетів, чекаючи доки там з’явиться вільне місце.

Ці екстрені ситуації не покривають 100% можливих ситуацій, однак є реальними кейсами, тому вважається що тестування сценарію реального використання пройдено успішно. Варто зауважити, що екстрені ситуації не відносять до оптимізації маршрутів, однак вони є частиною екосистеми авіа компаній, і необхідні в тому числі для тестування.

Розглянувши проведені експерименти та їхні результати, можна зробити кілька висновків щодо ефективності розробленого програмного модуля для оптимізації маршрутів в авіаційній сфері.

У результаті функціонального тестування виявлено значне поліпшення в часі проходження маршрутів, досягнуте завдяки використанню алгоритмів оптимізації. У порівнянні з відсутністю оптимізації, модуль продемонстрував зменшення часу на 22%, що свідчить про його високу ефективність у вирішенні завдань оптимізації маршрутів.

Експерименти з обсягом даних показали, що модуль ефективно справляється з оптимізацією при різних обсягах вхідних даних. Виявлено покращення часу

проходження маршрутів на 17%, що свідчить про стабільну роботу модуля при збільшенні об'єму інформації в два рази. Ефективність є меншою, ніж у результаті тестування з меншою кількістю маршрутів, адже більше часу витрачається на маршрути.

Модуль успішно справляється з оптимізацією складних маршрутів та великою кількістю зупинок. Зафіксовано покращення часу на 12%, що вказує на його здатність ефективно вирішувати завдання в умовах складних та реалістичних сценаріїв. При цьому відсоток ефективності зменшився відносно попереднього тестування, адже кожен літак тепер проходить свій маршрут через додаткову точку, яка може знаходиться не на шляху до кінцевої точки маршруту.

У сценаріях збоїв та відновлення виявив певні недоліки, пов'язані з відсутністю захисту від можливих збоїв. Проте, рівень працездатності в цілому високий, а історія вибору маршрутів та їх генерація в кожному випадку забезпечують стабільність роботи.

Тестування сценаріїв реального використання, які включають екстрені ситуації, підтверджують придатність модуля до практичного використання авіакомпаніями. Його здатність перебудовувати маршрути в разі аварій, погіршення погодних умов та проблем з перевезеннями є важливою функціональністю для реальних сценаріїв використання.

Розроблений програмний модуль є високоефективним інструментом для авіакомпаній, здатним оптимізувати маршрути в реальному часі, враховуючи різноманітні умови та обмеження. Потенційні аспекти для вдосконалення включають захист від можливих збоїв та опрацювання додаткових умов перебудови маршрутів. Не менш важливими кроками є перетворення з симуляції на реальну авіаційну систему, тобто налаштування відслідковування реальної позиції літаків, погоди на маршрутах, тощо. Необхідно також провести додаткові тестування різних сценаріїв роботи, та працездатності на різному апаратному забезпеченні та з різними стартовими наборами.

4.2. Аналіз отриманих результатів та визначення переваг та обмежень запропонованого підходу

Аналізуючи отримані результати в контексті існуючих систем оптимізації маршрутів в авіаційній сфері, можна визначити кілька ключових аспектів, однак порівняти їхню ефективність не вийде, адже такі продукти не надають цієї інформації публічно, а їхні алгоритми є засекреченими.

Розроблений програмний модуль виявився значно часово ефективнішим порівняно з відсутністю оптимізації маршрутів. Зменшення часу проходження 10 маршрутів на 22% свідчить про високу продуктивність системи та її здатність швидко оптимізувати маршрути.

Модуль продемонстрував стабільну роботу при збільшенні обсягу даних у два рази, з покращенням часу на 17%. Це свідчить про його ефективність при оптимізації маршрутів в умовах збільшення обсягів вхідних даних. При подальшому додаванні списку маршрутів, цей відсоток буде зменшуватись, однак в авіаційних системах, кожен відсоток є цінним.

Модуль зберіг високий рівень ефективності при оптимізації складних маршрутів та зупинок, навіть з покращенням часу на 12%. Це вказує на його здатність ефективно оптимізувати маршрути в реальних та варіативних сценаріях.

Хоча є деякі недоліки у сценаріях збоїв та відновлення, загальний рівень працездатності модуля високий. Наявність історії вибору маршрутів та їхнє генерування допомагають забезпечити стабільність роботи системи.

Можливість модуля ефективно перебудовувати маршрути в реальному часі відповідно до екстрених ситуацій, таких як поломка літака, погіршення погоди, проблеми з перевезеннями, тощо, робить його корисним для практичного використання авіакомпаніями.

Незважаючи на гарні показники, існують можливості для вдосконалення, такі як введення захисту від збоїв та розширення тестування на різних апаратних засобах та в різних умовах. У конкурентних системах є інші критерії оптимізації, які можуть бути додані до програмного модуля.

В цілому, розроблений програмний модуль виглядає перспективним та конкурентоспроможним інструментом для оптимізації маршрутів в авіаційній сфері. З урахуванням подальшого вдосконалення та доробок, він може стати значущим внеском у сучасні системи управління маршрутами.

Враховуючи переваги та обмеження, подальший розвиток та вдосконалення системи може значно покращити її придатність для практичного використання в авіаційній галузі.

До переваг запропонованого підходу до оптимізації маршрутів в авіаційній сфері можна віднести наступні пункти:

– часова ефективність. Застосування алгоритмів оптимізації дозволяє суттєво скоротити час проходження маршрутів, що є ключовим фактором в авіації, де швидкість реакції на зміни дуже важлива;

– гнучкість та адаптивність. Модуль ефективно працює в різних умовах та при різних обсягах даних, що робить його гнучким і адаптивним до різних сценаріїв використання;

– спрощена реакція на екстрені ситуації. Можливість перебудови маршрутів в реальному часі у випадках аварій, поганих погодних умов чи інших екстрених ситуацій є важливим елементом для забезпечення безпеки та ефективності авіаперевезень;

– спрощення планування. Автоматизований процес оптимізації маршрутів спрощує завдання авіакомпаній щодо розподілу ресурсів та маршрутизації літаків.

– можливості подальшого вдосконалення. Виявлені області для подальшого вдосконалення, такі як захист від збоїв та опрацювання додаткових умов перебудови маршрутів, створюють можливості для розвитку та вдосконалення системи;

До обмежень запропонованого підходу до оптимізації маршрутів в авіаційній сфері можна віднести;

– нестійкість у випадку збоїв. Система не володіє захистом від можливих збоїв, що може призвести до перерв у роботі та втрати даних. Введення надійних механізмів відновлення та захисту є необхідним;

– потреба в додатковому тестуванні. Недостатня кількість тестів у різних умовах та на різних апаратних платформах може обмежувати впевненість у стабільності та ефективності системи;

– обмежена універсальність. Система може бути менш ефективною у випадках, коли змінюються умови, які не були враховані в тестових сценаріях;

– не враховані всі можливі екстрені ситуації. Сценарії реального використання охоплюють тільки обрані екстрені ситуації, і не враховують всі можливі варіанти аварій та проблем, що можуть виникнути в реальному середовищі;

– необхідність налаштування для реального використання. Для переходу від симуляції до реального використання, системі потрібно налаштування для взаємодії з реальною авіаційною системою та врахування реальних обмежень.

4.3. Висновки до розділу

На сучасному етапі розвитку авіаційної галузі системи оптимізації маршрутів грають важливу роль у підвищенні ефективності та безпеки авіаперевезень. Проведений аналіз результатів отриманих від систем оптимізації та порівняння їх із запропонованим підходом дозволяє зробити кілька ключових висновків.

Існуючі систем оптимізації маршрутів, покладаються на наступні чинники:

– ефективність в реальному часі: багато систем вже на сьогодні здатні оптимізувати маршрути в реальному часі, що робить їх вельми конкурентоспроможними. Вони використовують розумні алгоритми та великі обсяги даних для швидкої реакції на зміни в умовах;

– широкий функціонал: більшість існуючих систем мають розширений функціонал, включаючи врахування погодних умов, екстрених ситуацій та інших факторів, що можуть впливати на маршрутизацію;

– стійкість та безпека: зазвичай ці системи обладнані заходами безпеки та мають механізми відновлення у разі виникнення проблем, що робить їх стійкими та надійними.

Зважаючи на аналіз існуючих систем оптимізації маршрутів, можна виділити переваги та обмеження запропонованого підходу:

– часова ефективність та гнучкість запропонованого підходу відзначається високими показниками часової ефективності та гнучкості при оптимізації маршрутів. Алгоритми оптимізації роблять його швидким та адаптивним до різних сценаріїв;

– можливість перебудови маршрутів в реальному часі у випадку екстрених ситуацій дозволяє покращити безпеку та ефективність авіаперевезень. Це дуже важлива функція, без якої не можливе повноцінне функціонування авіаційних систем. Наявність цього механізму полегшить подальший розвиток програмного модуля.

Проте, нестійкість у випадку збоїв та відсутність деталізованого захисту може вплинути на безперебійну роботу системи. В програмному модулі докладено належних зусиль для шифрування та зберігання даних, однак цей аспект потребує подальшого покращення.

Підводячи загальний висновок, запропонований підхід до оптимізації маршрутів у авіаційній сфері виявляється конкурентоспроможним у порівнянні із вже існуючими системами. Висока часова ефективність та можливість перебудови в реальному часі роблять його привабливим для впровадження.

Однак, для успішного впровадження та практичного використання, необхідно враховувати певні обмеження, зокрема, введення заходів стабільності та безпеки системи. Розвиток та вдосконалення механізмів відновлення, а також більш широке тестування в різних умовах, можуть сприяти подальшій інтеграції даного підходу у сучасну авіаційну систему.

ВИСНОВКИ

Сучасне життя дуже важко уявити без авіаційної галузі: пасажирські перевезення, вантажні перевезення, патрулювання територій, гуманітарні місії та десятки інших застосувань вже є невід’ємною частиною нашого життя. Авіація дуже стара галузь, яка бере свій початок аж в кінці 18 століття, коли люди реалізували перші польоти на повітряних кулях, глайдерах та інших літаючих об’єктах.

В 1903 році було розроблено перший контрольований моторизований літак, який був проривом в галузі авіації. Однак, перші літаки мали виключно військове призначення, тобто були частиною військової авіації, та використовувались для спостереження, зв'язку та бойових дій.

Ближче до середини 20 століття, відбулись перші трансатлантичні та трансконтинентальні авіапольоти, які відкрили міжнародну цивільну авіацію.

Не зважаючи на довгу історію розвитку, та постійне покращення наявних літальних апаратів, оптимізація витрат є однією з ключових та найбільш актуальних проблем у авіаційній галузі.

Існує безліч речей, які можна оптимізувати: маршрути літаків, планування та розподіл пального, викиди CO_2, бортовий персонал, обробку багажу і пасажирів, тощо. Раніше всі ці параметри розраховувались людьми і мали досить велику похибку, яка призводила до зменшення показників ефективності. З часом певні розрахунки почали делегувати машинам, а в наш час майже повністю автоматизовано процеси різних розрахунків.

У роботі було розглянуто різноманітні методи та стратегії для оптимізації маршрутів у транспортних системах, а також підкреслюється важливість ефективного управління рухом та логістичними процесами.

Розглянуті алгоритми включають в себе класичні та сучасні підходи, які дозволяють вирішувати завдання оптимізації в різних галузях, включаючи автомобільний та авіатранспорт, логістику та доставку.

Методи оптимізації маршрутів, генетичні алгоритми, жадібні алгоритми та алгоритми імітації відпалу, надають різні підходи до вирішення складних задач управління транспортними потоками. Кожен з них має свої переваги та області застосування, що визначає їхню ефективність в конкретних сценаріях. ЗМТЗ, така як задача комівояжера, є типовим прикладом оптимізаційної задачі, де важливою є здатність знаходження найбільш оптимального маршруту для ефективного пересування агента або транспортного засобу. Різноманітні варіації ЗМТЗ, включаючи обмеження по вантажопідйомності, часові вікна, різні види транспорту та інші умови, вказують на велику варіабельність цих задач і необхідність застосування різних стратегій для їх ефективного вирішення.

Розвиток та застосування методів оптимізації маршрутів в сучасних транспортних системах сприяє підвищенню продуктивності, ефективності та економічності у сфері транспорту та логістики. Вдосконалення цих методів відкриває нові можливості для подолання викликів, пов'язаних із ростом обсягів транспортних потоків та ускладненням логістики.

У роботі проведено обширне дослідження, спрямоване на розробку оптимізаційного алгоритму для вирішення завдання оптимізації маршрутів та підвищення ефективності використання літаків авіакомпанією. Основна увага була сконцентрована на визначенні початкових умов для маршрутів та створенні математичної моделі, щоб відобразити основні аспекти оптимізації. Розроблюваний алгоритм має сприяти ефективній роботі авіакомпанії, оптимізуючи вибір маршрутів та використання літаків. При цьому прибутковість не зводиться в абсолют, а система покладається на інші ключові фактори, для підвищення реалістичності системи.

Важливим етапом стала розробка математичної моделі, яка враховує об'єктиви мінімізації і максимізації та обмежень, що виникають з особливостей завдання. Уточнення функції для мінімізації, таких як час польотів, відображено як спроба покращити ефективність системи перевезень. Максимізація функції прибутковості, дозволить підвищити коефіцієнт ефективності використання літаків авіакомпанією. Алгоритм оптимізації був ретельно розроблений та розподілений на дрібні частини для полегшення впровадження та забезпечення його модульності.

Створення основи для подальшого вдосконалення та використання розробленого алгоритму в практичних умовах авіакомпанії є одною з основних цілей цієї роботи. Проведені аналіз та розробка є важливими етапами в дослідженні, що сприятиме подальшій оптимізації та вдосконаленню системи управління маршрутами авіакомпанії. На основі цих кроків будуються наступні етапи дослідження та розробки, де будуть проведені тестування розробленого алгоритму та визначені його переваги та недоліки.

Під час проектування програмного модуля для оптимізації маршрутів та підвищення ефективності використання літаків авіакомпанією, для досягнення ефективності та надійності програмного модуля, була використана архітектура *MVC*. Ця архітектура дозволяє чітко визначити ролі різних компонентів, забезпечуючи модульність та розширюваність.

Визначено функціональні вимоги, що включають в себе здійснення оптимізації маршрутів для авіакомпанії, використовуючи дані з локальної та хмарної баз даних. Синхронізація даних, застосування шифрування для захисту конфіденційності, а також можливість роботи в автономному режимі та моніторинг подій для виявлення проблем безпеки також є важливими частинами функціональних вимог.

Алгоритм оптимізації було ретельно розроблено та реалізовано з використанням мови програмування та підходів, які найкраще відповідають поставленим завданням. Процес реалізації включав в себе визначення початкових умов маршрутів, розробку математичної моделі, а також деталізацію алгоритму на дрібні частини для забезпечення ефективності та чіткості коду.

Додатковий функціонал включає в себе використання локальної та хмарної баз даних, що дозволяє зберігати та отримувати дані в режимі реального часу. Також, шифрування даних забезпечує конфіденційність та безпеку інформації, а реалізація моніторингу та журналювання подій дозволяє вчасно виявляти та вирішувати проблеми безпеки.

Графічна реалізація програмного модуля використовує фреймворк *Flutter* для створення інтуїтивно зрозумілого та естетичного інтерфейсу користувача. Взаємодія

з користувачем оптимізована для зручного використання функцій модуля, забезпечуючи при цьому зручність та ефективність роботи.

В цілому, проектування програмного модуля було виконано з урахуванням найкращих практик розробки програмного забезпечення, а реалізація включає в себе високоякісний код, гнучкість та можливість масштабування для подальшого розвитку.

Існуючі систем оптимізації маршрутів, покладаються на наступні чинники:

– ефективність в реальному часі. Багато систем вже на сьогодні здатні оптимізувати маршрути в реальному часі, що робить їх вельми конкурентоспроможними. Вони використовують розумні алгоритми та великі обсяги даних для швидкої реакції на зміни в умовах;

– широкий функціонал. Більшість існуючих систем мають розширений функціонал, включаючи врахування погодних умов, екстрених ситуацій та інших факторів, що можуть впливати на маршрутизацію;

– стійкість та безпека. Зазвичай ці системи обладнані заходами безпеки та мають механізми відновлення у разі виникнення проблем, що робить їх стійкими та надійними.

Зважаючи на аналіз існуючих систем оптимізації маршрутів, можна виділити переваги та обмеження запропонованого підходу:

– часова ефективність та гнучкість запропонованого підходу відзначається високими показниками часової ефективності та гнучкості при оптимізації маршрутів. Алгоритми оптимізації роблять його швидким та адаптивним до різних сценаріїв;

– можливість перебудови маршрутів в реальному часі у випадку екстрених ситуацій дозволяє покращити безпеку та ефективність авіаперевезень.

Це дуже важлива функція, без якої не можливе повноцінне функціонування авіаційних систем. Наявність цього механізму полегшить подальший розвиток програмного модуля.

Проте, нестійкість у випадку збоїв та відсутність деталізованого захисту може вплинути на безперебійну роботу системи. В програмному модулі докладено зусиль

для шифрування та зберігання даних, однак цей аспект потребує подальшого покращення.

Підводячи загальний висновок, запропонований підхід до оптимізації маршрутів у авіаційній сфері виявляється конкурентоспроможним у порівнянні із вже існуючими системами. Висока часова ефективність та можливість перебудови в реальному часі роблять його привабливим для впровадження.

Проте, для успішного впровадження та практичного використання, необхідно враховувати певні обмеження, зокрема, введення заходів стабільності та безпеки системи. Розвиток та вдосконалення механізмів відновлення, а також більш широке тестування в різних умовах, можуть сприяти подальшій інтеграції даного підходу у сучасну авіаційну систему.

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. Тенденції розвитку транспорту і зв'язку [Електронний ресурс] – режим доступу: *http://buklib.net/books/33285/* (дата звернення: 3.10.2023)

4. Особливості розвитку авіаційної галузі на міжнародному та національному рівні в умовах глобалізації [Електронний ресурс] – режим доступу: *https://ev.nmu.org.ua/docs/2017/4/EV20174_092-099.pdf* (дата звернення: 5.10.2023)

5. Аналіз розвитку підприємств авіаційної галузі і визначення економічних проблем глобалізації [Електронний ресурс] – режим доступу: *http://www.agrosvit.info/pdf/3_2021/9.pdf* (дата звернення: 5.10.2023)

6. Гуляницький Л. Ф., Мулеса О. Ю. Прикладні методи комбінаторної оптимізації: навч. посіб. Київ : Видавничо-поліграфічний центр «Київський університет», 2016. 146 с.

7. Оптимізація жадібних алгоритмів пошуку для скомбінованих послідовностей даних [Електронний ресурс] – режим доступу: *https://science.lpnu.ua/sites/default/files/journal-paper/2017/oct/6574/005-029-036.pdf* (дата звернення: 6.10.2023)

8. Транспортна задача, її властивості та методи розв'язування [Електронний ресурс] – режим доступу: *http://www.cyb.univ.kiev.ua/library/books/iksanov-26.pdf* (дата звернення: 6.10.2023)

9. *New benchmark instances for the Capacitated Vehicle Routing Problem. European Journal of Operational Research* [Електронний ресурс] – режим доступу: *https://www.researchgate.net/publication/306069468_New_Benchmark_Instances_for_the_Capacitated_Vehicle_Routing_Problem* (дата звернення: 8.10.2023)

10. *Quantum Inspired Algorithm for a VRP with Heterogeneous Fleet Mixed Backhauls and Time Windows* [Електронний ресурс] – режим доступу: *https://www.researchgate.net/publication/306586493_Quantum_Inspired_Algorithm_for_a_VRP_with_Heterogeneous_Fleet_Mixed_Backhauls_and_Time_Windows* (дата звернення: 10.10.2023)

11. *An improved particle swarm optimization for carton heterogeneous vehicle routing problem with a collection depot* [Електронний ресурс] – режим доступу: *https://link.springer.com/article/10.1007/s10479-015-1792-x* (дата звернення: 10.10.2023)

12. *An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows* [Електронний ресурс] – режим доступу: *https://www.sciencedirect.com/science/article/abs/pii/S0360835214003453* дата звернення: 10.10.2023)

13. *Vehicle routing with backhauls: Review and research perspectives* [Електронний ресурс] – режим доступу: *https://www.sciencedirect.com/science/article/abs/pii/*S0305054817302794 (дата звернення: 10.10.2023)

14. *An improved formulation for the multi-depot open vehicle routing problem* [Електронний ресурс] – режим доступу: *https://link.springer.com/article/10.1007/s00291-015-0408-9* (дата звернення: 13.10.2023)

15. *Multi-depot periodic vehicle routing problem with due dates and time windows* [Електронний ресурс] – режим доступу: *https://www.tandfonline.com/doi/abs/10.1057/s41274-017-0206-7* (дата звернення: 16.10.2023)

16. *The vehicle routing problem* [Електронний ресурс] – режим доступу: *https://www.sciencedirect.com/science/article/abs/pii/S0360835215004775* (дата звернення: 20.10.2023)

17. *Integrating a heterogeneous fixed fleet and a flexible assignment of destination depots in the waste collection VRP with intermediate facilities* [Електронний ресурс] – режим доступу: *https://infoscience.epfl.ch/record/215053* (дата звернення: 21.10.2023)

18. *A survey on dynamic and stochastic vehicle routing problems. International Journal of Production Research* [Електронний ресурс] – режим доступу: *https://inria.hal.science/hal-01224562/document* (дата звернення: 23.10.2023)

19. *Heuristics for tactical time slot management: a periodic vehicle routing problem view* [Електронний ресурс] – режим доступу: *https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12403* (дата звернення: 23.10.2023)

20. Постановки та математичні моделі проблем оптимізації маршрутів літальних апаратів із динамічними депо [Електронний ресурс] – режим доступу: *https://icyb180.org.ua/publications/postanovki-ta-matematichni-modeli-problem-optimizatsiyi-marshrutiv-litalnih-aparativ-iz-dinamichnimi-depo/* (дата звернення: 23.10.2023)

21. *Dart overview* [Електронний ресурс] – режим доступу: *https://dart.dev/overview* (дата звернення: 1.11.2023)

22. *Why is Flutter a Good Choice for Cross-Platform Projects?* [Електронний ресурс] – режим доступу: *https://www.solutelabs.com/blog/flutter-for-cross-platform-app-development* (дата звернення: 3.11.2023)

23. *Widget catalog* [Електронний ресурс] – режим доступу: *https://docs.flutter.dev/ui/widgets* (дата звернення: 4.11.2023)

24. *Why Bloc?* [Електронний ресурс] – режим доступу: *https://bloclibrary.dev/#/whybloc* (дата звернення: 6.11.2023)

25. *What Is Unit Testing? Types, Tools, and Best Practices* [Електронний ресурс] – режим доступу: *https://www.spiceworks.com/tech/devops/articles/what-is-unit-testing/* (дата звернення: 9.11.2023)

Лістинг коду класу *AlgorithmUtil*

```
import 'package:aircrafts_router/src/algorithm_util/models/airport.dart';

import 'models/aircraft.dart';
import 'models/aircraft_route.dart';
import 'models/transportation_resource.dart';
import 'package:collection/collection.dart';

class AlgorithmUtil {
  AlgorithmUtil._privateConstructor();

  static final AlgorithmUtil _instance = AlgorithmUtil._privateConstructor();

  factory AlgorithmUtil() {
    return _instance;
  }

  final List<AircraftRoute> _generatedRoutes = [];

  List<AircraftRoute> get generatedRoutes => _generatedRoutes;

  //Should be moved to bloc
  final List<Aircraft> _aircrafts = [];

  //Should be moved to bloc
  List<Aircraft> get aircrafts => _aircrafts;
```

```
//Should be moved to bloc
final List<Airport> _airports = [];

final List<TransportationResource> _criticalPriorityRoutes = [];

final List<TransportationResource> _highPriorityRoutes = [];

final List<TransportationResource> _midPriorityRoutes = [];

final List<TransportationResource> _lowPriorityRoutes = [];

final List<TransportationResource> _notSortedRoutes = [];

void generateStartRoutesWithoutAlgorithm(
  List<Airport> airports,
  List<TransportationResource> transportationResources,
  List<Aircraft> planes,
) {
  _airports.addAll(airports);
  _notSortedRoutes.addAll(transportationResources);

  _generatedRoutes.addAll(_notSortedRoutes.map(
      (resource) => getAircraftRouteFromTransportationResource(resource)));

  planes.forEach((plane) {
    AircraftRoute? nextRoute = getNextRouteSimplified(plane);
    if (nextRoute != null) {
      _generatedRoutes.add(getAdditionalTransitionIfNeeded(plane, nextRoute));
      _notSortedRoutes.remove(getTransportationResourceFromAircraftRoute(
          nextRoute, _notSortedRoutes));
```

```
      plane.aircraftRoutes
          .add(getAdditionalTransitionIfNeeded(plane, nextRoute));
      _aircrafts.add(plane);
    }
  });
}

void generateStartRoutes(
  List<Airport> airports,
  List<TransportationResource> transportationResources,
  List<Aircraft> planes,
) {
  _airports.addAll(airports);
  _criticalPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.critical));

  _highPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.high));

  _midPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.mid));

  _lowPriorityRoutes
      .addAll(sortByPriority(transportationResources, Priority.low));

  planes.forEach((plane) {
    AircraftRoute? nextRoute = getNextRoute(plane);
    if (nextRoute != null) {
      removeFromList(nextRoute);
      _generatedRoutes.add(getAdditionalTransitionIfNeeded(plane, nextRoute));
```

```
      plane.aircraftRoutes
          .add(getAdditionalTransitionIfNeeded(plane, nextRoute));
      _aircrafts.add(plane);
    }
  });
}

AircraftRoute getAdditionalTransitionIfNeeded(
    Aircraft plane, AircraftRoute route) {
  AirportPosition routeStartPosition = plane.aircraftRoutes.isEmpty
      ? plane.baseAircraftPosition.airportPosition
      : route.startPoint.airportPosition;

  if (routeStartPosition != route.startPoint.airportPosition) {
    var start = _airports.firstWhere((element) =>
        element.airportPosition.position == routeStartPosition.position);

    return AircraftRoute(
      name:
          "Route From${plane.baseAircraftPosition.name}to${route.endPoint.name}",
      startPoint: _airports.firstWhere((element) =>
          element.airportPosition.position == routeStartPosition.position),
      transitionPoint: route.startPoint,
      endPoint: route.endPoint,
      routeProfit: route.routeProfit,
      routePriority: route.routePriority,
    );
  } else {
    return route;
  }
```

```
}

void removeFromList(AircraftRoute route) {
  if (_notSortedRoutes.isNotEmpty) {
    _notSortedRoutes.remove(
        getTransportationResourceFromAircraftRoute(route, _notSortedRoutes));
    return;
  }

  if (route.routePriority == Priority.critical) {
    _criticalPriorityRoutes.remove(getTransportationResourceFromAircraftRoute(
        route, _criticalPriorityRoutes));
  } else if (route.routePriority == Priority.high) {
    _highPriorityRoutes.remove(getTransportationResourceFromAircraftRoute(
        route, _highPriorityRoutes));
  } else if (route.routePriority == Priority.mid) {
    _midPriorityRoutes.remove(getTransportationResourceFromAircraftRoute(
        route, _midPriorityRoutes));
  } else if (route.routePriority == Priority.low) {
    _lowPriorityRoutes.remove(getTransportationResourceFromAircraftRoute(
        route, _lowPriorityRoutes));
  }
}

TransportationResource getTransportationResourceFromAircraftRoute(
    AircraftRoute route, List<TransportationResource> listOfRoutes) {
  return listOfRoutes.firstWhere((element) =>
      element.startPoint == route.startPoint &&
      element.endPoint == route.endPoint &&
      element.amount * element.costPerUnit == route.routeProfit);
```

```
}

List<TransportationResource> sortByPriority(
    List<TransportationResource> transportationResources, Priority priority) {
  var prioritizedRoutes = transportationResources
      .where((element) => element.priority == priority);
  return sortByProfit(prioritizedRoutes.toList());
}

List<TransportationResource> sortByProfit(
    List<TransportationResource> transportationResources) {
  transportationResources
      .sort((a, b) => getProfit(a).compareTo(getProfit(b)));
  return transportationResources.reversed.toList();
}

int getProfit(TransportationResource transportationResource) {
  return transportationResource.amount * transportationResource.costPerUnit;
}

bool didPlaneCanPickup(int planeSpaceAmount, int resourceAmount) {
  return planeSpaceAmount >= resourceAmount ? true : false;
}

AircraftRoute? getNextRouteSimplified(Aircraft plane) {
  final firstSimplifiedTransportation = getFirstSimplifiedTransportation(
      plane.transportSpaceAmount, _notSortedRoutes);

  if (firstSimplifiedTransportation == null) {
    return null;
```

```
  }

  final nextRoute = getAircraftRouteFromTransportationResource(
      firstSimplifiedTransportation);
  return nextRoute;
}

AircraftRoute? getNextRoute(Aircraft plane) {
  AirportPosition routeStartPosition =
      plane.baseAircraftPosition.airportPosition;

  if (_criticalPriorityRoutes.isNotEmpty) {
    try {
      return getAircraftRouteFromTransportationResource(
          getFirstSuitableTransportation(routeStartPosition,
              plane.transportSpaceAmount, _criticalPriorityRoutes));
    } catch (e) {
      print("No suitable objects in _criticalPriorityRoutes list");
    }
  } else if (_highPriorityRoutes.isNotEmpty) {
    try {
      return getAircraftRouteFromTransportationResource(
          getFirstSuitableTransportation(routeStartPosition,
              plane.transportSpaceAmount, _highPriorityRoutes));
    } catch (e) {
      print("No suitable objects in _highPriorityRoutes list");
    }
  } else if (_midPriorityRoutes.isNotEmpty) {
    try {
      return getAircraftRouteFromTransportationResource(
```

```
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _midPriorityRoutes));
    } catch (e) {
      print("No suitable objects in _midPriorityRoutes list");
    }
  } else if (_lowPriorityRoutes.isNotEmpty) {
    try {
      return getAircraftRouteFromTransportationResource(
        getFirstSuitableTransportation(routeStartPosition,
            plane.transportSpaceAmount, _lowPriorityRoutes));
    } catch (e) {
      print("No suitable objects in _lowPriorityRoutes list");
    }
  }

  return null;
}

TransportationResource getFirstSuitableTransportation(
      AirportPosition routeStartPosition,
      int planeSpaceAmount,
      List<TransportationResource> prioritizedList) =>
    prioritizedList.firstWhere(
      (element) =>
          element.startPoint.airportPosition == routeStartPosition &&
          didPlaneCanPickup(planeSpaceAmount, element.amount),
      orElse: () => prioritizedList.firstWhere((element) =>
          didPlaneCanPickup(planeSpaceAmount, element.amount)));

TransportationResource? getFirstSimplifiedTransportation(
```

```
      int planeSpaceAmount, List<TransportationResource> prioritizedList) =>
    prioritizedList.firstWhereOrNull(
      (element) => didPlaneCanPickup(planeSpaceAmount, element.amount),
    );

AircraftRoute getAircraftRouteFromTransportationResource(
    TransportationResource transportation) {
  return AircraftRoute(
    name:
        "Route From${transportation.startPoint.name}to${transportation.endPoint.name}",
    startPoint: transportation.startPoint,
    endPoint: transportation.endPoint,
    routeProfit: transportation.costPerUnit * transportation.amount,
    routePriority: transportation.priority,
  );
 }
}
```