Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра прикладної математики

Магістерська робота

Освітній ступінь: магістр

На тему: «Моделювання процесів оптимізації при вирішенні логістичної задачі на прикладі спеціальних та регулярних повітряних рейсів»

Виконала: студентка 2 року навчання
Спеціальності
122 Інженерія програмного забезпечення
Харченко Марина Вадимівна
Керівник Авраменко О.В.
кандидат фіз.-мат. наук

Рецензент О. В. Авраменко
Магістерська робота захищена
з оцінкою_____
Секретар ЕК С.А. Мелещенко
«___» _____2023

Київ 2023

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра прикладної математики

ЗАТВЕРДЖУЮ
Зав. кафедри мультимедійних систем,
доктор техн. наук, декан ФІ
_____…
(підпис)
“____”_____ 202_ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на магістерську роботу
студентці Харченко Марині
факультету інформатики 2 курсу магістерської програми

ТЕМА: Моделювання процесів оптимізації при вирішенні логістичної задачі на прикладі спеціальних та регулярних повітряних рейсів.

Вихідні дані:

Зміст ТЧ до курсової роботи:

Вступ

Анотація

1.

2.

3.

Висновки

Список використаної літератури та електронних ресурсів

Додатки

Дата видачі “_____” _____202_ р.

Керівник _____ Завдання отримано _____

Тема: Моделювання процесів оптимізації при вирішенні логістичної задачі на прикладі спеціальних та регулярних повітряних рейсів.

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи		
2.	Вивчення предметної області		
3.	Огляд засобів реалізації		
4.	Вивчення технологій		
5.	Написання першої частини курсової роботи		
6.	Написання другої частини курсової роботи		
7.	Написання третьої частини курсової роботи		
8.	Написання висновків курсової роботи		
9.	Перегляд змісту роботи з керівником		
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника		
11.	Створення презентації		
12.	Захист роботи		

Студентка Харченко Марина
Керівник

“____” _____

Зміст

Анотація

У сучасному глобалізованому світі, здатність ефективно вирішувати логістичні проблеми стає життєво важливою не тільки для держав, але й для приватних підприємств. Конкретні рішення логістичних задач лягають в основу росту економіки та благоустрій суспільства, впливаючи на все - від ефективності доставки товарів та послуг для звичайних споживачів, до стратегічних заходів національної безпеки.

Ці виклики зумовлюють необхідність глибокого розуміння складних систем та взаємозалежностей, а також здатності маневрувати в умовах невизначеностей та ризиків. Знаходження оптимального рішення логістичних задач стає ключовим аспектом, що дозволяє зменшити витрати ресурсів та збільшити продуктивність.

В цьому контексті, моделювання виступає критичним етапом. Воно дозволяє симулювати обрану логістичну стратегію та оцінити різні сценарії, виявляючи перспективи та неочевидні помилки заздалегідь. Більше того, процес моделювання відіграє важливу роль у виявленні більш оптимальних рішень, нерідко відкриваючи альтернативні шляхи, які на перший погляд можуть бути непомітними.

Таким чином, моделювання при вирішенні логістичних задач не є просто технічним кроком, але й є стратегічно важливим аспектом в процесі прийнятті рішень при вирішенні логістичних проблем. Роль моделювання у знаходженні більш продуктивних, економічно ефективних та стабільних логістичних процесів складно переоцінити.

В рамках цієї роботи була поставлена та розв’язана логістична задача оптимізація авіарейсів за допомогою процесу моделювання.

Вступ

Логістика, як поняття, включає детальне планування, організацію, керування та виконання складних операцій. Її коріння сягають військової потреби в розвинутих підходах до руху та обслуговування військових частин, але сьогодні логістика знайшла своє застосування в широкому спектрі секторів, від бізнес-операцій до секторів, за які є відповідальною держава. Успіх вирішення будь-якої логістичної задачі неминуче залежить від здатності досягти поставленої мети якомога ефективніше, що підкреслює критичну потребу в оптимізації.

Державні актори регулярно стикаються з логістичними рішеннями, починаючи від аспектів військового розгортання та громадської безпеки до будівництва та обслуговування життєво важливої інфраструктури, такої як залізниці, порти та автомагістралі. Ці завдання слугують основою для економіки держави і благополуччя її громадян. Наприклад, визначення найефективніших маршрутів та графіків для громадського транспорту може мати величезний вплив на продуктивність міста та якість життя. Стратегічне розміщення військових ресурсів, з іншої сторони, може значно вплинули на національну безпеку. Рішення щодо інфраструктурних проектів, таких як будівництво автомагістралей або розширення портів, мають довготривалий вплив на економічний розвиток та конкурентоспроможність країни.

Приватний сектор має схожий набір логістичних проблем. Транспортування сировини до виробничих потужностей, розповсюдження готової продукції до роздрібних торговців або безпосередньо до споживачів - всі ці завдання вимагають вироблення алгоритму або його адаптації для впровадження певної стратегії. Наприклад, виробник автомобілів повинен забезпечувати неперервне постачання компонентів на свою лінію збірки.Таким чином, успішні бізнеси часто є тими, хто

зумів оптимізувати свої логістичні операції, забезпечуючи їм конкурентні переваги, такі як економічність, надійність та швидкість доставки.

При розв'язанні логістичних завдань необхідно враховувати різні фактори, включаючи, але не обмежуючись, швидкістю доставки, потенційними ризиками, вартістю транспортування, потужністю різних видів транспорту, масштабованістю та доцільністю. Деякі логістичні проблеми можуть бути вирішені в автономному режимі, тоді як інші вимагають прийняття рішень в реальному часі. Наприклад, маршрут вантажного судна можна запланувати завчасно, враховуючи різні фактори, тоді як диспетчеризація таксі в місті повинна динамічно реагувати на поточний попит.

Не можна не підкреслити, що при вирішенні логістичних проблем необхідно враховувати широкий спектр параметрів та обмежень. Перед тим, як будь-яке рішення можна впровадити, його спочатку потрібно змоделювати та протестувати в контрольованих умовах. Процес моделювання не тільки зменшує ризики та помилки, але й допомагає виявити проблеми та обмеження, які спочатку могли бути не очевидними. Більше того, під час процесу моделювання може бути можливо виявити більш оптимальні рішення.

Отже, практика моделювання - це не просто підготовча фаза, але і невід'ємна частина побудови самого рішення логістичних проблем.

Розділ 1. ПОСТАНОВКА ЛОГІСТИЧНОЇ ЗАДАЧІ ОПТИМІЗАЦІЇ

1.1 Постановка задачі

Сформулюємо абстрактну логістичну проблему:

Компанія, яка займається перевезенням вантажів, володіє кількома типами вантажних літаків. Кожен тип літака має такі характеристики:

1) максимальна відстань безперервного польоту без отримання додаткового пального у кілометрах;

2) максимальний корисний об'єм простору для вантажу всередині літака;

3) максимальна вага, яку він може безпечно перевозити, у кілограмах;

4) вартість експлуатації літака за кожні 100 кілометрів польоту.

Компанія співпрацює з кількома аеропортами на території одного континенту. Дана компанія займається перевезенням вантажу з між аеропортами.

Авіаційні рейси компанії існують в двох типах: регулярних та спеціальних. Регулярні рейси плануються за місяць наперед. Але розклад може бути змінений, якщо з'являються нові запити протягом поточного місяця. Кожен запит, будь то регулярний або спеціальний, має дедлайн (це дата, до якої вантаж повинен прибути до аеропорту призначення). Спеціальні запити можуть бути подані не пізніше, ніж за 5 днів до дедлайну. Спеціальні запити можуть бути відхилені не пізніше, ніж за 7 днів до дедлайну, регулярні запити не можуть бути відхилені. Рейси можуть бути перепланові кожні 6 годин.

Завдання для компанії полягає в максимізації прибутку. Гроші, які компанія отримує за кожен вантаж, обчислюються наступним чином: об'єм вантажу ділиться на 6000. Але кожен рейс не є безкоштовним для компанії,

сума грошей, яку вона втрачає, залежить від кілометрів між вихідним і призначеним аеропортами та вартості літака за кожні 100 кілометрів.

1.2 Аналіз обмежень, що випливають з поставленої умови

Здійснивши аналіз поставленої умови задачі, ми знаходимо обмеження на фізичні характеристики літаків, запитів на перевезення вантажу та сам вантаж. Вважаємо, що нам відомі координати всіх аеропортів, з якими співпрацює компанія. Ці обмеження повинні бути використані для створення верхньої та нижньої границі при генерації сутностей літаків, запити на перевезення вантажу та вантажу.

1.2.1 Обмеження на літаки

Обмеження, що накладаються на літаки випливають з даних про розташування аеропортів.

При обмеженнях для кожного типу літака (враховуючи відому відстань між кожною парою аеропортів), існує кілька важливих факторів, які необхідно врахувати:

1) Мінімальна кількість кілометрів, яку літак може пролетіти без заправки, не може бути меншою за відстань між двома найближчими аеропортами, з якими компанія співпрацює. Це обмеження гарантує, що літак зможе безпечно здійснювати польоти між найближчими аеропортами, не потребуючи додаткової заправки.

2) Крім того, існує верхня межа відстані, яку літак може пролетіти без заправки. Це обмеження встановлює верхній ліміт в кілометрах для кожного типу літака. Враховуючи це обмеження, компанія може планувати рейси та визначати оптимальні маршрути, які не перевищують максимальну відстань польоту без заправки.

3) Компанія також повинна враховувати максимальну вартість за кожні 100 кілометрів. Це важливо для планування витрат та оптимізації прибутку. Встановлення розумного верхнього обмеження допомагає уникнути зайвих витрат і забезпечити ефективне використання ресурсів.

4) З іншого боку, компанія повинна враховувати мінімальну вартість за кожні 100 кілометрів. Це може впливати на вибір оптимального типу літака для конкретного маршруту. Встановлення розумного нижнього обмеження гарантує, що компанія не зазнає збитків від перевезення вантажів на короткі відстані.

5) Крім того, важливо враховувати максимальний об'єм, який літак може мати всередині. Це означає, що компанія має обмеження щодо кількості і об’єму вантажу, який можна перевезти на одному рейсі. Встановлення максимального об'єму допомагає оптимізувати використання простору всередині літака та забезпечує ефективне розподілення вантажів.

6) Не менш важливим є врахування максимальної ваги, яку літак може перевозити в кілограмах. Це обмеження гарантує безпеку польоту та забезпечує, що літак не буде перевантажений.

Зважаючи на ці обмеження для кожного типу літака, компанія може здійснювати планування рейсів та приймати рішення щодо використання відповідного типу літака для кожного маршруту. Це допомагає забезпечити ефективність, безпеку та економічну вигоду при перевезенні вантажів.

1.2.2 Обмеження на запити на перевезення від клієнтів

Обмеження, що накладаються на запити на перевезення вантажу випливають з даних про фізичні характеристики літаків, що в свою чергу випливають з даних про аеропорти.

1) Вантаж не може бути забраний з вихідного аеропорту до дати та часу, коли він стає доступним у цьому аеропорту. Це обмеження гарантує, що компанія забирає вантаж тільки після його прибуття в вихідний аеропорт.

2) Вантаж не може бути доставлений після дати та часу, визначеного для його передачі. Це забезпечує своєчасну доставку вантажу до призначеного аеропорту.

3) Запити, що використовуються для складання розкладу регулярних рейсів, повинні бути виконані вчасно і виконуватися вчасно (запит може бути виконаний декілька днів до дедлайну). Ми будемо називати їх регулярними запитами, оскільки вони використовуються для планування регулярних рейсів. Ці запити не можуть бути відхилені.

4) Спеціальні запити можуть бути подані не менше, ніж за 5 днів до дедлайну доставки (враховуючи дату та час). Це обмеження забезпечує вчасне подання спеціальних запитів з достатнім терміном для їх обробки та планування.

5) Спеціальні запити можуть бути відхилені не менше, ніж за 7 днів до дедлайну доставки. Це обмеження забезпечує достатній термін для вирішення спеціальних запитів та можливість їх відхилення у встановлений строк.

1.3 Аналіз задачі та викликів, які постають при моделюванні вирішення даної логістичної задачі

Це складна задача оптимізації, що включає багато факторів та обмежень, як просторового, так і часового характеру. Компанія повинна ефективно використовувати різні типи літаків для перевезення вантажів, враховуючи характеристики кожного типу літака: максимальна відстань без додаткового пального, максимальний об'єм, максимальна вага допустимого вантажу та вартість експлуатації за 100 км), а також враховувати різні типи запитів на рейси (спеціальні та регулярні) з їхніми власними обмеженнями.

Головна мета компанії - максимізація прибутку, який обчислюється на основі обсягу перевезеного вантажу, враховуючи витрати на відстань та вартість польоту за 100 км

При вирішенні цієї задачі необхідно враховувати багато обмежень, включаючи просторові обмеження, такі як максимальна відстань, на яку літак може пролетіти без заправки, а також обмеження щодо обсягу та ваги перевезеного вантажу. Крім того, є тимчасові обмеження, пов'язані з датами і часами забору та доставки вантажу, а також з обробкою різних типів запитів на рейси.

Для досягнення оптимальних результатів вирішення цієї задачі компанія повинна використовувати математичне моделювання та вдало вибрати використовувані алгоритми.

Висновки до розділу 1

В цьому розділі була поставлена логістична задача на оптимізацію в контексті авіаційної вантажної компанії, яка працює в рамках регулярних та спеціальних рейсів. Також, було проаналізовані додаткові умови та обмеж

Розділ 2. ЗАСТОСУВАННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ ДЛЯ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Після детального аналізу існуючих алгоритмів, які могли б підійти для вирішення поставленої логістичної задачі з просторовими та часовими обмеженнями, було обрано генетичний алгоритм.

2.1 Обґрунтування доцільності використання генетичного алгоритму

Генетичні алгоритми добре підходять для вирішення задач оптимізації, де простір пошуку великий і складний, як, наприклад, логістична проблема, про яку ми говоримо. Ось декілька причин, чому генетичний алгоритм є хорошим вибором для цієї проблеми:

1. Дослідження великого простору факторів: задача планування рейсів для карго-компанії включає значну кількість змінних, включаючи різні аеропорти, часи рейсів, ємності літаків та вимоги до вантажів. Це призводить до великого простору пошуку з багатьма можливими рішеннями. Генетичні алгоритми розроблені для дослідження таких великих просторів пошуку і з часом зближуються до оптимального або майже оптимального рішення.

2. Обробка кількох обмежень: наша проблема характеризується кількома обмеженнями, включаючи часи рейсів, ємності літаків та характеристики вантажів. Генетичні алгоритми можуть ефективно обробляти ці обмеження за допомогою підходящих функцій пристосування та операторів.

3. Здатність адаптуватися з часом: Однією з ключових особливостей генетичних алгоритмів є їхня здатність розвивати рішення з часом. Це особливо корисно для нашої проблеми, де з часом вводяться спеціальні рейси (які пропонують більше прибутку, але менш гнучкі). Генетичний

алгоритм може адаптувати рішення для включення цих спеціальних рейсів, коли це прибутково і виконуємо.

4. Паралельний пошук: генетичні алгоритми проводять паралельний пошук в просторі рішень. Це означає, що вони підтримують та досліджують кілька рішень одночасно, що збільшує ймовірність знайти високоякісні рішення.

5. Оптимізація: генетичні алгоритми розроблені для оптимізації функції пристосування, яка в нашому випадку може бути загальним прибутком, отриманим компанією. Розвиваючи рішення протягом поколінь, генетичний алгоритм здатний знаходити рішення, які максимізують цей прибуток.

2.2 Загальний опис генетичного алгоритму

Генетичні алгоритми (ГА) є популярним класом еволюційних алгоритмів, які використовують принципи природного відбору та генетики для вирішення проблем оптимізації.

Основні концепціями генетичних алгоритмів є початок з генерації початкової популяції потенційних рішень. Кожне рішення, або "індивід", представлене як структура даних, яка часто подібна до хромосоми. Кожна хромосома складається з "генів", які відповідають окремим аспектам рішення.

ГА використовують три основні типи операторів: селекцію, схрещування (або рекомбінацію) та мутацію. Селекція вибирає найкращі індивіди з популяції для створення наступного покоління. Схрещування обмінює гени між парою індивідів, створюючи нові рішення. Мутація вносить випадкові зміни в гени окремих індивідів.

Генетичний алгоритм використовує циклічний процес. Популяція індивідів оцінюється за допомогою функції пристосування, яка вимірює якість кожного рішення. Найкращі індивіди вибираються для створення

нового покоління через схрещування та мутацію. Процес продовжується до досягнення критерію зупинки, такого як фіксована кількість поколінь або достатнє покращення в якості рішення.

Застосування ГА: генетичні алгоритми використовуються в широкому спектрі областей, включаючи штучний інтелект, оптимізацію, моделювання та обробку сигналів. Вони особливо корисні для вирішення складних проблем оптимізації, які мають багато змінних та обмежень.

Переваги та недоліки: ГА є потужним інструментом, але вони мають свої власні виклики. До переваг відносяться здатність обробляти багато різних типів проблем і робити це паралельно. Проте, вони можуть вимагати значного обчислювального ресурсу, і немає гарантії, що вони знайдуть глобальне оптимальне рішення. Також не завжди очевидно, як ефективно кодувати проблему для розв'язання ГА.

2.3 Застосування алгоритму та його ілюстрація за допомогою псевдокоду

Генетичний алгоритм має 3 основні стадії, а саме: ініціалізація популяції, ітерація, термінація. Стадія ітерації ділиться на окремі етапи, а саме: селекція, кросовер, мутація та заміна.

2.3.1 Стадія ініціалізації популяції.

Популяція: у контексті генетичних алгоритмів, популяція — це сукупність потенційних рішень для даної проблеми. У нашому випадку, кожне рішення в популяції представляє собою розклад польотів для всіх літаків в компанії.

Ініціалізація популяції. ми створюємо нову популяцію заданого розміру, заповнюючи її рішеннями. Кожне рішення ініціалізується як порожній розклад для всіх літаків.

Регулярні і спеціальні відправлення. У нашому випадку, є два типи відправлень: регулярні і спеціальні. Регулярні відправлення заплановані заздалегідь, тоді як спеціальні відправлення можуть додаватися протягом поточного місяця.

Заповнення розкладу польотів. Для кожного літака в розкладі, ми спочатку намагаємося запланувати всі регулярні відправлення. Після цього, ми намагаємося додати спеціальні відправлення, якщо це можливо. Ми перевіряємо, чи можна додати відправлення до розкладу польотів літака, перш ніж додавати його, і оновлюємо стан літака після кожного відправлення.

Повернення популяції. Після того, як всі рішення в популяції були ініціалізовані, ми повертаємо популяцію.

Псевдокод наведено у додатку 1.

2.3.2 Стадія селекції

У даному коді ми ініціалізуємо нову популяцію рішень, які представляють розклади польотів для різних типів літаків.

Популяція. В контексті генетичних алгоритмів, популяція — це набір потенційних рішень для даної проблеми. В нашому випадку, кожне рішення в популяції представляє собою розклад польотів для літака.

Ініціалізація популяції. Ми створюємо нову популяцію заданого розміру, заповнюючи її рішеннями. Кожне рішення ініціалізується як порожній розклад польотів для кожного літака відповідного типу.

Регулярні відправлення. У нашому випадку, є регулярні відправлення, які заплановані заздалегідь. Ми спочатку намагаємося запланувати ці відправлення для кожного літака.

Функції `insert_shipment` та `can_insert_shipment`. Ці функції використовуються для перевірки, чи можна додати відправлення до розкладу польотів літака, та для вставки відправлення у розклад, відповідно.

Створення розкладу польотів. Після того, як ми запланували всі регулярні відправлення для кожного літака, ми створюємо розклад польотів для цього літака та додаємо його до популяції.

Повернення популяції. Після того, як всі розклади польотів були створені та додані до популяції, ми повертаємо популяцію.

Псевдокод наведено у додатку 2.

2.3.3 Стадія кросоверу

Абсолютно, давайте переформулюємо цей процес у новому стилі.

Етап кросоверу, або обміну генами, здійснюється між двома батьківськими рішеннями для створення нового рішення - дитини.

Спочатку копіюємо літаки від одного з батьків. Це означає, що кожен літак у новому рішенні отримує копію відповідного літака від одного з батьків. Але ми не копіюємо розклад польотів цих літаків, лише їх параметри.

Потім ми проходимося по всім відправленням, які були в розкладах обох батьків, в хронологічному порядку. Для кожного відправлення ми намагаємося вставити його в розклад відповідного літака в дитині.

Якщо відправлення є регулярним, ми обов'язково вставляємо його в розклад. Так як регулярні відправлення не можна відхилити, ми повинні гарантувати, що всі регулярні відправлення з батьківських розкладів також присутні в розкладі дитини.

Якщо відправлення є спеціальним, ми спочатку перевіряємо, чи можна його вставити в розклад. Якщо це можливо, ми вставляємо його. Якщо ні - пропускаємо його. Спеціальні відправлення можна відхилити, тому не всі спеціальні відправлення з батьківських розкладів обов'язково будуть присутні в розкладі дитини.

Після того, як ми пройшлися по всім відправленням, дитина стає новим рішенням з власним розкладом польотів для кожного літака. Цей розклад являє собою комбінацію розкладі.

Псевдокод наведено у додатку 3.

2.3.4 Стадія мутації

Індивід. Це одне рішення з нашої популяції. В даному випадку, це розклад польотів для всіх літаків.

Мутація. Ми вносимо зміни в розклад кожного літака, спробувавши додати кожне спеціальне відправлення до розкладу літака.

Спеціальні відправлення. Це відправлення, які ми хочемо додати до розкладу літака під час мутації.

Функція `can_insert_shipment. Використовується для перевірки, чи можна вставити спеціальне відправлення в розклад літака без порушення регулярних рейсів.

Рівень мутації. Це ймовірність, з якою ми ставимо спеціальне відправлення в розклад. Якщо випадково згенероване число менше за рівень мутації, то ми ставимо спеціальне відправлення в розклад.

На цьому етапі ми вносимо деякий рівень випадковості в процес мутації, що дозволяє алгоритму дослідити більший діапазон потенційних рішень.

Псевдокод наведено у додатку 4.

2.3.5 Стадія заміни

Популяція. Це набір поточних рішень. У нашому випадку, це розклади польотів для всіх літаків.

Потомство. Це нові рішення, які були створені шляхом кросоверу та мутації поточної популяції.

Елітизм. Це кількість найкращих рішень з поточної популяції, які ми хочемо зберегти для наступного покоління.

Сортування популяції та потомства. Ми сортуємо поточну популяцію та потомство за їхньою пристосованістю, так що найкращі рішення знаходяться на початку списку.

Створення нової популяції. Ми зберігаємо кілька найкращих рішень з поточної популяції та заповнюємо решту нової популяції потомством.

Повернення нової популяції. Після того, як нова популяція була створена, ми повертаємо її.

Цей код представляє останній етап генетичного алгоритму, після якого ми отримуємо наступне покоління рішень.

Псевдокод наведено у додатку 5.

2.3.6 Стадія термінації

Ця стадія є превіркою, чи було досягнуто умову завершення генетичного алгоритму. У генетичних алгоритмах ми зазвичай встановлюємо деякі умови завершення, щоб алгоритм не працював нескінченно.

Поточне покоління: Це номер поточного покоління в алгоритмі.

Найкращий індивід: Це найкраще рішення, яке було знайдено алгоритмом на даний момент.

Список найкращих пристосованостей: Це список значень пристосованості найкращого індивіда для кожного покоління.

Максимальне число поколінь: Це максимальна кількість поколінь, які алгоритм повинен працювати.

Максимальне число поколінь без змін: Це максимальне число поколінь, протягом яких пристосованість найкращого індивіда може не змінюватись.

Задовільна пристосованість: Це рівень пристосованості, який ми вважаємо достатньо добрим для вирішення проблеми.

Умови завершення:

1. Алгоритм досяг максимального числа поколінь.
2. Пристосованість найкращого індивіда не змінилась протягом визначеного числа поколінь.
3. Пристосованість найкращого індивіда досягла задовільного рівня.

Якщо будь-яка з цих умов виконується, алгоритм завершується. Інакше, алгоритм продовжує працювати.

Псевдокод наведено у додатку 6.

Висновки до розділу 2

В цьому розділі було обґрунтовано доцільність використання генетичного алгоритму. Було наведено опис та застосування цього алгоритму для рішення цієї конкретної задачі.

Розділ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ МОДЕЛЮВАННЯ ПРОЦЕСІВ ОПТИМІЗАЦІЇ ПРИ ВИРІШЕННІ ПОСТАВЛЕНОЇ ЗАДАЧІ

Основними критеріями при виборі технологій були їх мультифункціональність, швидкість розробки та популярність, що на практиці означає існування великої кількості вже випробуваних рішень для багатьох підзадач. Це, в свою чергу дозволяє сконцентрувати ресурси знаходження та побудові рішення для наявної задачі. В контексті даної роботи рішенням є клієнт-серверний застосунок.

3.1 Огляд використаних технологій

Практичним рішенням є клієнт-серверний застосунок. Сервер та клієнт використовують різні технології.

3.1.1 Python, як мова серверної частини

Python - це інтерпретована мова програмування, яку цінують за її зрозумілий синтаксис, легкість вивчення та багатогранність застосування. Вона широко використовується в таких областях, як розробка веб-сайтів, автоматизація, наукові обчислення, і, що особливо важливо, в аналізі даних та математичному моделюванні. Сила Python полягає в її потужних бібліотеках, які спрощують та прискорюють вирішення складних задач, включаючи такі завдання, як моделювання процесів та оптимізація. З активною спільнотою та добре задокументованими ресурсами, Python продовжує залишатися популярним вибором у різних секторах, сприяючи постійному прогресу і розвитку.

3.1.2 TypeScript, як мова клієнтської частини

TypeScript - це статично типізована мова, що є надбудовою над мовою програмування JavaScript, яку переважно використовують для розробки веб-додатків на клієнтській стороні. Ця мова дозволяє виявляти помилки на етапі розробки, а не під час виконання, що поліпшує надійність та зручність підтримки коду. Код TypeScript транспілюється в JavaScript, що гарантує сумісність з різними браузерами. Сумісність TypeScript з фреймворками JavaScript, такими як Angular, React та Vue.js, а також велика спільнота, зміцнюють його позиції у розробці сучасних веб-додатків. В своїй суті, TypeScript пропонує надійніше та більше зручне середовище для розробки на клієнтській стороні, ніж існуючі аналоги.

3.1.3 Django Framework

Django, відкритий веб-фреймворк, написаний на мові Python, має сильну репутацію як надійний та ефективний інструмент. Він розроблений для спрощення створення складних веб-сайтів з використанням баз даних, з увагою до перевикористання та "підключеності" компонентів, зменшення обсягу коду за допомогою принципу "не повторюй себе" (DRY) та філософії прагматичного та чистого дизайну.

Django надає ряд готових функцій, що прискорюють процес розробки. Серед них - аутентифікація користувачів, адміністрування контенту, карти сайту та RSS-стрічки. Ця багатозадачність вбудованих ресурсів робить Django особливо підходящим для моделювання реальних проектів, оскільки розробникам не потрібно витрачати час на побудову загальних елементів і можуть зосередитися на унікальних аспектах своїх додатків. Величезний функціонал Django робить його сильним гравцем у світі веб-фреймворків, його детальна документація та активна спільнота ще є його великими перевагами.

3.1.4 Django Rest Framework

Django Rest Framework (DRF) - це потужний та гнучкий набір інструментів для створення веб-інтерфейсів програмування додатків (Web APIs). Як розширення Django, DRF безпроблемно інтегрується в екосистему Django і використовує можливості Django для побудови надійних API. Це включає підтримку політик аутентифікації, серіалізацію складних типів даних та набори представлень (viewsets) для написання логіки, пов'язаної з HTTP-методами. DRF відзначається своєю простотою використання та універсальністю, що дозволяє розробникам легко створювати API незалежно від того, чи це невеликий веб-додаток, чи система великого масштабу. З чіткою та всебічною документацією DRF забезпечує легкість навчання, дозволяючи розробникам швидко створювати безпечні та підтримувані API. Його внесок у гнучкість та масштабованість Django робить його незамінним інструментом у сучасній веб-розробці.

3.1.5 React

React - це популярна бібліотека JavaScript, яка в основному використовується для створення ефективних та інтерактивних інтерфейсів користувача. React дозволяє розробникам будувати великі веб-додатки, які можуть ефективно оновлюватись та відображатись у відповідь на зміни даних, що робить його потужним вибором для односторінкових додатків, де безперебійний досвід користувача є пріоритетом.

Однією з найвідмінніших особливостей React є його компонентна архітектура. Компоненти, по суті, є ізольованими частинами коду, які представляють різні елементи користувацького інтерфейсу. Вони можуть бути повторно використані в різних частинах додатку, що робить код більш збереженим та процес розробки більш ефективним.

Інноваційна концепція віртуального DOM робить React ще більш продуктивним. Ця техніка полягає у створенні репліки DOM-структури в пам'яті, де зміни відбуваються швидше. Коли стан компонента змінюється, React спочатку оновлює віртуальний DOM, а потім застосовує алгоритм порівняння, щоб визначити мінімальний набір операцій для перетворення фактичного DOM у відповідність до віртуального DOM.

Ще одним перевагою React є його універсальність. React не накладає обмежень на структуру додатку, що дозволяє розробникам поєднувати його з різними бібліотеками або фрей

3.1.6 Chakra

Chakra UI - це дуже популярна відкрита бібліотека JavaScript для створення користувацького інтерфейсу, побудована на основі React. Вона надає розробникам набір простих у використанні, налаштовуваних та доступних компонентів, які можна використовувати для швидкої розробки привабливих та інтерактивних користувацьких інтерфейсів. Однією з відмінних особливостей Chakra UI є її сильне зобов'язання перед стандартами доступності, що забезпечує використання створених з нею додатків якомога більшою кількістю людей.

Бібліотека компонентів Chakra UI досить обширна і включає багато поширених елементів користувацького інтерфейсу, таких як кнопки, форми, модальні вікна та інші, зберігаючи при цьому послідовність та модульність. Крім того, вона пропонує систему стилів, яка адаптується до різних розмірів екрану: розробники можуть написати стилі один раз і мати їх пристосованими до різних розмірів екрану. Також вона підтримує темний режим з коробки, що спрощує створення додатків, які адаптуються до системних налаштувань користувачів.

Підсумовуючи, Chakra UI є комплексним та доступним інструментом, який дозволяє розробникам швидко й ефективно будувати високоякісні

додатки на React зі спрямованістю на зручність використання та естетичну привабливість.

3.1.7 Emotion

Emotion - це бібліотека CSS-in-JS з високою продуктивністю, яка відрізняється своїм динамічним підходом до стилізації в JavaScript. Зазвичай використовується разом з фреймворком React, Emotion дозволяє розробникам поєднувати JavaScript із CSS, сприяючи більш динамічній та компонент-специфічній стилізації. Цей підхід до включення стилів як фундаментального елементу компонента сприяє створенню повторно використовуваних та ізольованих компонентів, які забезпечують послідовний дизайн у всьому додатку.

Серед основних особливостей Emotion можна виділити підтримку створення ізольованого CSS, автоматичне додавання префіксів для різних браузерів та підтримку карт джерел. Ці інструменти допомагають розробникам ефективно управляти стилями. Крім того, підтримка Emotion для API styled і CSS prop надає розробникам широкий спектр методологій стилізації, підкреслюючи його позицію як гнучкого рішення для потреб стилізації.

Однак, те, що справді відрізняє Emotion від інших бібліотек, - це його продуктивність. Вона використовує оптимізований алгоритм вставки стилів та надає пріоритет швидкому рендерингу на сервері, що робить її відмінним вибором для розробників, які прагнуть зменшити час завантаження. Поєднуючи високу продуктивність з широкими можливостями та фокусом на досвіді розробника, Emotion перевершує багато інших бібліотек у ландшафті CSS-in-JS.

3.1.8 Heroku

Heroku - це хмарна платформа, яка надає розробникам безшовний та ефективний спосіб розгортання, управління та масштабування їх додатків. Вона спрощує процес розгортання веб-додатків, надаючи надійну інфраструктуру та зручний інтерфейс. Heroku підтримує різні мови програмування, включаючи Python, Ruby, Java та Node.js, що робить її доступною для широкого спектра розробників.

Однією з найвиразніших особливостей Heroku є її фокус на продуктивності та простоті використання. Завдяки всього кільком простим командам розробники можуть розгортати свої додатки у хмару, уникнувши складних конфігурацій серверів. Heroku бере на себе підтримку інфраструктури, такої як балансування навантаження та масштабування, дозволяючи розробникам зосередитися на створенні та вдосконаленні своїх додатків.

Ще однією перевагою Heroku є масштабованість. Вона автоматично адаптує ресурси відповідно до потреб додатка, забезпечуючи оптимальну продуктивність навіть під час пікових навантажень. Ця гнучкість робить її ідеальним вибором для стартапів та бізнесів, які очікують швидкого зростання.

Крім того, Heroku надає розгалужену екосистему додатків та інтеграцій, які розширюють функціональність платформи. Ці додатки охоплюють різні області, такі як бази даних, кешування, моніторинг та журналювання, дозволяючи розробникам легко розширювати свої додатки додатковими функція

3.2 Проектування сутностей

Дані про аеропорти були взяти з реальної бази даних аеропортів світу, вони були проаналізовані та нормалізовані в процесі побудови рішення.

Деякі фізичні характеристики літаків були взяті з реального світу, щоб використати їх для встановлення верхньої та нижньої межі. Прикладом невеликого вантажного літака став Boeing 767-300F, а прикладом великого вантажного літака - Boeing 747-8F.

Основні моделі в рішенні.

Модель аеропорту.

```
class Airports(models.Model):
  type = models.CharField(max_length=250, blank=True, null=True)
  is_deleted = models.BooleanField(default=False)
  name = models.CharField(max_length=250, blank=True, null=True)
  continent = models.CharField(max_length=250, blank=True, null=True)
  iso_country = models.CharField(max_length=250, blank=True, null=True)
  iso_region = models.CharField(max_length=250, blank=True, null=True)
  municipality = models.CharField(max_length=250, blank=True,
null=True)
  iata_code = models.CharField(max_length=250, blank=True, null=True)
  long = models.FloatField()
  lat = models.FloatField()
```

Модель обраного аеропорту.

```
class SelectedAirports(models.Model):
   session = models.ForeignKey(Sessions, on_delete=models.CASCADE)
   airport = models.ForeignKey(Airport, on_delete=models.CASCADE)
```

Модель типу літака

```
class PlaneTypes(models.Model):
  max_continuous_flight_distance = models.FloatField(default=0)
```

```
  max_volumetric_weight = models.FloatField(default=0)
  max_volume = models.FloatField(default=0)
  max_weight = models.FloatField(default=0)
  cost_per_100_km = models.DecimalField(max_digits=10,
decimal_places=2)
  sessions = models.ForeignKey(Sessions, on_delete=models.CASCADE)
```

Модель літака

```
class Planes(models.Model):
  name = models.UUIDField(primary_key=True, default=uuid.uuid4,
editable=False)
  starting_airport = models.ForeignKey(SelectedAirportss,
on_delete=models.CASCADE)
  type = models.ForeignKey(PlaneTypes, on_delete=models.CASCADE)
```

Модель вантажу

```
class Shipments(models.Model):
  is_scheduled = models.BooleanField()
  is_accepted = models.BooleanField()
  available_at = models.DateTimeField()
  shipment_due = models.DateTimeField()
  is_regular = models.BooleanField(null=True)
  origin_airports = models.ForeignKey(SelectedAirportss,
on_delete=models.CASCADE, related_name='origin_airports')
  destination_airports = models.ForeignKey(SelectedAirportss,
on_delete=models.CASCADE, related_name='destination_airports')
  sessions = models.ForeignKey(Sessions, on_delete=models.CASCADE)

  class Meta:
```

unique_together = ('origin_airports', 'destination_airports')

3.3 Огляд структури побудованого рішення

Побудоване рішення для моделювання вирішення даної логістичної задачі представляє собою клієнт-серверний застосунок. Всі початкові обмеження визначаються на серверній частині. Користувач на клієнті з кожним новим запитом отримує обирає характеристики літаків та вантажів, що у свою чергу задають нові обмеження. Після вводу всіх даних, запускається процес моделювання вирішення цієї логістичної задачі оптимізації. На клієнті користувач бачить логи, завдяки яким він може бачити роботу алгоритму.

Висновки до розділу 3

В цьому розділі ми здійснили загальний огляд використаних технологій. Ознайомились з моделями сутностей, які були використані та структурою кінцевого рішення.

Висновки

В Логістична оптимізація важлива складова багатьох галузей, включаючи транспорт, постачання та виробництво. Ми розглянули різні типи проблем логістичної оптимізації, такі як задачі маршрутизації транспорту, розкладу польотів та планування запасів. Всі ці проблеми можуть бути вирішені за допомогою оптимізації.

Ми сформулювали власну проблему оптимізації логістики: планування відправлень для авіакомпанії, що виконує регулярні та спеціальні відправлення різного обсягу та ваги, з використанням літаків різних типів. Ми визначили цільову функцію та обмеження цієї задачі, а також сформулювали її як задачу багатоцільової оптимізації.

Для вирішення цієї проблеми ми розробили генетичний алгоритм. Генетичний алгоритм включає процеси селекції, схрещування, мутації та заміни, а також критерії завершення. Ми детально розібрали кожний з цих етапів, використовуючи приклади та псевдокод.

У результаті, ми створили рішення для нашої задачі оптимізації логістики. Це рішення може бути використане авіакомпаніями для планування своїх відправлень та максимізації своїх прибутків.

Список використаної літератури та електронних ресурсів

1. Дані про аеропорти світу. [Online]. Available: https://github.com/lxndrblz/Airports
2. "The Vehicle Routing Problem: Latest Advances and New Challenges" by Bruce Golden, et. al.
3. "Optimization in Logistics and Freight Transportation" by Gilbert Laporte.
4. "Supply Chain Management: Strategy, Planning, and Operation" by Sunil Chopra.
5. "Network Flows: Theory, Algorithms, and Applications" by Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin.
6. "Integer Programming" by Laurence A. Wolsey.
7. "A Genetic Algorithm for the Vehicle Routing Problem" by Thangiah, S.R., et al..
8. "A Genetic Algorithm for the Multi-depot Vehicle Routing Problem" by Berger, J., & Barkaoui, M.

Додатки

Додаток 1:

```
def initialize_population(population_size, num_plane_types,
num_planes_of_type, all_regular_shipments, all_special_shipments):
    """
    Initializes a new population of solutions.

    :param population_size: The number of solutions in the population.
    :param num_plane_types: The number of different types of planes.
    :param num_planes_of_type: A list containing the number of planes
for each type.
    :param all_regular_shipments: A list of all regular shipments.
    :param all_special_shipments: A list of all special shipments.
    :return: A list of initialized solutions.
    """
    # Initialize an empty list for the population
    population = []

    # For each solution in the population
    for _ in range(population_size):
        # Initialize an empty solution
        solution = initialize_empty_solution()

        # For each plane type in the solution
        for plane_type in range(num_plane_types):
            # For each plane of the same type in the solution
            for plane_id in range(num_planes_of_type[plane_type]):
                # Schedule regular shipments for the plane
                for shipment in all_regular_shipments:
```

```
                if can_insert_shipment(solution, plane_type, plane_id,
shipment):

solution[plane_type][plane_id]['shipments_sequence'].append(shipment)
                    update_plane_state(solution, plane_type, plane_id,
shipment)

        # For each plane type in the solution
        for plane_type in range(num_plane_types):
            # For each plane of the same type in the solution
            for plane_id in range(num_planes_of_type[plane_type]):
                # Schedule special shipments for the plane
                for shipment in all_special_shipments:
                    if can_insert_special_shipment(solution, plane_type,
plane_id, shipment):

solution[plane_type][plane_id]['shipments_sequence'].append(shipment)
                    update_plane_state(solution, plane_type, plane_id,
shipment)

        # Add the initialized solution to the population
        population.append(solution)

    return population
```

Додаток 2.

```
function insert_shipment(plane: Plane, shipment: Shipment,
shipments_sequence: List[Shipment]) -> None:
    if can_insert_shipment(plane, shipment, shipments_sequence):
```

```
        # Find the right spot in the sequence based on the pickup time
        index = find_insertion_index(shipments_sequence, shipment)
        # Insert the shipment into the sequence at the correct position
        shipments_sequence.insert(index, shipment)
    else:
        raise ValueError("Cannot insert shipment")

function can_insert_shipment(plane: Plane, shipment: Shipment,
shipments_sequence: List[Shipment]) -> bool:
    # Check if the plane can carry the shipment given its volume and
weight constraints
    if shipment.volume > plane.max_volume or shipment.weight >
plane.max_weight:
        return False

    # Check if the plane can reach the destination before the delivery
deadline
    if plane.current_time + calculate_flight_time(plane.current_location,
shipment.source_airport) > shipment.pickup_time:
        return False
    if plane.current_time + calculate_flight_time(plane.current_location,
shipment.source_airport) +
       calculate_flight_time(shipment.source_airport,
shipment.destination_airport) > shipment.delivery_deadline:
        return False

    # Check if the plane can fit the shipment into its current schedule
    for i in range(len(shipments_sequence) - 1):
        # If the new shipment has to be delivered before the next shipment
in the sequence can be picked up,
```

```
        # or if the new shipment cannot be picked up before the current
shipment in the sequence is delivered,
        # then the new shipment cannot be inserted
        if (shipments_sequence[i].delivery_deadline >
shipment.pickup_time and
            shipments_sequence[i + 1].pickup_time <
shipment.delivery_deadline):
            return False

    return True

function find_insertion_index(shipments_sequence: List[Shipment],
shipment: Shipment) -> int:
    for i in range(len(shipments_sequence)):
        if shipments_sequence[i].pickup_time > shipment.pickup_time:
            return i
    return len(shipments_sequence)
```

Додаток 3

```
function crossover(parent1: Individual, parent2: Individual) ->
Individual:
    child = Individual()
    # Copy the planes from one of the parents
    child.planes = copy.deepcopy(parent1.planes)
    for plane in child.planes:
        # Clear the schedule of each plane
        plane.shipments_sequence = []
```

```
    # Iterate over all shipments in the parent's schedules in chronological order
    parent_shipments = sorted(parent1.shipments_sequence + parent2.shipments_sequence, key=lambda x: x.pickup_time)
    for shipment in parent_shipments:
        # Try to insert the shipment into the schedule of the appropriate plane in the child
        if isinstance(shipment, RegularShipment):
            # Regular shipments cannot be rejected
            insert_shipment(child.planes[shipment.plane_type], shipment, child.planes[shipment.plane_type].shipments_sequence)
        elif isinstance(shipment, SpecialShipment):
            # Special shipments can be rejected
            if can_insert_shipment(child.planes[shipment.plane_type], shipment, child.planes[shipment.plane_type].shipments_sequence):
                insert_shipment(child.planes[shipment.plane_type], shipment, child.planes[shipment.plane_type].shipments_sequence)

    return child
```

Додаток 4

```
function mutate(individual: Individual, mutation_rate: float, special_shipments: List[SpecialShipment]):
    for plane in individual.planes:
        # We mutate the individual by trying to insert each special shipment into the plane's schedule
        for special_shipment in special_shipments:
            # If the special shipment can be inserted without disrupting any regular flights
```

```
            if can_insert_shipment(plane, special_shipment,
plane.shipments_sequence):
                # With a certain mutation rate, we insert the special shipment
into the schedule
                if random() < mutation_rate:
                    insert_shipment(plane, special_shipment,
plane.shipments_sequence)
```

Додаток 5

```
function replacement(population: List[Individual], offspring:
List[Individual], elitism_size: int):
    # Sort the population and the offspring by their fitness
    sorted_population = sort_population_by_fitness(population)
    sorted_offspring = sort_population_by_fitness(offspring)

    # Keep the best individuals from the current population
    new_population = sorted_population[:elitism_size]

    # Fill the rest of the new population with the offspring
    new_population += sorted_offspring[:(len(population) - elitism_size)]

    return new_population
```

Додаток 6

```
function has_termination_condition_met(current_generation: int,
                                       best_individual: Individual,
                                       best_fitnesses: List[float],
```

```
                                max_generations: int,
                                max_stagnation_generations: int,
                                satisfactory_fitness: float) -> bool:

    # Termination condition 1: Has the algorithm run for the max number of generations?
    if current_generation >= max_generations:
        return True

    # Termination condition 2: Has the fitness of the best individual not improved for a number of generations?
    if len(best_fitnesses) >= max_stagnation_generations:
        if best_individual.fitness == best_fitnesses[-max_stagnation_generations]:
            return True

    # Termination condition 3: Has the fitness of the best individual reached a satisfactory level?
    if best_individual.fitness >= satisfactory_fitness:
        return True

    # If none of the termination conditions are met, return False
    return False
```