УДК 004.9: 519.876

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження та аналіз ефективності методів розв'язування задач на графах у системному проектуванні
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-19-2*
Спеціальність *122 Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо-наукова*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)

Паречин В.П.
(прізвище, ініціали)

Керівник *Калита Н.І.*
(посада, прізвище, ініціали)

Допускається до захисту

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

2021 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

(повна назва)

Кафедра *Системотехніки*

(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 Комп'ютерні науки*

(код і повна назва)

Тип програми *освітньо-наукова*

(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«____»_____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Паречину Владлену Петровичу*

(прізвище, ім'я, по батькові)

1. Тема роботи *«Дослідження та аналіз ефективності методів розв'язування задач на графах у системному проектуванні»*

затверджена наказом по університету від « *23* » *03* 20*21* р. № *389 Ст*

2. Термін подання студентом роботи (проекту) *18 травня 2021 р.*

3. Вихідні дані до роботи (проекту) *Функція: рішення задачі PDP для одного транспортного засобу. Організація даних: файлова система ntfs. Форма діалогу: веб-інтерфейс в інтерактивно-графічному режимі. Перелік використовуваних програмних засобів: ОС Microsoft Windows 10, Intellij IDE. Технічне забезпечення: IBM-сумісний ПК Core 2 Duo та вище.*

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) *Вступ, Задача PDP, Класифікація PDP, Постановка мети та завдань дослідження, Аналіз існуючих методі рішення задач, розробка ітеративного методу рішення задачі, Опис основних функцій програмного засоб,. Аналіз результатів, Висновки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів) *5.1 Класифікація PDP, 5.2 Пошуковий простір: локальні та глобальний мінімум, 5.3 Зведення до задачі PDP, 5.4 Додавання нової точка до шляху, 5.5 Додавання нового замовлення, 5.6 Додавання нового замовлення – неоптимальний варіант, 5.7 Додавання нового замовлення – оптимальний варіант, 5.8 Схема розробленого алгоритму, 5.9 Схема алгоритму оцінки зміни маршруту, 5.10 Інтерфейс роботи з графом, 5.11 Інтерфейс роботи з мапою*

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	Проф. Калита Н. І.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання, аналіз завдання, уточнення*	*23.03.2021*	
	плану роботи		
2	*Аналіз сучасного стану проблеми*	*01.04.2021*	
	пікапу та достави		
3	*Огляд існуючих методів рішення проблеми*	*06.04.2021*	
4	*Розробка математичного забезпечення задачі*	*13.04.2021*	
5	*Розробка програмного забезпечення задачі*	*20.04.2021*	
6	*Проведення експериментальних досліджень*	*28.04.2021*	
7	*Оформлення пояснювальної записки*	*03.05.2021*	
8	*Підготовка презентації*	*09.05.2021*	
9	*Подання закінченої роботи науковому*	*14.05.2021*	
	керівникові		
10	*Подання роботи на рецензування*	*15.05.2021*	
11	*Попередній захист*	*15.05.2021*	
12	*Подання роботи до екзаменаційної комісії*	*18.05.2021*	

Дата видачі завдання *23 березня 2021 р.*

Студент _____ *Паречин В.П.*

(підпис)

Керівник роботи _____ *проф. Калита Н.І.*

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 68 с., 17 рис., 2 додатки, 33 джерел інформації.

СИСТЕМНЕ ПРОЕКТУВАННЯ, ЗАДАЧА МАРШУТИЗАЦІЇ, PDP, 1-PDP, ОПТИМАЛЬНИЙ ШЛЯХ, ПОШУК ШЛЯХУ, ЗАДАЧА КОМІВОЯЖЕРА, TSP, АЛГОРИТМИ НА ГРАФАХ, JAVASCRIPT, NODE JS

Об’єкт дослідження – задача маршрутизації типу PDP.

Предмет дослідження – методи формування найкоротшого шляху в задачах PDP з додатковими умовами, які розв’язуються в логістичних інформаційних системах в онлайн режимі.

Мета кваліфікаційної роботи – розробка методу пошуку оптимального шляху на графі, який проходить через задані точки та задовольняє додатковим умовам; розробка програмного засобу побудови оптимального шляху онлайн, починаючи з будь-якої точки маршруту.

Методи дослідження – теорія систем в задачах проектування, теорія графів, математичні методи оптимізації, сучасні парадигми програмування і моделювання складних систем, методи проектування додатків з алгоритмами на графах, методи об’єктно-орієнтованої розробки, методи розробки, базовані на функціональному підході.

Результати кваліфікаційної роботи – програмний комплекс, контейнеризований разом з набором програм для підтримки системи, який складається з серверної та клієнтської частин. Серверний застосунок реалізує пошук оптимального шляху на графі за заданими параметрами та обмеженнями, клієнтська надає користувачеві інтерфейс для роботи з сервером. Програмний комплекс представляє з себе веб сторінку.

Результати дослідження та програмний комплекс призначені для використання в логістичних інформаційних системах компаній, які займаються доставкою товарів та виконанням замовлень на перевезення.

ABSTRACT

Master’s Thesis: 68 pages, 17 figures, 2 appendices, 33 sources.

ROUTING, PDP, 1-PDP, OPTIMAL PATH, PATH OPTIMIZATION, TRAVELLING SALESMAN PROBLEM, TSP, GRAPH ALGORITHMS, JAVASCRIPT, NODE JS

The object of this research is an optimal for finding the optimal path with given constraints.

The purpose of this research is to develop a web application that allows its users to find the optimal path to deliver all orders.

Development methods - methods of designing applications with algorithms on graphs, methods of object-oriented development, development methods based on a functional approach.

The results of the work are two software tools, containerized together with a set of programs to support the system. The first software tool is the server part, which is responsible for finding the optimal path on a graph for the specified parameters and constraints. The first application has only an API access interface. The second software tool is designed to provide a user with an interface to work with the server. The software is a web page.

Field of application - the software product is used by companies engaged in the delivery of goods and orders.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ОШ – оптимальний шлях.

ІС – Інформаційна система.

КСА – Клієнт-серверна архітектура.

ПЗ – Програмне забезпечення.

PDP – Pickup and Delivery Problem

1-PDP – Single Vehicle PDP

GPDP – General PDP

ТЗ – транспортний засіб

ВСТУП

Розробка ефективних інструментів підтримки прийняття рішень, які можуть бути застосовані у транспортній галузі, є життєво важливою у світі, оскільки це може призвести до значного зменшення витрат та ефективного споживання ресурсів. Вирішення проблеми маршрутизації транспортних засобів (VRP) та пов’язаних з нею варіантів є основою наукових досліджень для оптимізації логістичного планування. Одним з важливих варіантів VRP є проблема пікап та доставки (PDP) [1]. У PDP, як правило, потрібно знайти один або кілька оптимальних маршрутів для обслуговування усіх клієнтів. Застосування PDP часто зустрічаються у повсякденних транспортних та логістичних послугах, і проблема, ймовірно, набуде ще більшої популярності в майбутньому через збільшення електронної комерції та Інтернет-магазинів.

В атестаційні роботі розглядається задача маршрутизації транспортного засобу для виконання низки замовлень. З такою задачею можна зустрітися там, де необхідно перевозити деякі товари або людей. Наприклад, розподіл замовлень між таксистами, доставка їжі, пошта, кур'єрські служби тощо.

Об’єкт дослідження – задача маршрутизації типу PDP.

Предмет дослідження – методи формування найкоротшого шляху в задачах PDP з додатковими умовами, які розв’язуються в логістичних інформаційних системах в онлайн режимі.

Мета кваліфікаційної роботи – розробка методу пошуку оптимального шляху на графі, який проходить через задані точки та задовольняє додатковим умовам; розробка програмного засобу побудови оптимального шляху онлайн, починаючи з будь-якої точки маршруту.

На даний час сформульовано низку задач PDP, які включають декілька транспортних засобі та один, динамічні та статичні, з часовими обмеженнями та з обмеженнями в завантаженості транспортного засобу, розроблені відповідно математичні методи їх розв’язування. З огляду на практичне застосування

рішень задач маршрутизації, то розроблені та успішно використовуються в різних ситуаціях різноманітні програмні сервіси Google Maps, 2ГІС тощо.

Для розв'язування динамічної задачі PDP з одним транспортним засобом без обмежень на завантаження інтерес представляють три методи, а саме [2]:

- динамічне програмування – метод запропонований Harilaos N. Psaraftis в якому оптимальний шлях знаходиться рекурсивно;

- гілок та меж – метод запропонований Алісой Ленд і Елісон Дойг для вирішення задачі цілочисельного програмування. Розглянутий метод базується на цьому алгоритмі ;

- гілок та відсічень – метод базується на методі "гілок та меж" і алгоритму Горі. Для використання цього методу задача PDP зводиться до задачі лінійного цілочисельного програмування.

Для досягнення мети кваліфікаційної роботи необхідно розглянути питання. По перше, аналіз предметної області та задачі. Необхідно сформувати задачу в термінах предметної області. По друге, необхідно ознайомитися з існуючими підходами, методами та алгоритмами. Далі, за допомогою отриманих знань, необхідно рішити задачу. В контексті цієї роботи – це розробка алгоритму рішення задача PDP для одного транспортного засобу та розробка програмного комплексу з використанням розробленого алгоритму. Під кінець, необхідно провести аналіз отриманих результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Рішення будь-якої задачі в системному проектуванні починається з її осмислення та уточнення вхідних та вихідних даних. Завдання необхідно перевести на мову предметної області, сформулювати його максимально повно і грамотно, обгрунтувати необхідність його вирішення, тобто сформулювати технічне завдання, - перший і обов'язковий етап роботи.

Далі необхідно:

- проаналізувати методів та підходи для рішення задачі;
- обрати методу рішення;
- реалізувати метод або алгоритм рішення;
- проаналізувати отримані результати.

У цій кваліфікаційній роботі частина проектованої системи – підсистема рішення математичної задачі оптимізації. Завдання побудови оптимального маршруту – класична задача комбінаторики. Щоб наочніше показати її значимість, нижче наведена коротка історія вирішення поставленого завдання.

Ірландський математик Вільям Роуен Гамільтон і британський математик Томас Пенінгтон Кіркменом почали вивчати математичні проблеми, пов'язані із завданням побудови оптимального маршруту, в 1800-их роках. У 1857 Гамільтон створив гру Ікосіан, в правилах якої сказано, що учасники повинні з'єднати 20 точок додекаедру так, щоб кожна точка використовувалася не більше одного разу, також кінцева точка шляху повинна збігатися з вихідною. Ця гра лягла в основу появи гамільтонова графа [3].

Математик і економіст Карл Менджер у вперше розглянув завдання пошуку оптимально маршруту з математичної точки зору в 1930-их роках у Відні. У 1940-их статистики Мехаланобіс, Джессен, Гош і Маркс намагалися знайти застосування цій задачі в сільськогосподарському секторі, що призвело

до її популяризації – починаючи з середини 1950х роках методи розв'язання задачі стали публікуватися в наукових журналах.

Хоча завдання побудови маршруту проста для розуміння, її вкрай складно вирішити [4]. У 1972 Річард М. Короп показав, що завдання Гамільтона циклу належить до класу NP-повних задач (Nondeterministic Polynomial time), що має на увазі NP-складність завдання TSP [5]. Це відкриття дало наукове пояснення очевидною обчислювальною складністю знаходження оптимальних маршрутів.

Методи рішення TSP ставали більш складними, кількість міст зростала. Нижче представлена коротка історія етапів рішення задачі знаходження оптимальних маршрутів.

У 1954 Данциг, Фалкерсона і Джонсон видали опис методу для рішення TSP і практично проілюстрували його, вирішивши завдання з 49 містами, з'єднавши таким чином міста кожного штату Америки. У 1971 році Гельд і Карпо вирішили задачу пошуку оптимального шляху між 64 містами. Пізніше в 1977 році, Мартин Гротчел побудував шлях між 120 німецькими містами. У 1973 Лін і Керниган знайшли застосування завдання TSP в інженерії - вони поєднали 318 пунктів, отриманих в результаті різання лазером [6].

У 1987 році Голландія і Гроешел була вирішена задача з 666 містами, пізніше в цьому ж році Падберг і Рінальді знайшли оптимальний тур, зв'язуючи вже 2392 міста. У 1994 році кількість міст збільшилася до 7397, через 10 років 11 кількість міст досягло до 24978. У 2006 було отримано рішення задачі оптимального розрахунку маршрутів з 85900 містами [6].

1.1 Аналіз предметної області

У цьому розділі розглянемо важливу категорію VRP, яка стосується проблем з пікапом-та-доставкою. Ці проблеми привертають все більше уваги з кожним днем, завдяки постійній потребі в оптимізації витрат на маршрутизацію

та планування в програмах у реальному часі. Далі наведено аналіз основних характеристик проблеми PDP та загально існуюча класифікація.

Задачі маршрутизації застосовуються у багатьох сферах надання послуг. Усі ці задачі можна віднести до GPDP (General Pickup and Delivery Problem) [2]. Існують різні типи проблем з пікапом-та-доставкою та різні класифікації цих проблем, що відрізняє їх від інших задач маршрутизації транспортного засобу (VRPs). Проблеми з пікапом-та-доставкою широко застосовуються в таких сферах, як: транспортування сировини від постачальників до фабрик, доставка інтернет замовлень від продавців та їх доставка покупцям, збір та доставка їжі та напоїв, доставка пошти та посилок, розповсюдження газет та планування авіаперевезень та автобусів.

Проблеми з перевезенням замовлень (PDP) становлять важливе сімейство проблем маршрутизації [1]. Ці проблеми, як правило, визначаються на графіку на якому вершини представляють пункти завантаження або пункти призначення для різних сутностей (або товарів), що підлягають транспортуванню.. У проблемах багатьох-до-багатьох (M-M) кожен товар може мати декілька пунктів завантаження та пунктів призначення, і будь-який пункт може бути пунктом розвантаження або пунктом завантаження декількох товарів. Ці проблеми виникають, наприклад, при транспортуванні товарних запасів між роздрібними магазинами або в управлінні системами спільного використання велосипедів або автомобілів (кар шерінг). Проблеми "один-до-багатьох-до-одного" (1-M-1) характеризуються наявністю деяких товарів, що поставляються з депо багатьом споживачам, а інших товарів, що збираються у споживачів і транспортуються назад у депо . Вони застосовуються, наприклад, доставка з магазину або пошта. Вони також виникають у прямих та зворотних логістичних системах, де, крім постачання нових товарів, потрібно планувати збір використаних, несправних або застарілих товарів. Нарешті, при односторонніх (1–1) проблемах кожен товар має одне походження та єдиний пункт призначення, між якими його потрібно транспортувати. Типовим застосуванням цих проблем є перевезення вантажів

менше вантажів та міські кур'єрські операції. На рис. 1.1 зображено класифікацію задач PDP [1,2].

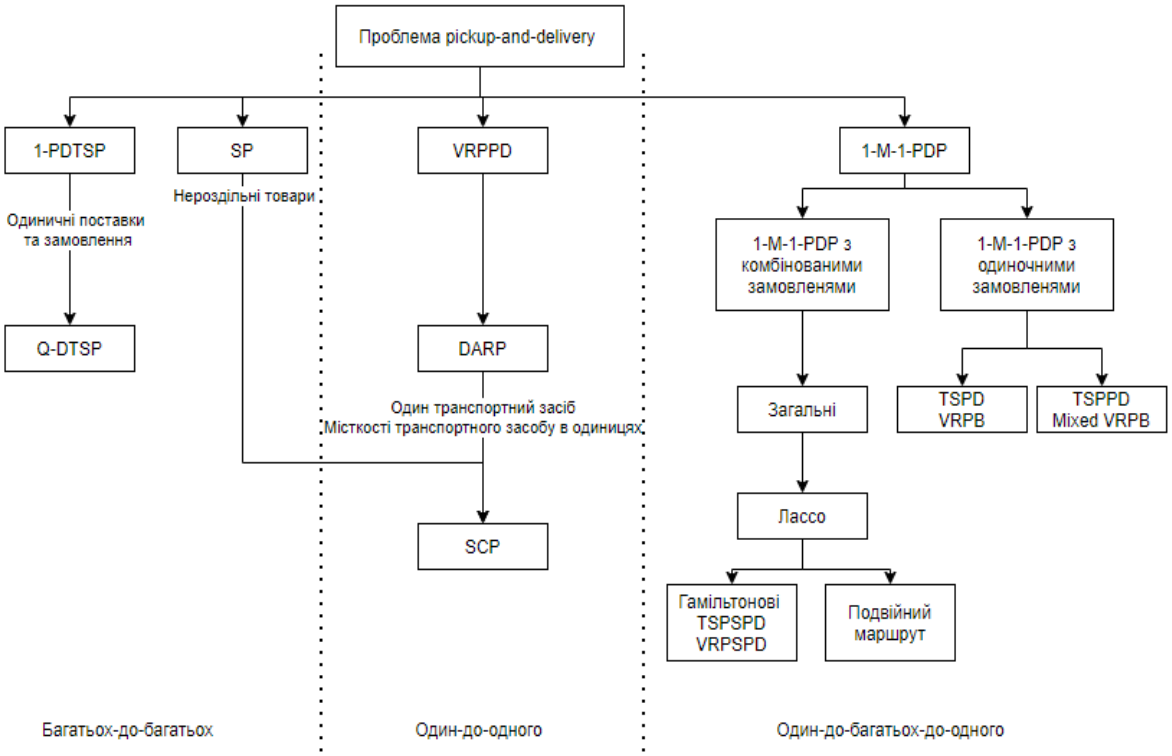


Рисунок 1.1 - Класифікація PDP

- Capacitated VRP (CVRP): кожний транспортний засіб має обмежену вантажопідйомність;

- VRP with Time Windows (VRPTW): кожен замовник повинен бути обслужений в певне «тимчасове вікно»;

- Multiple Depot VRP (MDVRP): використовуються кілька депо для обслуговування клієнтів;

- VRP with Pick-Ups and Delivering (VRPPD): клієнти можуть повертати деякі товари в депо;

- VRP with Backhauls (VRPB): аналогічно попередньої, але повернення починається тільки після доставки всіх товарів з депо;

- Split Delivery VRP (SDVRP): кожен клієнт може обслуговуватися одночасно декількома машинами;

- Periodic VRP (PVRP): доставка може здійснюватися протягом кількох днів;

- Stochastic VRP (SVRP): деякі компоненти завдання (кількість і запити клієнтів, довжина шляху) можуть мати випадкове поводження;

- VRP with Satellite Facilities (VRPSF): існує можливість дозавантаження автомобіля на маршруті.

В задачі GPDP необхідно побудувати набір маршрутів які б задовольнили усі транспортні запити [2]. Є декілька транспортних засобів які можуть виконувати транспортування. Кожен запит на транспортування визначає розмір вантажу, місця розташування вантажу та пункту призначення. Кожен вантаж повинен бути перевезений з його пункту відправлення до його пункту призначення одним транспортним засобом.

VRPSF це класична задача VRP передбачає, що кожен маршрут починається і закінчується в депо. Однією з причин повернення в депо є обмежена вантажопідйомність. Коли машина розвозить усі товари, вона повинна повернутися в депо за новою порцією товарів. Однак, в деяких випадках вигідніше провести дозагрузку на маршруті, без повернення в депо, за допомогою додаткових транспортних засобів. Типовий випадок, коли безліч споживачів очікують регулярних поставок від одного центрального постачальника. Метою є мінімізація витрати на доставку товарів за певний термін (можливо, що з огляду на витрати на допоміжні машини, загальна вартість виконання завдання в короткостроковій перспективі буде вище, ніж, наприклад, при вирішенні класичної завдання VRP). Обмеженням є те, що товар на складі клієнта не повинен закінчуватися.

SVRP - маршрутизація з випадковими даними. В даному варіанті VRP один або кілька компонентів завдання можуть мати випадкове поводження.

Випадкові клієнти: кожен клієнт існує з ймовірністю p і відсутній з ймовірністю p - 1.

Випадкові запити: запит кожного клієнта - випадкова величина.

Випадкові часи: часи поїздок (відстані між споживачами) - випадкові величини.

Рішення SVRP відбувається в два підходи. Перший етап готується рішення без урахування випадкових змінних. На другому етапі, коли стають відомими випадкові значення, відбувається корекція раніше отриманого рішення. Метою є мінімізація парку транспорта і загальний час обслуговування всіх клієнтів.

Обмеження: коли деякі дані невідомі, стає неможливим виконання всіх обмежень для всіх випадкових змінних. Таким чином, може вимагатися виконання деяких умов із заданою вірогідністю, або побудова коректує моделі, що виконується при порушенні будь-яких обмежень. Наприклад, в завданню SVRP з поверненням товарів і обмеженням по місткості, можливими способами корекції будуть наступні:

- повернутися в депо, коли машина заповниться, для розвантаження, потім продовжити шлях по маршруту;

- повернутися в депо, коли машина заповниться, і заново оптимізувати решту шляху;

- запланувати дострокове повернення в депо, навіть якщо машина заповнена не до кінця. В даному випадку, рішення може залежати від кількості зібраного вантажу, і відстані до депо. Машина може повернутися в депо, якщо відомо, що вантаж наступного споживача перевищить місткість автомобіля.

Проблема Dial-a-Ride (DARP) - це PDP, в якому вантажі, що транспортуються, представляють людей. Тому ми зазвичай говоримо про клієнтів або замовників, а не про транспортні запити, і всі розміри вантажів дорівнюють одиниці [8]. Проблема маршрутизації транспортного засобу (VRP) - це PDP, в якому в пункті зберігання знаходяться або всі джерела відправлення, або всі пункти призначення.

У цій роботі буде детально розглянуто пошук оптимального шляху для одного транспортного засобу 1-PDP.

1.2 Постановка задачі дослідження

Розглянемо таку задачу. Кур’єр починає робочий день у депо. На початку відомий список запитів на дрібнопартійні перевезення. Запит являє собою адреса отримання та доставки товару. Кур’єрові необхідно виконати всі запити на транспортування. Під час їх виконання нові замовлення можуть з’являтися у будь-який час. Кожного разу, коли з’являється нове замовлення, решта вже побудованого шляху має бути зміненою для того, щоб включити виконання нового замовлення. Новий шлях повинен бути найкоротшим. Усі невиконанні заявки переносяться на наступній робочий день. Під кінець робочого дня кур’єр повинен повернутися у депо.

Для вирішення цієї задачі потрібно:

- дослідити предметну область, ознайомитися з проблемами пікапу та доставки (PDP) та їх класифікацією – проблемою до якої належить поставлена задача;
- розглянути можливі постановки задач, можливі параметри, критерії оптимізації, обмеження тощо;
- розробити формальну постановку задачі;
- проаналізувати існуючі методи та алгоритми;
- виходячи з отриманих знань розробити метод рішення поставленої задачі;
- розробити програмний комплекс, який дозволяє вирішувати поставлену задачу на мапі або задовільному графі;
- провести обчислювальні експерименти, виконати порівняльний аналіз алгоритмів.

1.3 Характеристики задачі PDP

1.3.1 Запити на транспортування

Однією з найважливіших характеристик проблем маршрутизації є доступність запитів на транспортування. У статичній ситуації всі запити відомі під час побудови маршрутів. У динамічній ситуації деякі запити відомі під час побудови маршрутів, а інші запити з’являться в режимі реального часу протягом виконання маршрутів. Отже, в динамічній ситуації, коли з’являється новий запит на транспортування, принаймні один маршрут потрібно змінити, щоб включити обслуговування нового запиту [9]. Більшість проблем з маршрутизацією транспортних засобів є статичними, тоді як більшість проблем з PDP є динамічними.

На практиці динамічні задачі часто розбиваються на підзадачі та вирішується як послідовність статичних задач. При чому знання о просторовому та часовому розподілу майбутніх запитів ігноруються. Тобто кожен раз коли з’являється запит на транспортування та необхідно включити його до маршруту розглядаються тільки уже відомі запити.

Інша важлива концепція маршрутизації – концепція депо. У літературі про маршрутизацію депо зазвичай є місцем, де транспортні засоби починають і закінчують свої маршрути. Оскільки більшість проблем з PDP є динамічними, часто з довгим горизонтом планування, концепція депо зникає. Водії сплять в останньому місці яке вони відвідали або у першому місті які вони мають відвідати наступного дня. Навіть для проблем із коротким горизонтом планування, наприклад, одного дня, коли транспортні засоби починають і закінчують роботу у центральному депо, запити, які з’являються в реальному часі, призводять до проблем без депо.

Алгоритми та методи рішень задач маршрутизації де транспортні засоби починають та закінчує маршрут у депо можуть бути вирішені алгоритмами та методами для задач без депо. Також це є вірним і навпаки – методи без депо

можуть бути використаними для вирішення задач з депо. Це можливо тому що будь яку задачу з депо можна трансформувати у задачу без депо і навпаки за лінійний час.

1.3.2 Обмеження в часі

Окрім обмежень пов’язаних з вантажопідйомністю транспортного засобу, майже у кожній практичній ситуації в PDP виникають обмеження в часі. Хоча обмеження часу стали невід’ємною частиною моделей для проблем маршрутизації транспортних засобів (VPR), вони відіграють ще виднішу роль у проблемах пікап-та-доставки. Однією з причин є те що одна з найбільш вивчених проблем PDP, Dial-a-Ride Problem, має справу з запитами у яких вказано бажаний час для відправлення і отримання замовлення.

Наявність обмежень у часі значно ускладнює проблему [10]. Якщо обмежень у часі немає, пошук оптимального маршруту для виконання усіх замовлень спрощується: достатньо розподілити транспортні запити між усіма транспортними засобами, довільно упорядковувати транспортні запити, призначені транспортним засобам, та обробляти кожен транспортний запит окремо. За наявності обмежень у часі проблема пошуку оптимального плану прийому та доставки є NP-складною. Отже, може бути важко побудувати здійсненний план. З іншого боку, метод оптимізації може отримати вигоду від наявності часових обмежень, оскільки простір рішення може бути набагато меншим.

В задачах PDP з обмеженнями у часі часто застосовуються наступні обмеження:

- рішення є неприйнятним якщо клієнт обслуговується після часового обмеження;
- транспортний засіб який прибув раніше нижньої часової границі очікує на її настання;

- у деяких задачах, запізнення транспортного засобу на замовлення не впливає на прийнятність рішення, а додає неяке штрафне значення до цільової функції.

Отримавши рішення задачі PDP з обмеженнями у часі є можливість точніше підібрати час виїзду транспортного засобу з депо ы тим самим уникнути уникнути простоїв.

Інколи розглядаються обмеження у часі які відносяться до транспортних засобів. Транспортні засоби зазвичай недоступні протягом усього дня. Водії повинні їсти і спати, а також транспортні засоби підпорядковуються планам обслуговування. Ці обмеження можна змоделювати як часові вікна для транспортних засобів. Зазвичай транспортний засіб має кілька часових вікон, що визначають всі періоди, в які він доступний.

1.3.3 Обмеження по вантажопідйомності

У завдання цього типу вводиться додаткове обмеження: кількість або вага вантажів на кожному маршруті R_i не повинен перевищувати деякої заданої величини Q (однакової для всіх транспортних засобів). У деяких задачах, додатково до вантажопідйомності, враховується порядок завантаження замовлень. Наприклад, транспортний засіб може доставити тільки такий вантаж, який був завантаженим останнім або за доставки вантажу який був завантаженим не останнім вводиться штраф у часі.

У деяких задачах окрім ваги замовлення, враховують інші параметри. Так, наприклад, в задачі CVRP з тривимірними (3D) обмеженнями навантаження (3L-CVRP) замовник вимагає транспортування ряду паралелепіпедів (коробок) і необхідно враховувати їх розміри [7].

Метою є мінімізація парку машин, необхідних для виконання завдання, а також загальний час виконання завдання.

1.4 Цільові функції PDP

У проблемі PDP виявляється широке розмаїття цільових функцій. Розглянемо найбільш поширені та вивчені [2].

Цільові функції, які відносяться до 1-PDP:

- Мінімізація тривалості. Тривалість маршруту – це загальний час, який транспортному засобу потрібно на виконання маршруту. Тривалість маршруту включає час у дорозі, час очікування, час завантаження та розвантаження та час перерви.

- Мінімізація часу завершення. Час завершення маршруту – час коли останнє замовлення було виконано. Якщо час запуску транспортного засобу встановлений у нульову годину, час завершення збігається з тривалістю маршруту.

- Мінімізація часу у дорозі. Час у дорозі маршруту – загальний час, витрачений на подорожі між різними місцями. Час завантаження, розвантаження та очікування не враховуються.

- Мінімізація негодування клієнтів. У системах Dial-a-Ride негодування клієнта вимірюються з точки зору відхилення часу відправлення, тобто різниці між фактичним часом відправлення та бажаним часом, відхилення часу доставки, тобто різницею між бажаним часом доставки та фактичним часом доставки та надлишкового часу поїздки, тобто різниці між реалізованим часом їзди та прямим часом їзди. В ситуаціях, коли клієнти вимагають негайного обслуговування, різниця між часом отримання та часом розміщення запиту може також сприяти визначенню негодування клієнтів. Для моделювання незручностей клієнта було запропоновано різні типи функцій, як лінійних, так і нелінійних.

Далі наведені найбільш розповсюджені цільові функції для задач PDP з декількома транспортними засобами.

- Мінімізація використання транспортних засобів. Ця функція майже завжди використовується в системах Dial-a-Ride, наприклад таксі, у поєднанні з однією з вищезазначених функцій для оптимізації підпроблем з використанням одного автомобіля. Метою є мінімізація витрат (в основному разом із рівнем незадоволеності клієнтів). Завичай, транспортні засоби та їх водії є найдорожчими деталями системи. Тому, як правило, мінімізація кількості транспортних засобів для обслуговування всіх замовлень є основною метою задачі [11].

- Максимізація прибутку. Ця функція, яка може комбінуватися з усіма вищезазначеними функції, може бути використана в системі, де диспетчер має можливість відхилити транспортний запит, коли транспортування відповідного вантажу є несприятливим. Наприклад, у системі Dial-a-Ride не дозволяється відхиляти запит на перевезення. Модель, заснована на цій цільовій функції, повинна включати не лише витрати, але й доходи, пов’язані з перевезенням вантажів.

Що стосується проблем динамічної PDP, незрозуміло, які об’єктивні функції слід використовувати. В умовах, коли в будь який час можуть з’являтися нові замовлення, такі цільові функції, як тривалість маршруту, час завершення і час у дорозі не мають чіткого визначення. Інтуїтивно, цільова функція для динамічних проблем повинна наголошувати на рішеннях, які впливають на найближче майбутнє, аніж на рішення щодо віддаленого майбутнього. Зауважимо, що якщо динамічна задача вирішується як послідовність статичних задач, цільова функція для статичної підзавдання не обов’язково повинна дорівнювати цільовій функції для динамічної задачі. Цільова функція для статичної підзадачі може містити передбачення майбутніх запитів.

1.5 Огляд сучасних підходів до розв'язування задачі маршрутизації

1.5.1 Speedy Route

Speedy Route - картографічних сервісів з функцією побудови оптимального маршруту який проходить через декілька точок (Рисунок 1.2). В описі продукту сказано, що сервіс розраховує найбільш ефективний по кількості витрат палива маршрут між декількома локаціями, де вихідна і кінцева точка єдині [12].

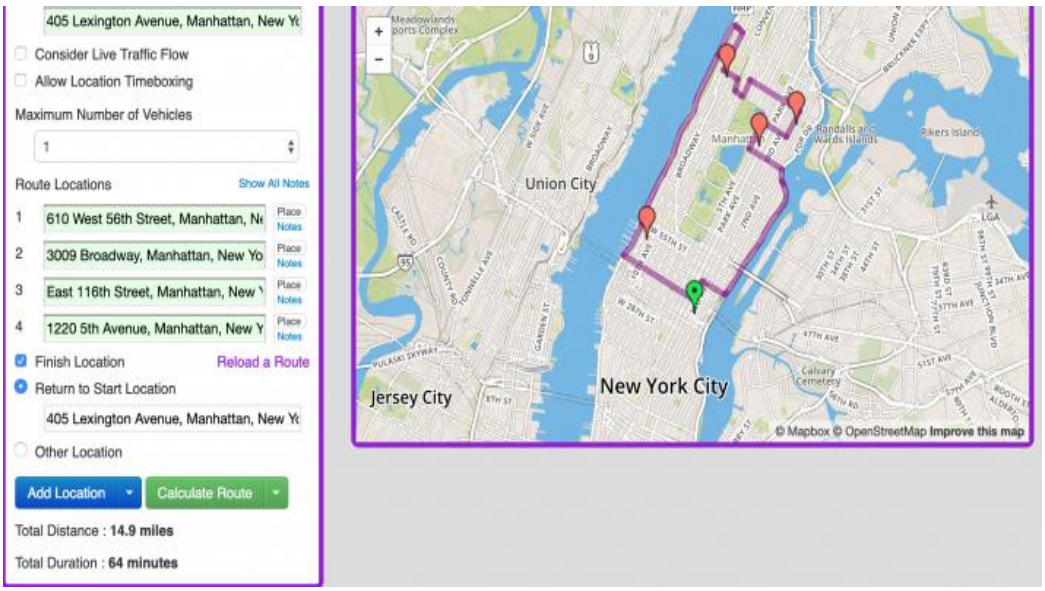


Рисунок 1.2 - Інтерфейс Speedy Route

В першу чергу можна виділити такі недоліки як:

- Speedy Route розрахований виключно для автомобілістів;
- при введенні даних виникають труднощі перекладу російських та українських назв на англійську мову;
- у сервісу передбачено мінімальну кількість пунктів маршруту - 5, маршрути, що складаються з 4 пунктів побудувати неможливо;
- на сайті представлена урізана версія карти для ознайомлення, для доступу до повної версії потрібно оформити платну підписку.

1.5.2 Вирішувачі оптимізаційних задач

Існує велика кількість вирішувачів оптимізаційних задач, наприклад [13-16]. У таких вирішувачах зазвичай задається трансформація вхідних даних за допомогою математичної функції f, а далі йде генерація та вибір найкращого рішення з деякого набору доступних альтернатив шляхом систематичного вибору вхідних значень з дозволеного набору, обчислення результату функції, та запис найкращих вихідних значень, знайдених під час процесу. Багато реальних проблем можна змоделювати таким чином. Наприклад, вхідними даними можуть бути конструктивні параметри двигуна, вихідними показниками можуть бути енергоспоживання, або вхідними даними можуть бути бізнес-рішення, а вихідними показниками може бути отриманий прибуток. Задача PDP може бути вирішена таким чином.

Протягом останніх кількох років оптимізатори все більше привертають увагу дослідницьких кіл завдяки своїй здатності справлятися з великою кількістю обмежень. Вони забезпечують кращий спосіб роботи суперечливими обмеженнями, а також цільовими функціями. Для вирішення складних задач розроблена велика кількість оптимізаційних вирішувачів. Дизайн та розробка цих вирішувачів залежить від характеру конкретної проблеми, що підлягає вирішенню. Існує велика кількість вірішувачів оптимізаційних задач. Серед них комерційні:

- Оптимізатор Gurobi - це комерційний вирішувач оптимізації для лінійного програмування (LP), квадратичного програмування (QP), квадратично-обмеженого програмування (QCP), змішаного цілочисельного лінійного програмування (MILP), змішаного цілочисельного квадратичного програмування (MIQP) та змішаного цілочисельного квадратично обмеженого програмування (MIQCP) [13].

- CPLEX (IBM ILOG CPLEX Optimization Studio) - це програмний пакет для оптимізації. Оптимізатор вирішує завдання цілочисельного програмування, дуже великі задачі лінійного програмування, використовуючи або первинні, або

подвійні варіанти симплексного методу, або метод бар'єрної внутрішньої точки, опуклі та неопуклі квадратичні завдання програмування, а також опуклі квадратично обмежені задачі (вирішені за допомогою другого програмування конусів замовлення, або SOCP) [14].

Декілька оптимізаторів з відкритим вихідним кодом:

- SCIP (Solving Constraint Integer Programs) - вирішувач базується на алгоритмах "branch and cut" та "branch and price". Був розроблений в Інституті Цузе в Берліні. На відміну від більшості комерційних вирішувачів, SCIP надає користувачеві низькорівневий контроль та інформацію про процес вирішення. Запущений як повноцінний вирішувач, це один з найшвидших некомерційних вирішувачів для змішаних цілочисельних програм [15].

- GLPK (GNU Linear Programming Kit) - це програмний пакет, призначений для вирішення задач лінійного програмування, змішаного цілочисельного програмування та інших супутніх проблем. Являє собою набір процедур, написаних на ANSI C та організованих у формі бібліотеки, що викликається [16].

Усі наведені вирішувачі можуть бути використані для пошуку оптимальних маршрутів в задачах PDP. Один з недоліків такого підходу, це те що такі вирішувачі низькорівневі та не можуть бути використані звичайним користувачем, який хоче знайти оптимальний шлях.

Для рішення задачі, використовуючи запропоновані вирішувачі, її необхідно математично сформулювати - задати цільову функцію та множину обмежень. Більшість вирішувачів мають різні формати даних для опису задачі. Деякі з них можуть бути використані тільки у програмному коді, тобто програмістами.

Нижче наведено приклад рішення задачі лінійного програмування в CPLEX використовуючи інтерактивний інтерфейс.

```
CPLEX> enter example
Enter new problem ['end' on a separate line terminates]:
maximize x1 + 2 x2 + 3 x3
```

```
subject to -x1 + x2 + x3 <= 20
x1 - 3 x2 + x3 <=30
bounds
0 <= x1 <= 40
0 <= x2
0 <= x3
end
CPLEX> optimize
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time = 0.00 sec. (0.00 ticks)
Iteration: 1 Dual infeasibility = 0.000000
Iteration: 2 Dual objective = 202.500000
Dual simplex - Optimal: Objective = 2.0250000000e+002
Solution time = 0.01 sec. Iterations = 2 (1)
Deterministic time = 0.00 ticks (3.38 ticks/sec)
CPLEX> display solution variables x1-x3
Variable Name Solution Value
x1 40.000000
x2 17.500000
x3 42.500000
```

Незважаючи на зазначений недолік, будь який вирішувач може бути використаний у програмному засобі для пошуку оптимальних шляхів. Таким чином не потрібно програмувати складні алгоритми, достатньо сформувати задачу, як цього потребує вирішувач, відправити її на вирішувач та показати результат користувачеві.

1.5.3 Консультація з оптимізації маршруту

Існує багато консалтингових компаній які допомагають з оптимізацією маршрутів. Компанії які надають тільки консалтингові послуги більше підходять

для вирішення статичних задач маршрутизації. Наприклад ми маємо декілька відділень пошти у декількох містах. Необхідно спланувати маршрути між усіма відділеннями там мінімізувати витрати на це. Недоліком таких сервісів є те, що вони не можуть бути використаними для PDP. PDP у більшості є динамічною, замовлення на транспортування можуть з’являтися і зникати, а маршрути повинні бути динамічно оновлені.

Приклади таких сервісів: “Route Optimization Consultants”, “Paul Trudgian” [17,18].

Деякі з них пропонують сервіси для динамічного прокладання оптимальних маршрутів. Такі сервіси можуть бути використані для рішення розглянутої задачі пікапу та достави для одного транспортного засобу. Наприклад “Optimo Route” (Рисунок 1.3).

Сервіс надає мобільний додаток для водіїв та кур’єрів. Також в наявності є Web API, що надає можливість інтегрувати сервіс у свій програмний додаток. До недоліків можно віднести ціну яку необхідно платити за кожного водія/кур’єра, або за кожен виконаний заказ.

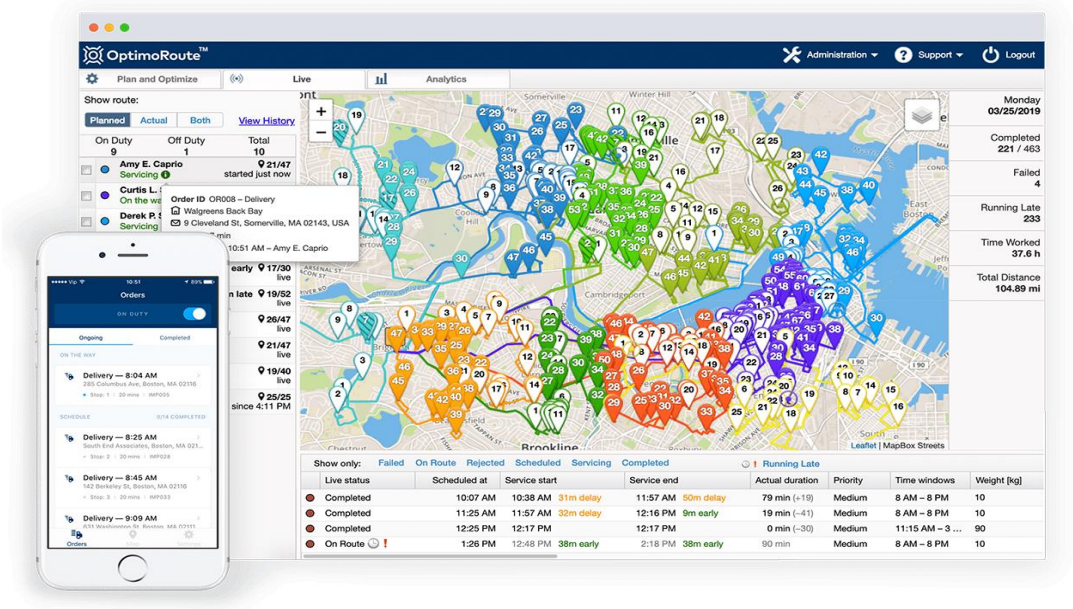


Рисунок 1.3 - Інтерфейс Optimo Route

1.6 Математична постановка задачі

Формальна постановка задачі має наступний вигляд [19]. Задано простий граф G(V, E), де V – множина вершин, E – множина ребер. Відома вагова функція $f: E \to R$, яка відображає ребро e_i на його вагу r_i. Є N запитів на транспортування. Нехай запит $q_i = (a_i, b_i)$ де a_i- вершина в якій треба отримати вантаж, та b_i- вершина, куди необхідно доставити вантаж. Позначимо через A={a_i} і B={b_i} $i = 1, n$ відповідно множини вершин, які відповідають пунктам відправлення та пунктам споживання, причому

$$A \in V, B \in V, A \cap B = \emptyset$$

Необхідно знайти оптимальний шлях $P(v_1, v_2, v_3, \dots, v_n)$ на графі G такий, що

$$\sum_{i=1}^{n-1} f(v_i, v_{i+1}) \to min \qquad (1.1)$$

за умов:

$$A \cup B \in P \qquad (1.2)$$

Цільова функція (1.1) – мінімізація суми ваг дуг які входять до маршруту. В обмежені (1.2) показано, що всі точки замовлень (точки завантаження та розвантаження) мають бути у побудованому маршруті.

Також необхідно переконатися що для кожного замовлення точка завантаження знаходиться у маршруті перед точкою розвантаження. Це можна показати наступним чином.

Нехай

o_i буде таким, що $P_{o_i} = a_i$,

d_i буде таким, що $P_{d_i} = a_i$,

тоді

$$o_i < d_i \quad i = 1, \dots, n.$$

2 ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

2.1 Огляд існуючих методів розв’язування задачі пошуку оптимального шляху

2.1.1 Динамічне програмування

Динамічне програмування — розділ математики, який присвячено теорії і методам розв'язання багатокрокових задач оптимального управління [20].

У динамічному програмуванні для керованого процесу серед множини усіх допустимих управлінь шукають оптимальне у сенсі деякого критерію тобто таке, яке призводить до екстремального (найбільшого або найменшого) значення цільової функції — деякої числової характеристики процесу. Під багатоступеневістю розуміють або багатоступеневу структуру процесу, або розподілення управління на ряд послідовних етапів (ступенів, кроків), що відповідають, як правило, різним моментам часу. Таким чином, в назві «Динамічне програмування» під «програмуванням» розуміють «ухвалення рішень», «планування», а слово «динамічне» вказує на суттєве значення часу та порядку виконання операцій в процесах і методах, що розглядаються.

Методи динамічного програмування є складовою частиною методів, які використовуються при дослідженні операцій, і використовуються як у задачах оптимального планування, так і при розв'язанні різних технічних проблем (наприклад, у задачах визначення оптимальних розмірів ступенів багатоступеневих ракет, у задачах оптимального проектування прокладення доріг та ін.)

Харілаос Н. Псарафтіс запропонував наступний метод динамічного програмування для вирішення задачі PDP для одного транспортного засобу [21]. У задачі кожен клієнт звертається за послугою по телефону, таким чином може

бути сформований хронологічний список запитів. У певний момент часу (t = 0) стає доступним один транспортний засіб. Завданням оператора транспортного засобу є виконання усіх замовлень, список яких відомий при t = 0. Іншими словами, при t = 0 наш список закривається і до нього не додаються нові запити клієнтів під час виконання маршруту. Будь-які такі запити можуть бути розміщені в іншому списку для розгляду після завершення маршруту або для планування руху іншого транспортного засобу. У «динамічному» випадку нові запити клієнтів, що виникають під час виконання маршруту, автоматично підлягають розгляду під час їх появи. Наступні два пункти будуть зроблені при формулюванні "статичної" проблеми з метою полегшення її подальшого поширення на еквівалентний "динамічний" випадок.

По-перше, маршрути транспортних засобів будуть не замкнутими шляхами, що закінчується доставкою останнього замовлення, замість повернення в депо, як у TSP.

По-друге, будуть введені спеціальні пріоритетні обмеження. Їх опис виглядає наступним чином.

У будь-якому конкретному маршруті транспортного засобу ми можемо ідентифікувати послідовність “складів” та послідовність споживачів, які, як правило, будуть об'єднані між собою. Позиція (1-а, 2-а та ін.), яку конкретний клієнт займає в послідовності пікапів, загалом не буде такою ж, як його позиція (FCFS) у початковому списку запитів клієнтів. Наприклад, клієнт, який займає п’яту позицію у початковому списку але обслуговується третім, має зміщення позиції обслуговування +2, та має зміщення -1, якщо клієнт підібраний шостим. Зміщення позиції обслуговування це різниця між позицією замовлення в листі замовлень та фактичним номером.

Згідно з нашими спеціальними пріоритетними обмеженнями для кожного клієнта, ми не допускаємо, щоб будь-який із зміщень позицій, визначених вище, був більшим за величиною, ніж встановлений невід’ємний цілий ліміт МЗП. МЗП означає максимальний зміщення положення та є параметром алгоритму.

Якщо MPS = 0, кожного клієнта слід забрати та доставити відповідно до його позиції у початковому списку. З іншого боку, для N клієнтів, $ПМЗ \geq N - 1$ означає, що наші пріоритетні обмеження зайві, і клієнти можуть бути забрані або доставлені в будь-якому порядку.

Алгоритм має наступні кроки. По-перше ми призначаємо номери клієнтам відповідно до того порядку, в якому вони телефонували в агентство для обслуговування, і розміщуємо їх у списку в тому ж порядку. Таким чином, і-тій клієнт займає і-ту позицію у списку. Нехай N - загальна кількість клієнтів. Також нехай "+i" буде пунктом відправлення, а "-i" - пунктом доставки (призначення) клієнта i (i = l, 2, … , N) та нехай A буде початковою точкою транспортного засобу (розташування транспортного засобу при t = 0). Будемо вважати, що час переходу від будь-якої з 2N + 1 точок нашої задачі, i, до будь-якої іншої точки j є відомою і фіксованою величиною $t(i,j)$. Наша мета - знайти маршрут транспортного засобу, що починається в A і закінчується в одному з пунктів доставки, та задовольнити наступні умови:

1) Мінімізувати функцію

$$w_1 \sum_{j=1}^{2N} T_j + w_2 \sum_{i=1}^{N} [aWT_i + (2-a)RT_i], \text{ де} \qquad (2.1)$$

w_1, w_2 - задані коефіцієнти

a - пріоритет клієнта ($0 \leq a \leq 2$)

T_j- Тривалість j-тої ділянки маршруту

WT_i - час з моменту замовлення до моменту завантаження

RT_i - час виконання замовлення, з моменту завантаження до моменту розвантаження

2) Маршрути транспортних засобів повинні бути законними, а саме - кожен клієнт має бути забраним до його доставки.
3) Транспортний засіб може перевозити тільки C клієнтів в один і той же час.

4) Обмеження МЗП мають бути виконаними, таким чином, якщоp_iце позиція яку займає клієнт i в послідовності пікапі та d_i - позиція, яку він займає в послідовності доставок, то для даного невід'ємного цілого числа МЗП , ми повинні мати:

$$|i - p_i| \leq МЗП \text{ для i = 1, 2, … , N,} \quad (2.2)$$

$$|i - d_i| \leq МЗП \text{ для i = 1, 2, … , N.} \quad (2.3)$$

В формулі (2.1) ми маємо $\sum_{j=1}^{2n} T_j$ - час необхідний для виконання всіх замовлень. При ($w_1 = 1, w_2 = 0$) ми маємо особовий випадок - класичну задачу маршрутизації (TSP).

Чисельник $\sum_{i=1}^{n}[aWT_i + (2 - a)RT_i]$ являє собою загальну форму "невдоволення", що виникає у клієнтів до моменту їх доставки. Час очікування та їзди загалом зважуються неоднаково (крім випадків, коли $a = 1$, коли клієнти байдужі між ними). При ($w_1 = 0, w_2 = 1$) ми мінімізуємо рівень загального незадоволення клієнтів. Мінімізація затрат ігнорується.

Наведена вище цільова функція не включає час очікування клієнтів від моменту дзвінка до моменту, коли транспортний засіб стає доступним (t = 0).

У нерівностях (2.2) та (2.3) значення МЗП можуть бути різними. Жодна з нерівностей не повинна бути двосторонньою або симетричною. Також кожен клієнт може мати власні верхні та нижні значення МЗП.

Для вирішення даної проблеми введемо поняття поточного стану $(L, k_1, \ldots, k_n)$. L - точка до якої зараз прямує транспортний засіб. При $L = 0$транспортний засіб на початковій точці A. При $1 \leq L \leq N$ транспортний засіб відвідує точку +L, тобто підбирає клієнта L. При $N + 1 \leq L \leq 2N$ ТЗ відвідує - $(L - N)$ точку, тобто виконує доставку клієнта $L - N$. Також припускається що завантаження та розвантаження ТЗ відбувається моментально.

k_j- статус j-того замовлення, при чому:

$k_j = 3$: завантаження j-того клієнта ще не відбулося

$k_j = 2$: j-тий клієнта був завантажений

$k_j = 1$: j-тий клієнта був доставлений

Вектор стану $(L, k_1, \ldots, k_N)$ може мати суперечливі стани. Щоб цього уникнути введені наступні обмеження:

1) Стан повинен бути консистентним, а саме

   a) Якщо $L = 0$, то $k_j = 3$ для будь якого j (2.4)

   b) Якщо $1 \leq L \leq N$, то $k_L = 2$ (2.5)

   c) Якщо $N + 1 \leq L \leq 2N$, то $k_L = 1$ (2.6)

2) Також береться до уваги вмістимість транспортного засобу. Нехай x_2 дорівнює кількості k зі значенням 2.

   a) Якщо $1 \leq L \leq N$, то $x_2 \leq C$ (2.7)

   b) Якщо $N + 1 \leq L \leq 2N$, то $x_2 \leq C - 1$ (2.8)

3) Обмеження МЗП повинно бути задовільнена, додатково до x_2визначеного вище, нехай x_1дорівнює кількості k зі значенням 1. Тоді обмеження МЗП можуть бути описані як:

   a) Якщо $1 \leq L \leq N$, то $|L - (x_1 + x_2)| \leq МЗП$ (2.9)

   b) Якщо $N + 1 \leq L \leq 2N$, то $|(L - N) - x_1| \leq МЗП$ (2.10)

Стани які задовольняють усі наведені обмеження вважаються можливими. Якщо немає такого наступного стану, який вважається можливим, то попередній стан стає неможливим. З іншого боку, якщо хоча б один з наступних станів є можливим, то вихідний стан також можливий. Те, що для визначення коректності будь якого стану потрібно знати коректність наступних станів говорить про рекурсивну породу алгоритму.

Множина станів $(L', k_1', \ldots, k_N')$ які є “наступними” для стану $(L, k_1, \ldots, k_n)$. Таким чином L'належить множині S, яка складається з двох множин:

1) Множина наступних точок завантаження:

$$S_3 = \{i: 1 \leq i \leq N \text{ при } k_i = 3\} \quad (2.11)$$

2) Множина наступних точок розвантаження:

$$S_2 = \{i: N + 1 \leq i \leq 2N \text{ при } k_{i-N} = 2\} \quad (2.12)$$

Наступний k-вектор виглядає як:

$$k_j' = k_{j-1} \text{ якщо } j = L \text{або } j = L - N \quad (2.13)$$
$$k_j' = k_j \text{ в іншому випадку}$$

Випадок, коли S_2 та S_3 пусті відповідає випадку де $k_j = 1$ для усіх $j = (1, \ldots, N)$ що значить що усі замовлення було виконано. Будь-який з таких станів називається кінцевим та не має наступних станів. Таким чином ми можемо перебирати можливі маршрути.

Для пошуку оптимального шляху нехай $V(L, k_1, \ldots, k_N)$ дорівнює оптимальному значенню всіх послідовних виборів з $V(L, k_1, \ldots, k_N)$ до кінця шляху. V існує тільки для коректних станів. Vповинно задовольняти наступну оптимізаційну рекурсію:

Якщо $S \neq \varnothing$, то $V(L, k_1, \ldots, k_N) = Min_{L' \in S}\left[t(L, L') \cdot M + V(L', k_1', \ldots, k_N')\right]$

Якщо $S = \varnothing$, то $V(L, k_1, \ldots, k_N) = 0$

$$M = w_1 + w_2[a \cdot x_3 + (2 - a) \cdot x_2]$$

Алгоритмічна складність алгоритму складає $(2n + 1)^2 \cdot 3^n$. Також необхідно виділити $2n \cdot 3^{n-1}$.

2.1.2 Метод гілок та відсічень

Метод гілок та відсічень (branch and cut) - є методом комбінаторної оптимізації для вирішення задач цілочисельного лінійного програмування (ILP), тобто задач лінійного програмування (LP), де деякі або всі невідомі обмежені цілими значеннями. Метод гілок та відсічень включає використання методу гілок та меж та алгоритму Гоморрі для зменшення лінійної релаксації [22].

Метод гілок і відсічень є розвитком алгоритму повного перебору. Основна ідея полягає в перевірці критерію обмежуваної функції, за яким можна призупинити побудову гілки дерева перестановок на певному рівні. Відомі модифікації даних підходів дозволяють вирішувати задачу PDP з декількома сотнями вершин, однак вони вимагають високої обчислювальної потужності комп'ютера.

Алгоритм гілок та відсічень відрізняється від традиційного методу гілок та меж для проблем лінійного програмування, тим, що він використовую алгоритм Гоморрі під час фази межування. Метод гілок та меж обчислює межі шляхом рішення релаксації цілочисельного лінійного програмування. Алгоритм гілок та відсічень використовує результат цього обчислення (дробове рішення) в алгоритмі Гоморрі, що покращує межі, зменшуючи тим самим розмір дерева пошуку.

Підсумовуючи, метод вирішує задачу лінійного програмування без цілочисельного обмеження за допомогою звичайного симплексного алгоритму. Коли отримано оптимальне рішення, що має не ціле значення для змінної яка вважається цілою, алгоритм Гоморрі може бути використаний для пошуку подальших лінійних обмежень, які задовольняються всіма можливими цілими точками, але порушують поточне дробове рішення.

Далі використовується метод гілок та границь. Проблема розділяється на кілька (зазвичай дві) версії. Отримані проблеми вирішуються за допомогою симплекс-методу і процес повторюється. Під час використання методу гілок та

границь не цілі розв’язки служать верхніми межами, а цілі розв’язки - нижніми межами.

Для рішення 1-PDP методом гілок і порізів необхідно звести задачу до задачі цілочисельного програмування. Позначимо стартову позицію транспортного засобу як +0 та кінцеву позицію -0. Для будь якого замовлення на транспортування i позначимо $+i$ та $-i$ точки завантаження та розвантаження відповідно. Проблема описана графом $G_n = (V_n, E_n)$ де

$$V_n = \{+0, -0\} \cup \{+i, -i \mid i \in N\}$$
$$E_n = \{(+0, -0)\} \cup \{(+0, +i) \mid i \in N\} \cup \{(-0, -i) \mid i \in N\} \cup E(K_{2n})$$

Маршрут який складається з ребер $R \subset E$ повинен задовольняти наступні умови:

$(+0, -0) \in R$, (2.14)

$|R \cap \delta(v)| = 2$ для кожного $v \in V$, (2.15)

G - зв’язний граф, (2.16)

$+i$знаходиться на маршруті з $+0$до $-i$, (2.17)

де $\delta(v) = \{e \in E\}$ - ребра зв’язані з точкою v.

Кожен маршрут, який задовольняє всі обмеження, має характеристичну функцію $x_R: E \to R$визначену як:

$$x_R(e) = 1, \text{ якщо } e \in E$$
$$x_R(e) = 0, \text{ в іншому випадку}$$

Обмеження для маршруту переводяться в алгебраїчні обмеження. Умови (2.14) та (2.15) еквівалентні умовам

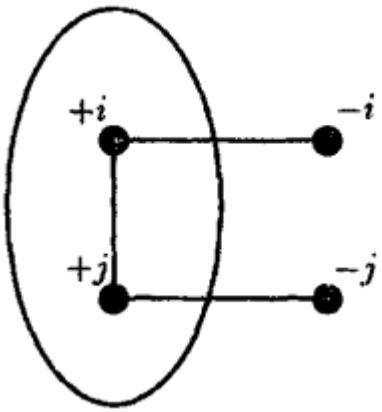
$$x(+0, -0) = 1$$
$$x(\delta(v)) = 2 \qquad \forall v \in V$$

Для $U \subset V$, $[U, U']$це множина ребер які з'єднують U та U'.

$$[U, U'] = \{(u, u' \ ) \in E, u \in U \ ma \ u' \in U'\}$$
$$\mu = \{U \subset V | + 0 \in V\}$$

Для кожного $U \in \mu \quad x([U: U']) \geq 2$, при чому, якщо $U = \{+0\}, \{+0, -0\}$ *або* $\{v\}$ для будь якого $v \in V$, то $x([U: U']) = 2$ і обмеження є лінійною комбінацією (2.14) та (2.15).

Обмеження порядку точок може бути заданим як $x(+i, -i) + x(+i, +j) + x(+j, -j) \leq 2$ показано на Рисунку 2.1 для двох замовлень. Це обмеження може бути узагальненим шляхом заміни ребра $(+i, -j)$.



Рисунку 2.1 - Найпростіше обмеження порядку. Незалежно від напрямку руху обмеження порушено.

Для будь-якої пари $i, j \in N$ і множини зупинок клієнтів $\{+i, -j\} \subset W \subset V$

$$x(W) + x(+i, -i) + x(+j, -j) \leq |W|$$

2.1.3 Використання адитивних функцій границь в методі гілок та меж

Метод гілок і меж (Branch-and-Bound) — один з широко відомих методів дискретної оптимізації. Цей метод працює на дереві рішень та визначає

принципи роботи конкретних алгоритмів пошуку розв'язків, тобто, є мета-алгоритмом. Для різних задач комбінаторної оптимізації створюють спеціалізовані алгоритми гілок та меж [23].

Результатом роботи алгоритму є знаходження максимуму або мінімуму функції на допустимій множині. При чому множина може бути як дискретною, так і раціональною. В ході роботи алгоритму виконується дві операції: розбиття вихідної множини на підмножини (гілки), та знаходження оцінок (меж). Існує оцінка множини згори та оцінка знизу. Оцінка згори — точка що гарантовано не менша за максимум на заданій підмножині. Оцінка знизу — точка що гарантовано не більша за мінімум на заданій підмножині. Множина що має найбільшу оцінку зверху зветься рекордною. На початку вся множина вважається рекордною. В загальному вигляді алгоритм має наступні кроки:

1. Початкова множина розбивається на підмножини;
2. Знайти оцінки згори та знизу для нових підмножин;
3. Визначити максимальну оцінку знизу серед усіх підмножин;
4. Видалити ті множини у яких оцінка зверху менша за максимальну оцінку знизу;
5. Знайти максимальну оцінку згори серед усіх підмножин та вважати її рекордною;
6. Якщо не досягнуто дискретності, або необхідної точності перейти по пункту 1;

Результатом роботи є значення між оцінкою згори та знизу для рекордної множини. Точністю є різниця між верхньою та нижньою оцінками, тобто для дискретних множин алгоритм завершений тоді, коли ці оцінки збігаються.

Фішетті та Тот розробили функцію адитивного обмеження яка може бути використана в методі гілок та меж для задачі Комівояжера з додатковими обмеженнями, наприклад Dial-a-ride задача для одного транспортного засобу. Ідея адитивного обмеження полягає в тому що ми використовуємо декілька функцій межування для проблеми адитивно замість окремого використання [24].

Дано граф $G = (V, A)$, де $V = \{1,2,\ldots,n\}$ - набір вершин та $A = \{(i,j), i \in V, j \in V\}$ - множина дуг. З кожною дугою $(i,j) \in A$ асоціюється вага $c_{i,j} \geq 0$. Для усіх $i \in V$ $c_{i,i} = \infty$. Також задано ряд відносин, визначених через набори вершин $P_2, P_3, \ldots, P_n$. Проблема полягає в тому, щоб знайти гамільтонівський ланцюг (тур), починаючи з вершини 1 і відвідуючи всі вершини P_h перед вершиною $h = (2,3,\ldots,n)$, таким чином, що сума ваг його дуг є мінімальною. Ця проблема, як відомо, є NP-складною і виникає в декількох задачах маршрутизації транспортного засобу. Математичним формулюванням проблеми $TSPPC$ виглядає наступним чином.

$$\vartheta(TSPPC) = min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j} \quad (2.18)$$

за умов

$$\sum_{j=1}^{n} x_{i,j} = 1, i = 1,\ldots,n; \quad (2.19)$$

$$\sum_{i=1}^{n} x_{i,j} = 1, j = 1,\ldots,n; \quad (2.20)$$

$$\sum_{i \in V/S} \sum_{j \in S} x_{i,j} \geq 1, S \subset V: 1 \notin S, S \notin \emptyset; \quad (2.21)$$

шлях з вершини 1 до h ($x_{i,j}$) повинен проходити через усі вершини в P_h;

(2.22)

$$x_{i,j} \geq 0, i = 1,\ldots,n,\ j = 1,\ldots,n; \quad (2.23)$$

$x_{i,j}$- ціле, $i = 1,\ldots,n,\ j = 1,\ldots,n;$ (2.24)

де $x_{i,j} = 1$тоді і тільки тоді, коли дуга (i,j) є оптимальним шляхом.

Різні підструктури можуть буди використані для пошуки нижньої межі у $TSPPC$. А саме будь яка функція межування для асиметричної задачі маршрутизації (ATSP) визначена у (2.18) - (2.21) та (2.23) - (2.24) може бути застосована до $TSPPC$.

Дві базові релаксації для ATSP це проблема призначення (AP) визначена у (2.18) – (2.20) і (2.23) та 1-SSAP визначена у (2.18), (2.20), (2.21) і (2.23). Обидві релаксації є проблемами лінійного програмування, тому значення меж та

залишкової вартості можуть бути знайденими шляхом рішення відповідних двоїстих задач.

Двоїста проблема AP:

$$\vartheta(DAP) = max \sum_{i=1}^{n} (u_1 + v_1)$$

$$c_{i,j} - u_i - v_j \geq 0, i = 1, \dots, n; j = 1, \dots, n$$

Двоїста проблема 1-SSAP:

$$\vartheta(D1SSAP) = max \sum_{\substack{S \subset V \\ S \neq \emptyset, 1 \notin S}} w(S) + \gamma$$

$$\mathrm{c}_{i,j} - \sum_{\substack{S \subset V: 1 \notin S \\ i \in V/S, j \in S}} w(S) \geq 0, \;\; i = 1, \dots, n; \;\; j = 2, \dots, n$$

$$c(i,j) - \gamma \geq 0, \;\; i = 1, \dots, n$$

$$w(S) \geq 0, \;\; S \subset V: 1 \notin S, S \notin \emptyset;$$

Оскільки обмеження (2.20) з $j \neq 1$ не враховується в проблемі 1-SSAP коли розглядається невід'ємна вартість, відповідні двоїсті параметри нехтуються в D-1-SSAP. Значення та зменшена вартість двоїстих задач може бути ефективно знайдена за допомогою Угорського алгоритму ($O(n^3)$) для проблеми AP та алгоритму Эдмонда – Карпа ($O(n^2)$) для проблеми 1-SSAP.

Додатково до описаної процедури межування, також можуть бути застосовані декомпозиція параметрів та диз'юнкції [21]. При включенні усіх межувальних процедур складність алгоритму становить ($O(n^5)$).

2.1.4 Евристичні алгоритми

Враховуючи обмеження точних методів у вирішенні великих проблем комбінаторної оптимізації, у багатьох практичних ситуаціях часто надають

перевагу методам апроксимації. Алгоритми апроксимації, як евристика та мета-евристика, є методами, які вирішують "важкі" проблеми комбінаторної оптимізації швидше за точні алгоритми. Однак немає жодних гарантій того, що отримане рішення є оптимальним, або що одне й те саме рішення буде отримано кожного разу, коли запускається алгоритм.

Евристичні алгоритми, як правило, базуються на досвіді, без застосування конкретних заздалегідь визначених правил. Важливий підклас евристичних алгоритмів - мета-евристичні алгоритми - можуть бути застосовані до широкого кола проблем комбінаторної оптимізації, лише з незначними змінами до базового визначення алгоритму. Багато мета-евристичних методів намагаються імітувати біологічні, фізичні чи природні явища, почерпнуті з реального світу.

При вирішенні проблем комбінаторної оптимізації з використанням евристичних або мета-евристичних алгоритмів пошук рішення проблеми близького до оптимального, як правило, поділяється на дві фази: побудова рішення та вдосконалення рішення. Побудова рішення стосується процесу створення одного або декількох початкових можливих рішень, які є відправною точкою, з якої починається пошук. На цій фазі, як правило, починається побудова можливого рішення проблеми. З іншого боку, фаза вдосконалення рішення відповідає за поступову модифікацію початкового рішення на основі певної заданої метрики, поки не буде отримана певна якість рішення або не пройде заданий проміжок часу [1].

У контексті пошуку якісного рішення ми зазвичай використовуємо термін пошуковий простір для позначення простору станів усіх можливих рішень / станів, до яких можна дійти з поточного рішення / стану. Щоб знайти якісне рішення, евристичні та мета-евристичні алгоритм переходять від одного стану до іншого, тобто від одного рішення-кандидата до іншого, за допомогою процесу, який часто називають локальним пошуком (ЛП). Під час ЛП нове рішення зазвичай генерується в околиці попереднього рішення. Околиця $N(x)$ розв’язку x - це підмножина пошукового простору, що містить один або кілька

локальних оптимумів, найкращих рішень у цій околиці. Процес переходу від одного рішення-кандидата до іншого що лежить в околиці цього першого рішення вимагає переміщення околиці (її меж), яке змінює деякі атрибути поточного рішення, що перетворити його на нове рішення x′. Функція витрат $f(x)$ використовується для оцінки кожного рішення-кандидата x та визначення його вартості порівняно з іншими рішеннями в його околиці. Найкраще рішення в загальному просторі пошуку називається глобально-оптимальним рішенням або просто оптимальним рішенням. На рис. 2.2 показано простір пошуку для функції мінімізації. Ця функція має три точки-рішення: A, B і C. Глобальним оптимумом серед трьох точок є точка C.

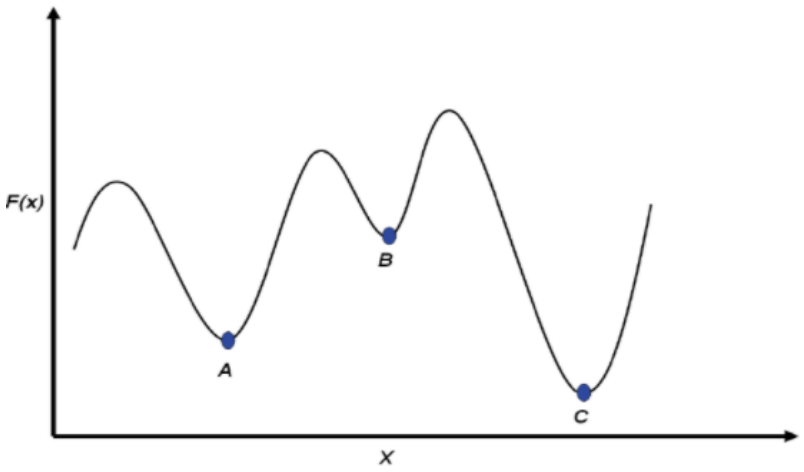


Рисунок 2.2 - Пошуковий простір: локальні та глобальний мінімуми

Далі наведено стислий опис широко відомих евристичних алгоритмів.

- Алгоритм сходження на вершину являє собою техніку математичної оптимізації, що належить сімейству алгоритмів локального пошуку. Алгоритм є ітераційним методом, який починається з довільного рішення задачі, а потім намагається знайти краще рішення шляхом покрокової зміни одного з елементів рішення. Якщо нове рішення є кращим за попереднє, робиться приріст для отримання наступного рішення. Це повторюється доки не знайдено рішення яке не може бути покращеним.

- Алгоритм імітації відпалу - алгоритм схожий на алгоритм сходження на вершину з доданням можливості вийти з локального мінімуму. Із-за цього він набагато повільніший. Алгоритм є стохастичним, а соме перехід до гіршого стану здійснюється з деякою ймовірність яка зменшується з кожною ітерацією алгоритму. Сама це дозволяю уникати локального мінімуму.

- Генетичний алгоритм - еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію. Суть алгоритму полягає в тому, по-перше генерується початкова популяція – набір можливих рішень задач. Далі усі рішення оцінюються, якщо рішення яке нас задовольняє не знайдено, то з існуючої популяції вибираються найкращі рішення. Далі вони схрещуються та мутують для отримання нової популяцій. Процес повторюється поки не знайдено рішення що б задовільнило нас.

- Мурашиний алгоритм – це мета-евристичний алгоритм, який натхненний поведінкою реальних мурах. Є один з ефективних поліноміальних алгоритмів для знаходження наближених рішень задачі комівояжера, а також аналогічних завдань пошуку маршрутів на графах. Справжні мурахи співпрацюють для пошуку їжі. Вони виділяють „феромон” на шляху від гнізда до джерела їжі. Робота алгоритму починається з розміщення мурашок у вершинах графа (містах), потім починається рух мурашок — напрям визначається імовірнісним методом, чим більше феромонів на шляху, тим більша ймовірність того, що цей шлях буде обраним. З часом коротші шляхи до їжі міститимуть більшу кількість феромону. Результат роботи алгоритму не є точним і навіть може бути одним з гірших, проте, в силу своєї стохастичності, повторення алгоритму може видавати досить точний результат.

- Табу-пошук – алгоритм покращує ефективність локального пошуку, пом’якшуючи його основне правило. По-перше, на кожному кроці можуть бути прийняті рішення гірші за попередні, якщо немає жодного кращого ходу

(наприклад, коли пошук застряг у локальному мінімумі). Окрім цього, вводяться заборони-табу, що запобігають повернення пошуку до вже існуючих рішень. Табу-пошук може бути використаний для пошуку задовільного рішення для задачі маршрутизації. По-перше, пошук із заборонами починається з початкового рішення, яке може бути згенероване випадково, або за допомогою якогось алгоритму найближчого сусіда. Для переходу до нового рішення два міста з поточного рішення змінюються місцями. Загальна відстань маршруту використовується для оцінки та порівняння рішень один з одним. Для запобігання циклів і, щоб не застрягти в локальних оптимумах, рішення буде додано до табу списку, якщо воно буде задовольняти рішення в околиці.

- Пошук змінних околиць - це мета-евристичний метод вирішення набору задач комбінаторної оптимізації та глобальної оптимізації. Метод досліджує віддалені околиці поточного діючого рішення та переходить звідти до нового, лише тоді, коли було зроблено поліпшення. Метод локального пошуку неодноразово застосовується для переходу від рішень в околицях до локальних оптимумів. Пошук змінних околиць був розроблений для апроксимації рішень дискретних і безперервних задач оптимізації, він призначений для вирішення задач лінійного програмування, задач цілочисельного програмування, змішаних задач цілочисельного програмування, задач нелінійного програмування тощо.

2.2 Розробка методу розв’язання поставленої задачі

2.2.1 Зведення задачі до задачі PDP

Згідно постановки задачі ми маємо простий граф G(V, E), вагову функцію $f: E \to R$та N запитів $q_i = (a_i, b_i)$, $i = 1, \dots, n$ де a_i- вершина в якій треба отримати вантаж, та b_i- вершина, куди необхідно доставити вантаж.

Нам необхідно отримати повний граф $G'(V', E')$ де

$$V' = \{a_i, i = 1..n\} \cup \{b_i, i = 1..n\}$$
$$E' = \{(a_i, b_j), i = 1...n, j = 1..n\}$$

V'- множина яка складається з усіх точок розвантаження та завантаження. E'- множина ребер між усіма точками V'.

Для прикладу, розглянемо граф на якому відмічено замовлення двох клієнтів (Рисунок 2.3). Для простоти нехай усі ребра одиничні. Необхідно отримати повний граф який складається з точок a_1,a_2,b_1 та b_2 (Рисунок 2.4).

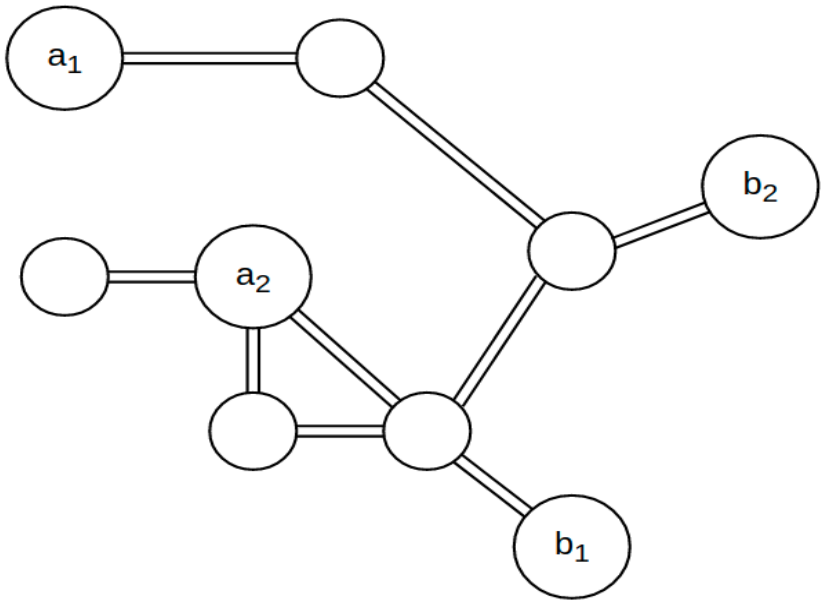


Рисунок 2.3 - Граф на якому відмічено два замовлення

Для побудови повного графа який складається тільки з вершин a_iта b_i необхідно знайти відстань між усіма цими точками. Розглянемо декілька алгоритмів які дозволяють це зробити.

- Алгоритм Флойда-Уоршалла
- Швидке множення матриць
- Алгоритм Дейкстри

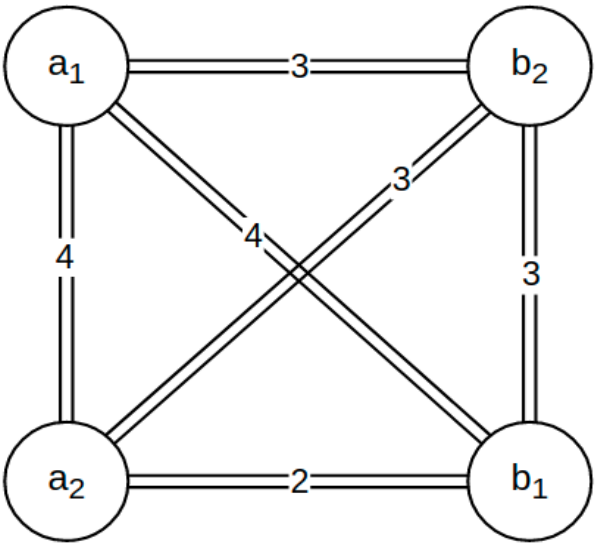


Рисунок 2.4 - Трансформований повний граф

Алгоритм Флойда-Уоршелла - це алгоритм пошуку найкоротших шляхів у зваженому графі з позитивною і негативною вагою ребер (але без негативних циклів) [25]. За одне виконання алгоритму будуть знайдені довжини (сумарні ваги) найкоротших шляхів між усіма парами вершин. Хоча він не повертає деталі самих шляхів, можна реконструювати шляхи за допомогою простих модифікацій алгоритму. Алгоритм Флойда-Уоршелла є прикладом динамічного програмування і був опублікований в своїй нині визнаної формі Робертом Флойдом в 1962 році. Однак він по суті такий же, як алгоритми, раніше опубліковані Бернардом Роєм в 1959 році, а також Стівеном Уоршеллом в 1962 році для пошуку транзитивного замикання графа, і тісно пов'язаний з алгоритмом Кліні (опубліковано в 1956 р) для перетворення детермінованого кінцевого автомата в регулярний вираз. Сучасна формулювання алгоритму у вигляді трьох вкладених циклів for була вперше описана Пітером Інгерманом також в 1962 році. Алгоритм Флойда-Уоршелла є ефективним для розрахунку всіх найкоротших шляхів в щільних графах, коли має місце велика кількість пар дуг між парами вершин. Складність алгоритму $\Theta(n^3)$.

Є алгоритми які прискорюють обчислення в щільних графах, але вони зазвичай мають додаткові обмеження (наприклад, вага дуг повинна бути у вигляді малих цілих чисел). Разом з тим, через великий константний час виконання перевага при обчисленнях над алгоритмом Флойда-Уоршелла проявляється тільки на великих графах.

Алгоритм Дейкстри - алгоритм на графах, винайдений нідерландським вченим Едсгер Дейкстрой в 1959 році [25]. Знаходить найкоротші шляхи від однієї з 20 вершин графа до всіх інших. Алгоритм працює тільки для графів без дуг негативного ваги. Алгоритм широко застосовується в програмуванні і технологіях. В середньому складність цього алгоритму Θ(n , де n - кількість вершин 2) графу. Алгоритм має перевагу при роботі на розріджених графах. На таких графах алгоритм має наступну складність - Θ($n\ log(n)$ + $m\ log(n)$). Враховуючи те шо нам необхідно застосувати алгоритм Дейкстри для всіх вершин то маємо наступну складність для розріджених графів - $\Theta(n^2 log(n) + mn\ log(n))$.

Виходячи з характеристик графу для розв'язку задачі найбільше підходить алгоритм Флойда-Уоршелла [26]. Також плюсом є те, що цей алгоритм може бути модифікований таким чином, що окрім відстаней між усіма точками, ми також можемо знайти найкоротший шлях між ними.

Модифікований алгоритм Флойда-Уоршелла має наступний вигляд:

```
dist = |V| × |V| масив мінімальних відстаней, ініціалізований∞
next = |V| × |V| масив індексів вершин, ініціалізований null
procedure FloydWarshallWithPathReconstruction() is
    for each edge (u, v) do
        dist[u][v] ← w(u, v)
        next[u][v] ← v
    for each vertex v do
        dist[v][v] ← 0
        next[v][v] ← v
```

```
    for k from 1 to |V| do
        for i from 1 to |V|
            for j from 1 to |V|
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                    next[i][j] ← next[i][k]
```

Процедура для відтворення найкоротшого шляху між двома точками має наступний вигляд:

```
procedure Path(u, v)
    if K[u][v] = null then
       return []
    path = [u]
    while u ≠ v
        u ← K[u][v]
        path.append(u)
    return path
```

Результатом застосування алгоритму до графу G є матриця відстаней між вершинами W та матриця K, за допомогою якої можна отримати мінімальний шлях між двома вершинами на графі.

$w_{i,j}$- відстань між вершинами i та j.

Усі вершини шляху з вершини i в вершину j можуть бути знайдені ітеративно $v_1 = i$, $v_2 = k_{i,j}$, $v_3 = k_{v_2,j}$, …, $v_n = k_{v_{n-1},j} = j$

2.2.2 Побудова оптимального шляху

На даному етапі можна почати будувати оптимальний шлях ітеративно. Розглянемо два можливі варіанти.

2.2.2.1 Використання жадібного алгоритму

На першій ітерації додамо перший запит на транспортування до шляху P. Таким чином на першій ітерації ми маємо $P = (a_1, b_1)$.

На i- тій ітерації необхідно додати вершини a_i та b_i до шляху P. Для цього використаємо жадібний алгоритм [27] та знайдемо таку позицію k, що

$$W_{ai,Pk} + \omega W_{ai,Pk-1} - \omega W_{Pk-1,Pk} \rightarrow min \qquad (2.25)$$

$$k \in [1, |P| + 1]$$

$$\omega = 1 \text{для } k \in [2, |P|], \text{ в іншому випадку } 0 \qquad (2.26)$$

Умова (2.25) дозволяє знайти найбільш вигідну позицію [28]. Наприклад, ми маємо шлях $a_1 \rightarrow a_2 \rightarrow b_1 \rightarrow b_2$ (Рисунок 2.5). Далі ми додаємо a_3 у різні позиції в шляху та знаходимо таку, яка дає мінімальний приріст загальної довжини шляху (Рисунок 2.6).

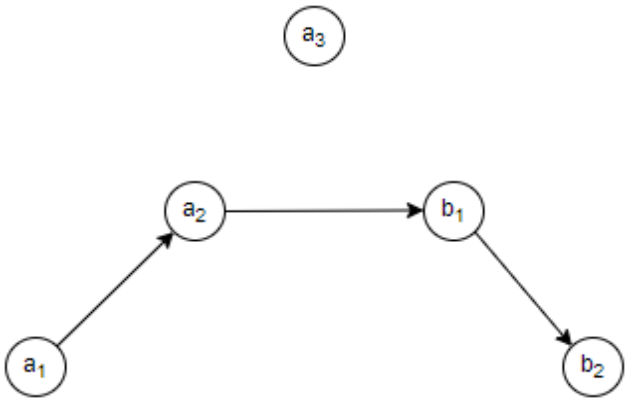


Рисунок 2.5 - Додавання нової точки до шляху

Для цього немає необхідності перераховувати довжину усього шляху. Достатньо розглянути відстань між додаваємою точкою та її сусідами.

При додаванні a_3 у початок шляху загальна довжина збільшиться на W_{a_3,a_1}- відстань між a_3 та a_1. При додаванні між позиціями a_1 та a_1- $W_{a_1,a_3} + W_{a_3,a_2} - W_{a_1,a_2}$.

Умова (2.26) необхідна для покриття випадків коли ми додаємо вершину у початок або у кінець шляху P.

Після того як позиція k найдена, додаємо вершину a_i до шляху P у позицію k. Усі вершини P_i, i = k..|P| зміщуються на одну позицію вправо. Тепер знайдемо позицію k_2 для вершини b_i при $k_2 \in [k+1, |P|+1]$ за допомогою формули (2.25) та додамо вершину b_i до шляху P у позицію k_2.

Повертаючись до прикладу наведеного на Рисунку 2.5, на цьому етапі необхідно знайти позицію де ми можемо додати адрес доставки замовлення b_3. Позиція b_3 може знаходитись тільки після позиції a_3.

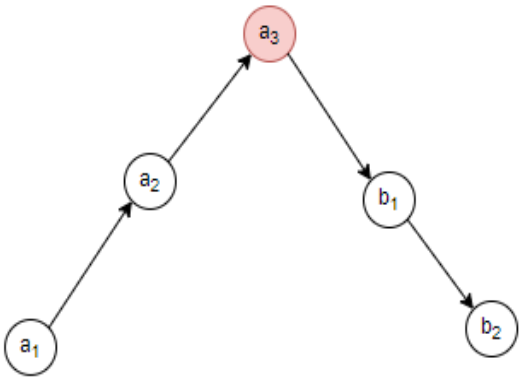


Рисунок 2.6 - Додавання нової точки до шляху

Оцінемо складність даного алгоритму. Для включення в існуючий шлях замовлення N необхідно:

1. Знайти позицію a_n- 2n операцій
2. Знайти позицію b_n- [1..2n] операцій

Маємо складність $\Theta(n)$ для включення замовлення в маршрут який складається з n замовлення.

Знаючи складність додавання замовлення до маршруту можемо знайти складність побудови маршруту для n замовлення:

$$1 + 2 + 3 + \ldots + n \Rightarrow \frac{n(n+1)}{2} \Rightarrow n^2$$

Таким чином ми маємо складність алгоритму $\theta(n^2)$.

Недоліком такого жадібного підходу є те, що можливі ситуації, коли результативний маршрут не є самим коротшим. Це можливо тому що точки завантаження та розвантаження додаються до існуючого шляху по черзі та додавання першої з них накладає обмеження на можливі позиції другої.

Розглянемо приклад такої ситуації зображений на рисунку 2.7. Ми маємо складений маршрут для трьох клієнтів. Його протяжність складає 51 умовна одиниця. Необхідно перерахувати маршрут для включення в нього нового замовлення.

Згідно з алгоритмом, по перше знайдемо таку позицію a_4, в якій загальна протяжність маршрути збільшується на мінімальну величину. Така позиція знаходиться між b_2 та b_1 (Рисунок 2.8 а).

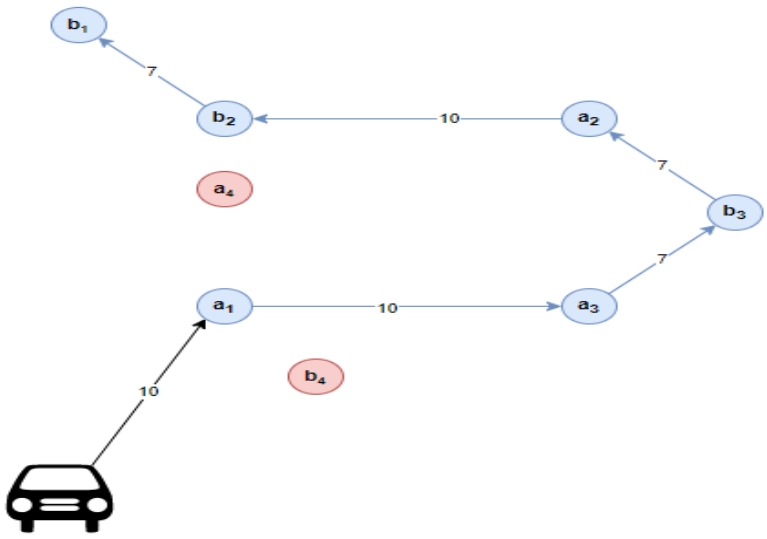
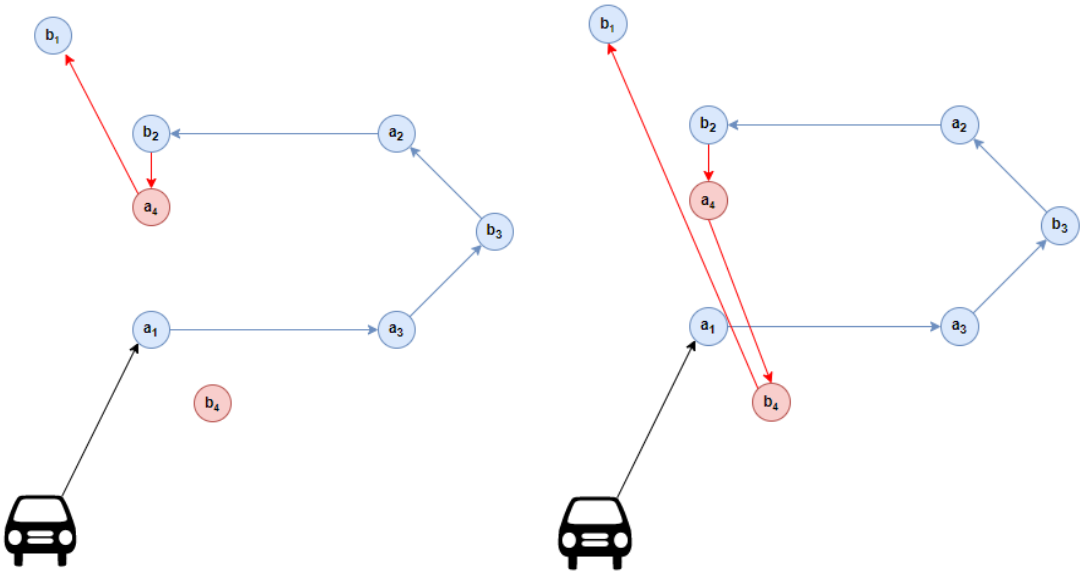


Рисунок 2.7 - Додавання замовлення до маршруту



a) b)

Рисунок 2.8 a - додавання точки завантаження; b - додавання точки розвантаження

Після цього до шляху додається точка розвантаження b_4 (Рисунок 2.8 b). В результаті ми маємо новий шлях який включає нове замовлення з новою протяжністю у ≈ *67.3* умовні одиниці. Та це не є оптимальним шляхом. Маршрут на рисунку 2.9 має протяжність у ≈ *53.3* умовні одиниці.

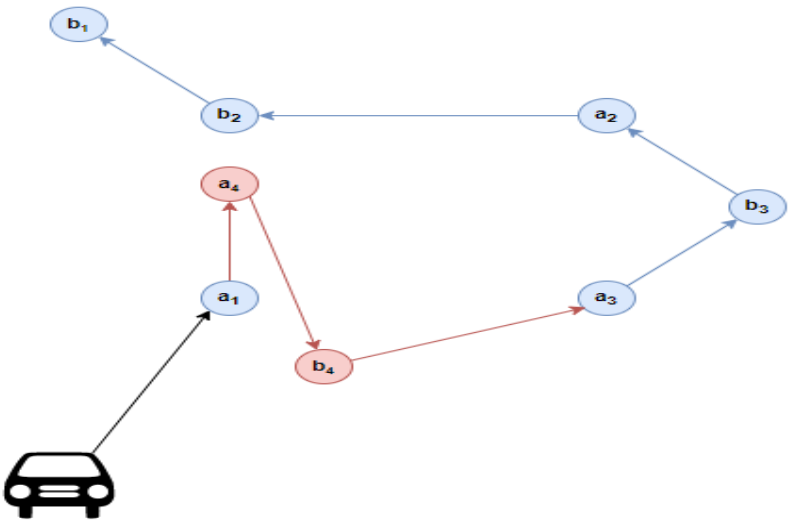


Рисунок 2.9 - Оптимальний маршрут

2.2.2.2 Перебір усіх варіантів при доданні нового замовлення

Ознайомившись з недоліком першого підходу можна зробити висновок - позиції додавання точок розвантаження та завантаження нового замовлення не можуть розглядатися окремо.

Замість того, щоб застосовувати формулу (2.25) двічі для пошуку позицій k_1та k_2 змінемо її наступним чином:

$$\Delta C_k + \Delta C_{k_2} \to min \quad (2.27)$$

$$k < k_2 \quad (2.28)$$

ΔC_k - зміна у загальній протяжності маршруту після додавання до нього точки завантаження - a_i в позицію k та ΔC_{k_2} - зміна у протяжності після додавання точки розвантаження - b_i в позицію k_2.

Нехай

$l_1 = P_{k-1}$ - точка після якої ми додаємо a_i до шляху

$r_1 = P_k$ - точка перед якою ми додаємо a_i до шляху

$l_2 = (P_{k_2-1}$ - якщо $k \neq k_2$, a_i - в іншому випадку) - точка після якої ми додаємо b_i до шляху

$l_2 = P_{k_2}$ - точка після якої ми додаємо b_i до шляху

Тоді

$$\Delta C_k = W_{a_i,r_1} + \omega_1 W_{a_i,l_1} - \omega_1 W_{l_1,r_1}$$

$$\Delta C_{k_2} = W_{b_i,r_2} + \omega_2 W_{b_i,l_2} - \omega_2 W_{l_2,r_2}$$

$$k \in [1, |P| + 1]$$

$$k_2 \in [2, |P| + 1]$$

$\omega_1 = 1$ для $k \in [2, |P|]$, в іншому випадку 0

$\omega_2 = 1$ для $k_2 \in [2, |P|]$, в іншому випадку 0

У виразах $W_{a_i,r_1} + \omega_1 W_{a_i,l_1} - \omega_1 W_{l_1,r_1}$ та $W_{b_i,r_2} + \omega_2 W_{b_i,l_2} - \omega_2 W_{l_2,r_2}$ перший та другий доданки – вага нових дуг які ми додаємо до вже існуючого маршруту. Третій доданок – вага дуги яку ми видаляємо з цього маршруту.

Коефіцієнти ω_1 та ω_2 необхідні для покриття випадку, коли ми додаємо нову точку до існуючого маршруту в початок або кінець. У такому випадку ці коефіцієнти дорівнюють нулю.

Після того, яка позиції k та k_2 знайдені додаємо пункт a_i в позицію k існуючого маршруту та пункт b_i в позицію $k_2 + 1$ відповідно. Повторюємо ітерацію для усіх нових замовлень.

У результаті ми маємо шлях P який проходить через усі вершини графу G'. Цей шлях є оптимальним та задовольняє задані обмеження. Для того щоб отримати реальний шлях на графі G ми повинні трансформований P використовуючи матрицю K.

Для пошуку k та k_2 які задовольняють умови (2.27) та (2.28) наступний алгоритм може бути використаний:

```
k1 = 0
k2 = 0
minimum = ∞
for each i in [1..|P|] do
      for each j in [i..|P|] do
            if (ΔC_k(i) + ΔC_k2(j) < minimum )
                  minimum = ΔC_k(i) + ΔC_k2(j)
                  k1 = i; k2 = j
```

На Рис. 2.10 наведено схему вищенаведеного алгоритму. У цій схемі використовується функція – «delta». Вона рахує як зміниться довжина маршруту при додаванні точки для шляху «path». Схема алгоритму цієї функції наведено на рис. 2.11.

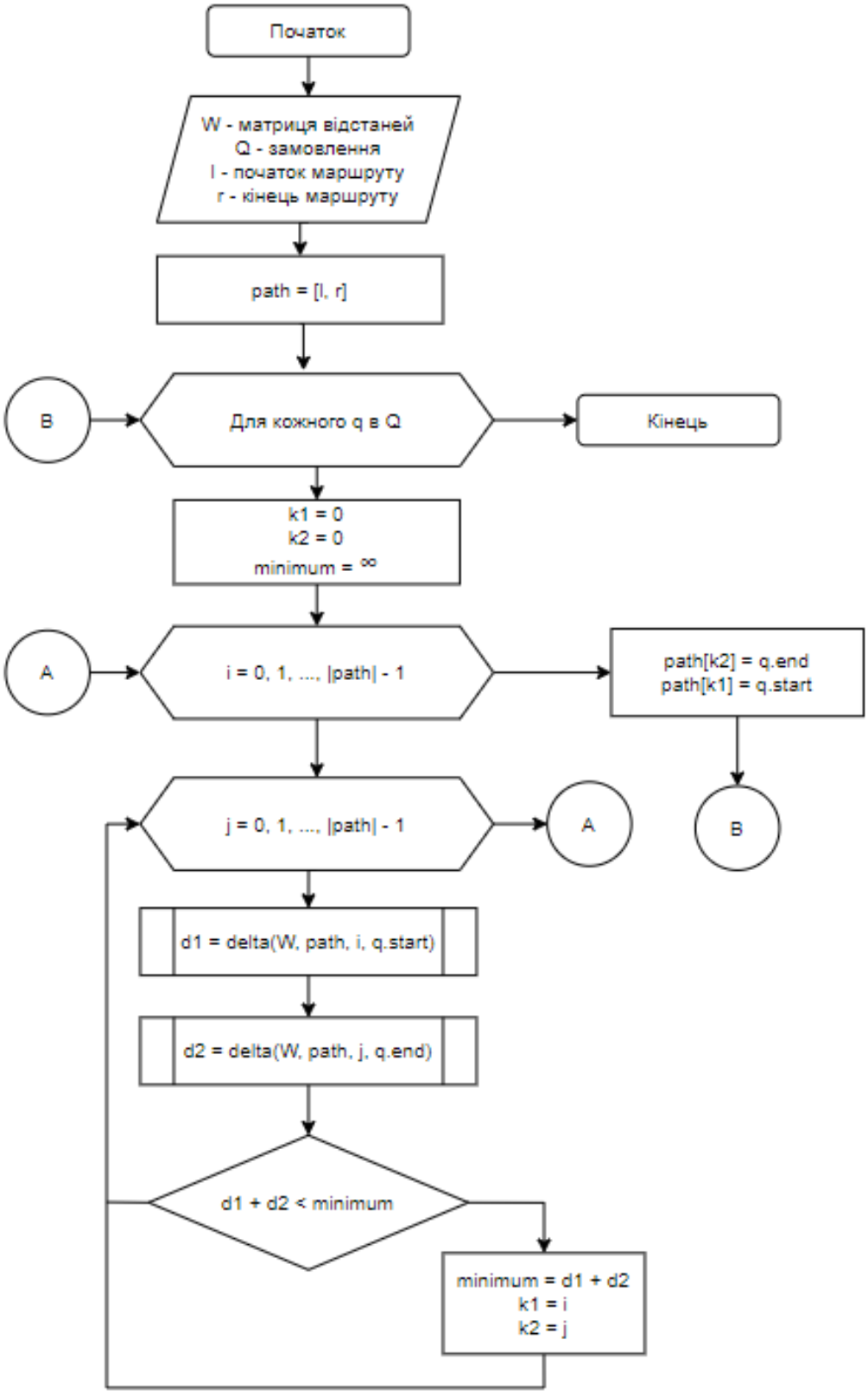


Рисунок 2.10 - Схема запропонованого алгоритму

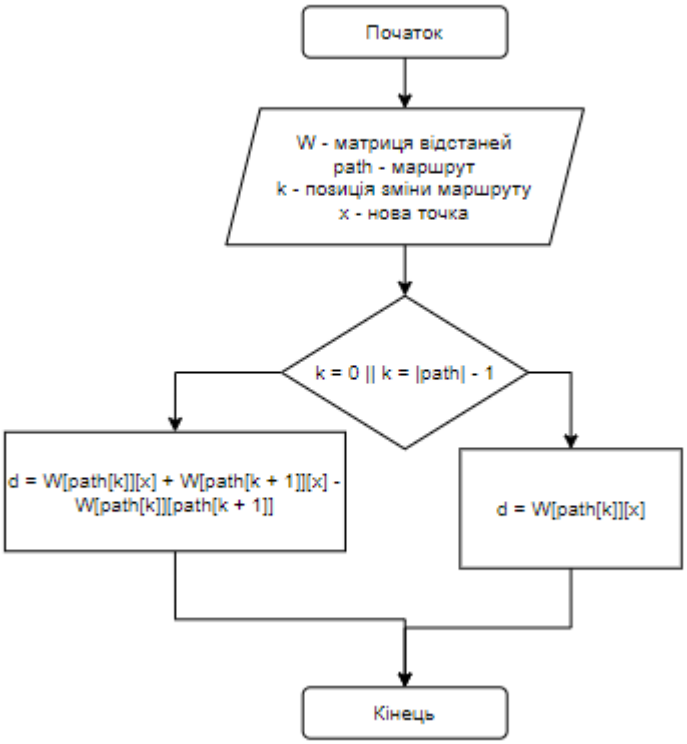


Рисунок 2.11 - Схема алгоритму оцінки модифікації шляху

Оцінемо складність цього алгоритму. Складність перерахування маршруту для n-го замовлення:

$$\Theta(1+2+3+\ldots+n) = \Theta(\frac{n(n+1)}{2}) = \Theta(n^2) \quad (2.29)$$

Виходячи з (2.29) складність розрахунку маршруту для n замовлень становить:

$$\Theta(1^2+2^2+3^2+\ldots+n^2) = \Theta(\frac{n(n+1)(2n+1)}{6}) = \Theta(n^3)$$

2.3 Аналіз отриманих результатів

Перший розглянутий метод розв’язку задачі PDP – метод що базується на динамічному програмуванні. Головним недоліком цього алгоритму є

експоненціальне зростання часу виконання та використаної пам’яті, а саме $(2n+1)^2 \cdot 3^n$ та $2n \cdot 3^{n-1}$ [21]. Таким чином ми можемо отримати розв’язок для задачі з десятьма замовленнями (20 точок на графі) то 100 замовлень є непід’ємною цифрою для цього алгоритму.

В методі гілок та меж з використанням декількох процедур межування використовується інший підхід. Множина рішень розбивається на менші множини. Для кожної множини рішень оцінюються найнижче та найвище значення цільової функції. Це дозволило знизити складність алгоритму до n^5 [24].

Метод гілок та відсічень базується на методі гілок та меж. У методі використовуються ріжучі площини (алгоритм Гоморрі) в процедурі межування. Алгоритм має таку саму алгоритмічну складність але з меншим коефіцієнтом тобто при малому значенні n це алгоритм буде швидшим [22]. З збільшенням n ця різниця зникає.

У роботі було запропоновано два схожі ітеративні алгоритми для рішення поставленої задач. Перший варіант – примітивний алгоритм коли ми кожну точку яку потрібно додати до існуючого маршрути додаємо в найвигіднішу позицію, таку яка дає мінімальний приріст до протяжності маршруту. Це дозволяє досягти складності алгоритму у n^2 но з недоліком, імовірно побудований маршрут не буде глобально оптимальним. Це трапляється тому що ми розглядаємо точки пікапу та доставки у замовленнях окремо. Додаванням точки пікапу у найвигіднішу позицію ми обмежуймо можливі позиції точки доставки у маршруті – вона може бути тільки після точки пікапу. Тож можливі ситуації коли додавання точки пікапу до маршруту у позицію яка дає мінімальний приріст у довжині не завжди оптимально. Така ситуація розглянута у розділі 2.2.2.1.

Друга варіація алгоритму націлена на вирішення цієї проблеми. В алгоритмі замість того, щоб шукати позиції для нових точок пікапу та доставки

окремо, використовується підхід, коли переглядаються усі можливі позиції двох точок. Кожна пара позицій оцінюється та обирається найкраща.

Цей підхід рішає проблему першого алгоритму ціною часу виконання, у цьому варіанті алгоритм має поліноміальний час виконання - n^3. До недоліків можна віднести те, що деякі замовлення умовно можуть бути ніколи не виконаними. Таке можливо, коли після планування маршруту будуть нескінченно з'являтися нові замовляння які вигідніше виконати першими. Для поставленої задачі це не є критичним критерієм, це лише потенціальна ситуація якої можна уникнути позачерговим обслуговування клієнтів, замовлення яких були перенесені на наступний день.

Також перевагою цього алгоритму є те, що він інтуїтивно зрозумілий та його легко реалізувати.

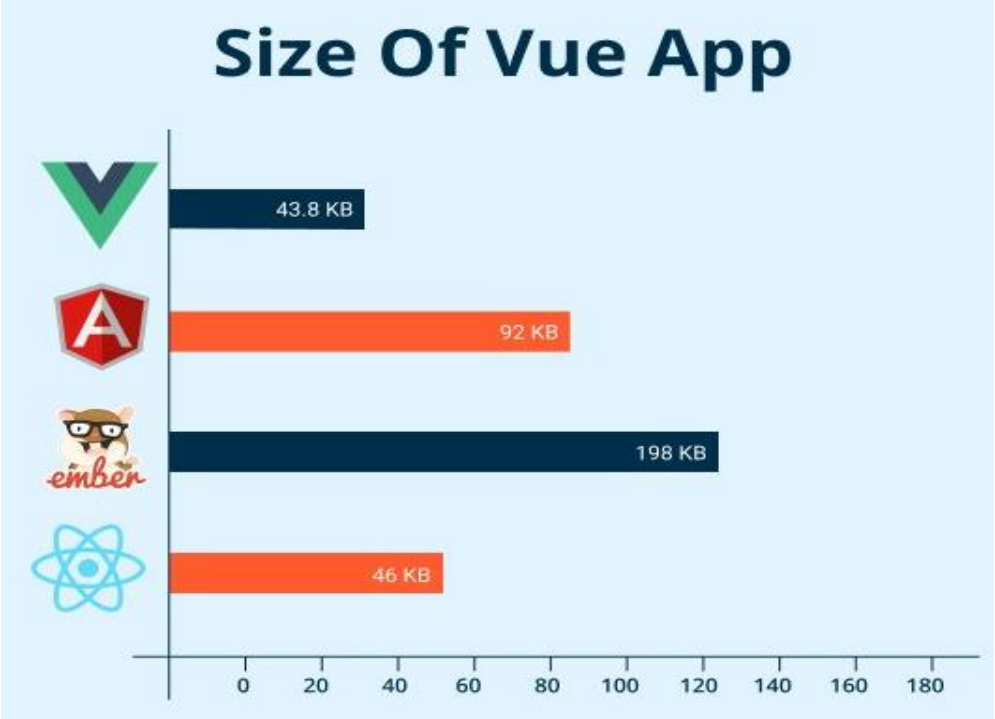
2.4 Розробка програмного комплексу

Для реалізації програмного комплексу було обрано клієнт-серверний веб-орієнтований напрям. У наші дні цей напрям набув широкого попиту. Можливості сучасних веб-браузерів не поступаються десктопним рішенням та порівняно з ними веб-додатки забезпечують цілу низку переваг. До цих програм можна отримати доступ з будь-якого комп'ютера через Інтернет, замість того, щоб встановлювати їх окремо на кожному комп'ютері, з якого ви хочете отримати доступ. Веб-додатком можна користуватися з будь-якого девайсу будь то персональний комп'ютер чи мобільний телефон з IOS або Android.

2.4.1 Фронтенд технології

Для реалізації фронтенду було обрано Vue.js фреймворк. Vue.js - це динамічна JavaScript система, яку розробники використовують для збірки інтерфейсів та створення потужних односторінкових додатків (SPA) [29].

Однією з переваг є відносно малий розмір фреймворку. На Рисунку 2.12 наведено порівняння з іншими поширеними фреймворками [30].



Рисунку 2.12 - Порівняння розміру поширених фреймворків

Також цей фреймворк має гарну і велику документацію. Вона полегшує створення власних додатків кожному, хто має трохи знань про JavaScript або HTML. Фреймворк поєднує у собі Angular та React.

Разом з Vue.js використовується «cli-plugin-pwa», що дозволить веб-сайту бути PWA (Progressive Wev Application) – сумісним. Це дозволить користувачам завантажити веб – сайт у вигляді програмного застосунку для IOS або Android. PWA застосунок майже не відрізняється від нативних застосунків.

Для полегшення розробки також було вирішено використовувати Bootstrap. Bootstrap - це бібліотека що дозволяє швидко розробляти та налаштовувати адаптивні сайти. Він включає змінні та міксини в Sass, адаптивну систему сіток велику кількість готових компонентів та плагіні [31].

2.4.2 Розробка фронтенду

Було розроблено веб-сторінку яка дозволяє відображати довільний граф та додавати замовлення на ньому (Рисунок 2.13). Головна сторінка складається з панелі для керування замовленнями та зображення графу з оптимальним маршрутом на ньому.

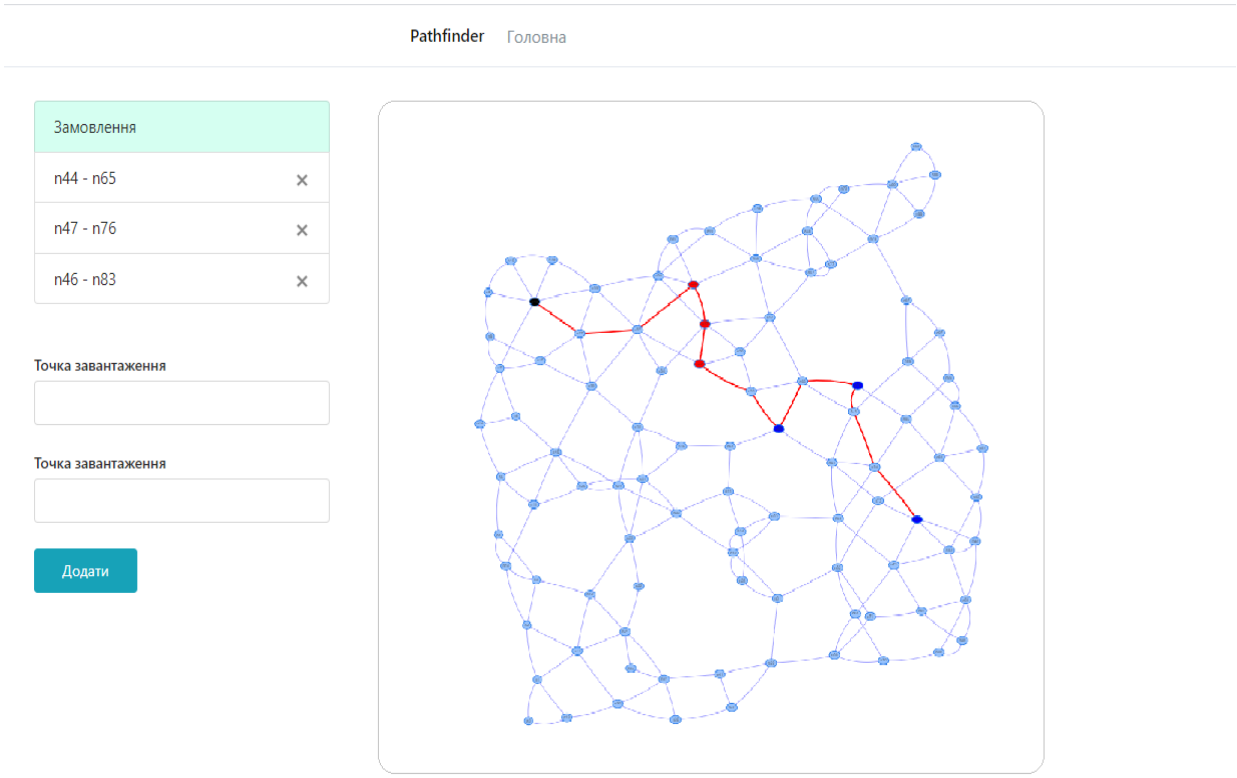


Рисунок 2.13 - Відображення задовільного графу

Граф можу бути завантажений на сервер через вкладку “Налаштування”. Для цього необхідно в полі “Граф” обрати csv файл з графом. У першій строчці графу задається список назв вершин. Далі знаходиться вагова матриця. Усі завантажені графи відображаються нижче. Між усіма завантаженими графами можна переключатися.

Користувач має можливість переключитися на побудову оптимального маршруту на гугл мапі (Рисунок 2.14). Для цього була проведена інтеграція з Google Maps API [32]. Користувач може відмітити дві точки на мапі - точку завантаження та точку розвантаження. Також адреси можуть бути введені в відведені для цього поля. Такі адреси будуть оброблені завдяки google maps api та відображені на мапі.

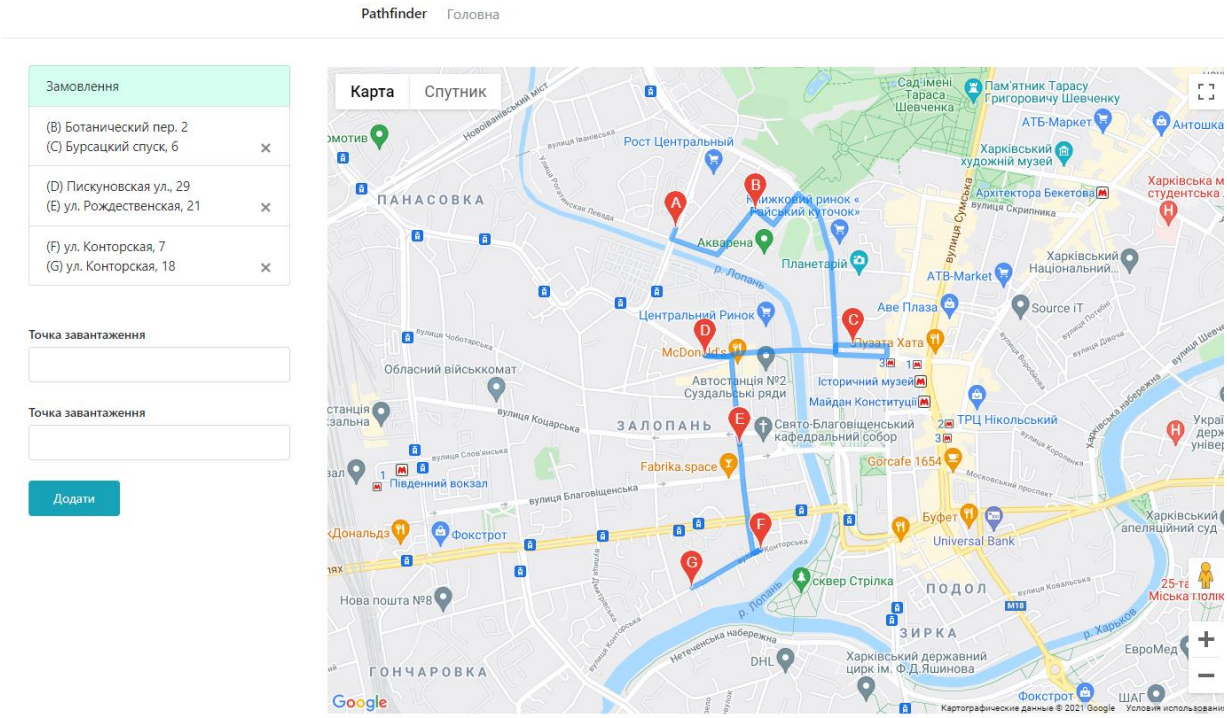


Рисунок 2.14 - Інтеграція з Google Maps

2.4.3 Розробка серверної частини

Серверна частина веб-сайту була реалізована на Node.js. Node.js - платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript. Засновником платформи є Раян Дал (Ryan Dahl) [33]. Якщо раніше Javascript застосовувався для обробки даних в

браузері користувача, то node.js надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього виконання.

На сервері реалізовані наступні функції:

- збереження та обробка графу в csv формат;
- зведення задачі на графі до задачі PDP;
- рішення задачі PDP;
- додання нових замовлень до існуючого рішення, тобто рішення задачі яке базується но попередньої рішенні.

Усі наведені функції можуть буди використаними через Rest API.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено повноцінний аналіз проблем класу пікапу та доставки (PDP). Наведено опис проблем які відносяться до PDP, їх класифікація, порівняння їх обмежень, цільових функцій, параметрів тощо. Розглянуто існуючі підходи які можуть допомогти в рішенні практичних задач PDP, такі як сервіси, оптимізаційні вирішувачі, консалтингові послуги.

Було розглянуто три точних методи рішення динамічної задачі PDP з одним транспортним засобом без обмежень на завантаження, а саме:

- динамічне програмування - метод запропонований Harilaos N. Psaraftis в якому оптимальний шлях знаходиться рекурсивно;
- гілок та меж - метод вдосконалений Алісой Ленд і Елісон Дойг для вирішення задачі динамічної задачі PDP. Розглянутий метод використовує низку функції в процесі пошуку меж;
- гілок та відсічень (branch and cut) – метод який базується на методі “гілок та меж” та алгоритму Горі. Для використання цього методу задача PDP зводиться до задачі лінійного цілочисельного програмування.

Також наведено стислу характеристику евристичних та мета-евристичних методів та алгоритмів для вирішення різних задач PDP.

У роботі було запропоновано два схожі ітеративні алгоритми для рішення поставленої задач. В обох варіантах оптимальний маршрут будується ітеративно. Спочатку шлях складається з початкової та кінцевої точок, якщо вони задані. В іншому випадку маршрут складається з точок пікапу та доставки першого замовлення. Далі для кожного замовлення по черзі ведеться пошук місць, у якому вигідно змінити маршрут. Різниця двох алгоритмів лише у підході для пошуку цих місць.

Перший варіант – примітивний алгоритм коли ми кожну точку яку потрібно додати до існуючого маршрути додаємо в найвигіднішу позицію, таку яка дає мінімальний приріст до протяжності маршруту. Основним недоліком алгоритму є те, що за деяких умов можливі локально оптимальні рішення.

Другий варіант алгоритму вирішує цю проблему. У ньому ведеться пошук одразу двох оптимальних позицій, перебираються всі можливі комбінації позицій точки пікапу та точки доставки в існуючому маршруті. Найкращі з отриманих позицій використовуються. Алгоритм має один недолік - виконання невигідного замовлення може постійно відкладатися. Таке можливо коли до системи постійно надходять нові, кращі замовлення. Приведений недолік не є критичним для поставленої задачі, також в роботі запропоновано рішення цієї проблеми при використанні розробленого алгоритму для вирішення прикладної задачі.

На базі цього алгоритму було розроблено сервіс та веб-інтерфейс до нього що дозволяє користувачеві вирішити поставлену задачу пікапу та доставки для одного транспортного засобу.

Сервіс реалізовано мовою javascript та виконується за допомогою Node.js. Основні функції сервісу:

- завантаження графів на сервер;
- збереження графів;
- рішення задачі на заданому графі з заданими замовленнями – парами вершин на цьому графі;
- збереження рішень;
- додання нових замовлень до існуючого рішення, тобто рішення задачі яке базується но попередньої рішенні.

Сервіс надає REST API. На базі цього API було розроблено веб-сайт з використанням Vue.js та bootstrap. Інтерфейс надає доступ до всіх функцій сервісу в повному обсязі. Більш того, у веб-сайт було інтегровано гугл мапи.

Завдяки цьому користувач може задавати умови задачі інтерактивно на мапі. Результат виконання алгоритму буде зображено на ній.

Виконано оцінку складності розробленого алгоритму та порівняльний аналіз з відомими розглянутими алгоритмами розв’язування динамічної задачі PDP з одним транспортним засобом без обмежень на завантаження.

Підсумовуючи, поставлена задача кваліфікаційної роботи виконана у повному обсязі. Отримані результати досліджень доповідалися на 25-му Міжнародному молодіжному форумі «Радіоелектроніка і молодь у XXI столітті», тези доповіді подані на Тринадцяту Міжнародну науково-практичну конференцію «Сучасні інформаційні та інноваційні технології на транспорті MINTT-2021», яка відбудеться 25 – 27 травня 2021 року у м. Херсоні, та на VI Міжнародну науково-технічну конференцію «Поліграфічні, мультимедійні та web-технології» у ХНУРЕ 14-17 травня 2021 р.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Manar I. Hosny Vehicle Routing with Pickup and Delivery: Heuristic and Meta-heuristic Solution Algorithms, LAP Lambert Academic Publishing, 2012
2. Марк Соль та Мартін Савелсберг. Загальна проблема PDP. Наука про транспорт, 29:17 - 29, 02 1995.
3. Епплгейт Д.Л., Біксбі Р.Є., Чватал В., Кук В. Дж. Проблема продавця. Прінстонська університетська преса, 2007. С. 44-52.
4. Левітін А. Алгоритми. Введення в розробку і аналіз. Вільямс, 2006. Р. 160.
5. Гері М., Джонсон Д. Обчислювальні машини і складнообчислювальні завдання. Світ, 1982.
6. Окано Х. Вивчення практичних рішень для задач комбінаторної оптимізації. Університет Тохоку, 2009. С. 14-17.
7. I.Grebennik. Solution Strategy for One-to-One Pickup and Delivery Problem Using the Cyclic Transfer Approach / I. Grebennik, O. Chorna, R. Dupas, I. Litvinchev, T. Romanova // EAI Endorsed Transactions on Energy Web, Special issue on Energy Conservation, Information Technologies and Large Scale Optimization, Issue 27, 2020, e5 Scopus https://eudl.eu/doi/10.4108/eai.13-7-2018.164110
8. Жан-Франсуа Кордо, Жільбер Емерік Ляпорт. Завдання Dial-a-Ride (DARP): варіанти, проблеми моделювання та алгоритми.
9. H Psaraftis, A dynamic programming solution to the single vehicle many to many immediate request dial-a-ride problem, Transportation Science
10. Oliveira, H.C.B.de; Vasconcelos, G.C. (2008). "A hybrid search method for the vehicle routing problem with time windows". Annals of Operations Research.

11. P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehiclerouting and scheduling problems. Operations Research, 41(5):935–946, 1993.
12. URL: https://www.speedyroute.com/
13. URL: https://www.gurobi.com/
14. URL: https://www.ibm.com/analytics/cplex-optimizer
15. Tobias Achterberg, SCIP: solving constraint integer programs, 2009, DOI 10.1007/s12532-008-0001-1
16. URL: https://www.gnu.org/software/glpk/#introduction
17. Route Optimization Consulting Services, URL: https://routeoptimizationconsultants.com.
18. Supply Chain, Warehouse & Logistics Consultants, URL: https://www.paultrudgian.co.uk/
19. Калита Н.І., Паречин В.П. Пошук оптимального шляху в задачі 1-PDP в реальному часі // Тринадцята Міжнародна науково-практична конференція «Сучасні інформаційні та інноваційні технології на транспорті MINTT-2021», 25 – 27 травня 2021 року, м. Херсон. – Херсон, 2021. –с.00-00.
20. Беллман Р. Динамическое программирование. — М.: Издательство иностранной литературы, 1960.
21. H. PSARAFTIS, "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem," Transportation Science 14, 130-1.34, (1980).
22. John E., Mitchell (2002). "Branch-and-Cut Algorithms for Combinatorial Optimization Problems" (PDF). Handbook of Applied Optimization: 65–77.
23. Дакін Р. Дж. (1965). Алгоритм пошуку дерева для змішаних цілочисельних задач програмування. У: Комп’ютерний журнал, том 8, С. 250—255
24. М. ФІШЕТТІ І П. ТОТ, "Додаткова обмежувальна процедура для комбінаторних задач оптимізації", Operations Research 37, 319-328, (1989).

25. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: Вильямс», 2006. — С. 1296.
26. Паречин В.П. Розв’язування задачі 1-PDP в online маршрутизації друкованих видань // VI Міжнародна науково-технічна конференція «Поліграфічні, мультимедійні та web-технології», Харків, ХНУРЕ, 14-17 травня 2021 р. – Харків, ХНУРЕ,2021. –с.00-00
27. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 16. Жадные алгоритмы // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005. — 1296 с.
28. Паречин В. П. Пошук оптимального шляху через декілька пунктів відправлення та пунктів споживання // 25-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6. – Харків: ХНУРЕ. 2021. –с. 326-327.
29. Фреймворк Vue.js, URL: https://vuejs.org/v2/guide/
30. Переваги Vue.js, URL: https://www.esparkinfo.com/advantages-of-vue-js.html
31. Бібліотека Bootstrap, URL: https://getbootstrap.com/
32. Google maps API overview, URL: https://developers.google.com/maps
33. Node.js, URL: https://nodejs.org/uk