Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Студента Молчанова Івана Володимировича

академічної групи 124-19-1

спеціальності 124 Системний аналіз

на тему: «Системний аналіз та оптимізація логістичних процесів в умовах підприємства роздрібної торгівлі господарчими товарами»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**Інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Желдак Т.А.*			
розділів:				
Інформаційно-аналітичний	*к.т.н., доц. Желдак Т.А.*			
Спеціальний розділ	*к.т.н., доц. Желдак Т.А.*			
Рецензент	*д.т.н.,проф. Алексєєв М.А.*			
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро
2023

ЗАТВЕРДЖЕНО:

завідувач кафедри

Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

«____»_____20___року

ЗАВДАННЯ
на кваліфікаційну
роботуступеня
бакалавра

студенту *Молчанову І.В* **академічної групи** *124- 19-1*
спеціальності: *124 Системний аналіз*
на тему «*Системний аналіз та оптимізація логістичних процесів в умовах підприємства роздрібної торгівлі господарчими товарами»*
затверджену наказом ректора НТУ «Дніпровська політехніка» №350-с від 16.05.2023 р.

Розділ	Зміст	Терміни виконання
1. Інформаційно-аналітичний розділ	*Проаналізувати структуру об'єкта дослідження. Визначити предметну область дослідження та проблему. Обґрунтувати методи виконання поставлених завдань*	10.05.2023
2.Спеціальний розділ	*Розв'язати поставлені задачі: розробити алгоритми та створити систему для оптимізації логістичних перевезень, враховуючи різні фактори, які необхідні для зручного оптимального переміщення товарів.*	10.06.2023

Завдання видано _____ *доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

Дата видачі: *06.09.2023 р.*

Дата подання до екзаменаційної комісії: 20.06.2023 р.

Прийнято до виконання _____ Молчанов І.В

РЕФЕРАТ

Пояснювальна записка: 76 с., 26 рис., 3 табл., 5 додатків, 16 джерел.

Об'єктом дослідження в роботі є логістичні процеси постачання товарів в мережі ТОВ «Товари-К».

Предметом дослідження є розробка та реалізація програмного додатка що будує оптимальний шлях враховуючи карту міста.

Метою даної кваліфікаційної роботи є покращення економічних показників роботи підприємства за рахунок мінімізації логістичних витрат

Методи дослідження: метод імітації відпалу та метод пошуку табу для вирішення задачі пошуку мінімального шляху, метод побудови інформаційних систем для застосування існуючого програмного забезпечення у роботу власної інформаційної системи.

В *інформаційно–аналітичному розділі* наведено аналіз об'єкту дослідження та ключових проблем на ньому. Поставлені задачі дослідження та обрано концепції їх розв'язання.

У *спеціальному розділі* сформовано бізнес-логіку та алгоритми програми для аналізу поставленої задач

Практична цінність отриманих результатів полягає в тому, що запропонована та розроблена система скорочує час на побудову оптимального маршруту та скорочує витрати компанії.

Ключові слова: CPG, логістика, задача комівояжера, оптимізація, VRP, Google OR-Tools.

ABSTRACT

Explanatory note: 76 pages, 26 figs., 3 tab., 5 appendices, 16 sources.

The object of research in the work is the logistics processes of supply of goods in the network of LLC "Tovars-K".

The subject of the study is the development and implementation of a software application that builds the best way, taking into account the city map.

The purpose of this qualification work is to improve the economic performance of the enterprise by minimizing logistics costs Research methods: annealing simulation method and taboo search method for solving the problem of finding a minimum path, method of building information systems for applying existing software to your own information system.

The information-analytical section provides an analysis of the object of research and key problems on it.

The tasks of the research are set and the concepts of their solution are chosen.

In a special section, the business logic and algorithms of the program for analyzing the tasks are formed

The practical value of the obtained results lies in the fact that the proposed and developed system reduces the time for building an optimal route and reduces the company's costs.

Tags: CPG, logistics, traveling salesman task, optimization, VRP, Google OR-Tools.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SPG - Consumer Packaged Goods , товари повсякденного попиту

ШІ – Штучний інтелект

КТГ - Керівник торгової групи

VRP - Vehicle Routing Problem (проблема маршрутизації транспортних засобів)

ОРТ - Оптимізація розкладу транспорту

ВСТУП

На сьогоднішній день український ринок наповнений різноманітними продуктами харчування, особливо так званою снековою продукцією. Кожен день пересічний українець вживає безліч різноманітної їжі, серед якої дуже часто трапляються чіпси, печиво, звичайно ж шоколад, та інші ласощі. Всі ці товари відносяться до товарів з швидкою оборотністю, бо вживаються кожен день. Хто же не купить жуйку по дорозі на роботу або не візьме шоколадний батончик за годину до обідньої перерви. Саме через часту вживаність таких продуктів їх продаж є прибутковим бізнесом. Дуже рідко трапляються ситуації, коли виробники самотужки реалізують власну продукцію. Як правило, кожна компанія - виробник має одного або декілька компаній - дистрибуторів, які беруть всю відповідальність за доставку товарів до кінцевого споживача на себе. У сучасній торгівлі дистрибуція є важливим поняттям, де розглядаються проблеми забезпечення оптимального руху товару каналом розподілу до кінцевого споживача, а у сучасному бізнесі дистрибуція – одна з найприбутковіших галузей.

Дистрибуція — це поняття у логістиці, іноді позначається, яке означає комплекс взаємопов'язаних функцій, які реалізуються в процесі розподілення матеріального потоку між різними, як правило, гуртовими покупцями.

Система дистрибуції - складна економічна система, яка об’єднує в своєму складі виробника готової продукції та різноманітних посередників, які на договірних засадах (на основі дистриб’юторського договору) спільно здійснюють маркетингову, комерційну, логістичну діяльність з переміщення продукції до кінцевого споживача і її продажу відповідно до стратегії суб’єкта господарювання – організатора такої системи з дотриманням встановлених ним умов продажу, цін продажу, стандартів обслуговування і під його контролем[1].

Система дистрибуції основана на поєднанні в процесах збуту готової продукції таких основних складових, як:

- стратегія поведінки підприємства (організатора системи дистрибуції) на ринку, зокрема – стратегія маркетингового розподілу, концепція організації системи дистрибуції тощо;

- партнерство з комерційними посередниками, які на договірній основі об'єднуються в канали розподілу;

- ціноутворення, яке має ґрунтуватися на єдиних для всіх учасників каналів розподілу продукції підходах і передбачати справедливе і прозоре встановлення не лише роздрібної ціни, але й цін перепродажу у всьому каналі збуту;

- логістика, яка має бути ефективною для забезпечення фізичного руху товару (обслуговування замовлень, транспортування, утримування складів, утримування запасів і забезпечення наявності всього заявленого асортименту товарів);

- аналіз і контроль, насамперед – контроль за роздрібними цінами, контроль наявності товарів у місцях продажу, контроль якості подання товару в кожному пункті продажу, контроль і аналіз діяльності партнерів виробничого підприємства з погляду дотримання домовленостей, стандартів обслуговування, недопущення внутрішньосистемної конкуренції, демпінгування, завдавання шкоди іміджу товаровиробника тощо, а також аналіз дій конкурентів.

Об'єктом дослідження даної дипломної роботи є логістичні процеси постачання товарів в мережі ТОВ «Товари-К».

Предмет дослідження: розробка та реалізація програмного додатка що будує оптимальний шлях враховуючи карту міста.

Виходячи з обраних об'єкту та предмету дослідження *метою роботи* є покращення економічних показників роботи підприємства за рахунок мінімізації логістичних витрат

Для досягнення поставленої мети в дипломній роботі поставлені наступні задачі:

1. Визначити послідовність обслуговування множини магазинів, вважаючи,

що кожна з них обслуговується окремо і послідовно.

2. Створити програму для розбиття кластеру магазинів на рівні за дистанцією множини для оптимізації доставки

Для розв’язання поставлених задач у роботі застосовуються наступні методи дослідження: метод імітації відпалу та метод пошуку табу для вирішення задачі пошуку мінімального шляху, метод побудови інформаційних систем для застосування існуючого програмного забезпечення у роботу власної інформаційної системи.

Практична цінність роботи полягає в тому, що запропонована та розроблена система скорочує час на побудову оптимального маршруту та скорочує витрати компанії.

Економічний ефект від впровадження розробки очікується позитивним, оскільки це дозволить вивільнити фінансові ресурси компанії, які можна буде використовувати для підтримки інших відділів.

Соціальний ефект від впровадження розробки очікується позитивним через те, що навантаження на водіїв буде зменшена, їх графік буде більш нормований і вони не будуть витрачати зайвий час у дорозі, а також, це зробить внесок у зменшення вуглеводного відбитка.

1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Загальний аналіз діяльності підприємства ТОВ «Товари-К»

Підприємство ТОВ «Товари-К» існує в Україні з 1996 року. До 2000 року компанія мала назву «Корона-Трейд». З самого початку і по сьогодні Товари-К займається дистрибуцією продуктів харчування. У 2000 році її дистриб‘юторський портфель поповнив контракт з відомою на весь світ Компанією «Крафт Фудз Україна» (КФУ).

«Крафт Фудз Україна» – українське підприємство, що входить до групи компаній Mondelēz International, найбільшого у світі виробника шоколадної продукції, печива та цукерок, а також другого найбільшого у світі виробника жувальної гумки. «Крафт Фудз Україна» – це 13 потужних брендів («Dirol», «Milka», «Belvita», «Tuc», «Oreo», «Jacobs», «Picnic», «Halls», «Tassimo», «Carte Noire», «Барні», «Люкс», «Корона»), 5 продуктових категорій (кава, печиво, жувальна гумка, шоколад та батончики) 4 виробництва, близько 1800 постійних працівників. Саме висока популярність брендів та якість продукції забезпечує компаніям Товари-К та КФУ стабільний дохід та довіру клієнтів. Якщо деякі бренди, як, наприклад, «Oreo» мають столітню історію та дуже популярні закордоном, то такі бренди, як «Корона» та «Люкс» є вітчизняними. Їх виробництво розташовано в Україні, у Київській області. КФУ працює в Україні з 1995 року. Із 2003-го КФУ керує розвитком бізнесу на ринку Молдови, з 2005 року - на ринках Білорусі, Грузії, Вірменії та Азербайджану, а з 2008 року – також на ринках Казахстану, Узбекистану, Киргизстану, Таджикистану, Туркменістану та Монголії. Більше 200 мільйонів доларів США становить сума інвестицій «Крафт Фудз Україна» у розбудову бізнесу та економіки України. Ця компанія має найкращі показники зростання та прибутковості в регіоні Центральна та Східна

Європа, Близький Схід та Африка. КФУ на даний момент єдиний і протягом 13 років незмінний партнер компанії Товари-К.

У ході свого динамічного розвитку Компанія Товари-К не тільки перейняла передовий досвід ведення бізнесу, сформувала систему навчання своїх фахівців, розширила свою представницьку мережу до 11-ти філій, а також отримала визнання серед кращих дистриб'юторів України:

«Підприємство року 2010»,

«Підприємство року 2014».

Філіальна мережа компанії охоплює східну частину України, а саме філії розташовані у наступних містах: Дніпропетровськ, Запоріжжя, Кіровоград, Олександрія, Дніпродзержинськ, Бердянськ, Нікополь, Павлоград, Мелітополь, Дніпрорудне, Кривий Ріг. Кожна філія має свій власний склад, де зберігається у належних умовах необхідна для певного регіону кількість продукції. Крім філій компанія має ще 2 представництва: перше - у Кривому Розі – центральний офіс, та друге – у Дніпропетровську – комерційний відділ.

Головними конкурентоспроможними перевагами Компанії "Товари-К" є наявність більш ніж 15 000 торгових точок, присутність різних каналів збуту, таких як опт, роздріб, кіоски, нестандартні канали збуту (до них відносяться аеропорти, вокзали, університети та інші навчальні заклади, тощо), робота з великими супермаркетами («АТБ», «Varus», «Spar») і дрібними торговими точками.

Налагоджена система логістики допомагає забезпечити повну наявність товару в будь-якому регіоні Компанії, а розробка і проведення акцій сприяє плідній співпраці з клієнтами.

Основою Компанії є високі стандарти якості, передовий підхід до бізнесу, розвиток взаємовигідних відносин з торговими партнерами та клієнтами, а також здатність оцінити значущість кожного співробітника.

Уся продукція компанії Товари-К є швидко оборотною, та реалізується на так званому ринку CPG.

CPG (Consumer Packaged Goods) – це товари повсякденного попиту: продукти харчування, побутова хімія, пиво і сигарети. Одним словом, ширвжиток. Професіонали люблять називати такі товари «фастмувери». Насправді CPG – це всього лише товари з високою оборотністю. Вони недорогі, і люди змушені купувати їх кожен день. CPG продаються в супермаркетах і використовуються в повсякденному житті.

Сьогодні можна спостерігати декілька явних тенденцій на ринку CPG:

- уповільнення темпів зростання галузі;
- інтенсивна боротьба за частку ринку між виробниками;
- зміна форматів роздрібної торгівлі (збільшення частки мережевого роздробу);
- прискорення темпів зростання впровадження новинок при скороченні життєвого циклу товарів.

Компанія Товари-К функціонує на ринку CPG України протягом 13 років. Отже, задачею та основною функцією компанії є дистриб`юція, або розповсюдження продуктів харчування на ввіреній їй території. Розповсюдження йде пліч о пліч з логістикою, тому опис структури функціонування почнемо з комерційного відділу компанії, який знаходиться у Дніпропетровську.

Для початку розглянемо загальну структуру компанії в розрізі її відділів:

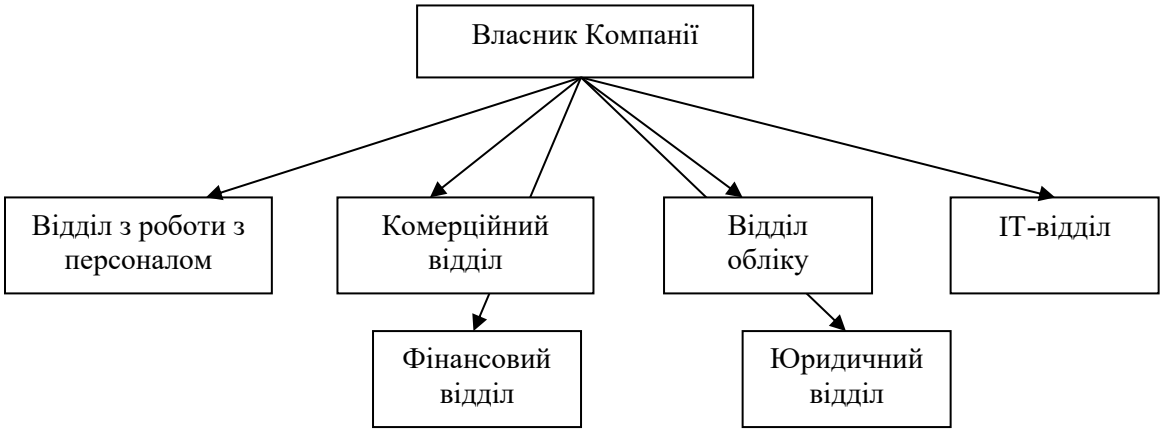

Рис. 1.1 Загальна структура компанії Товари-К

Структура комерційного відділу має вигляд:

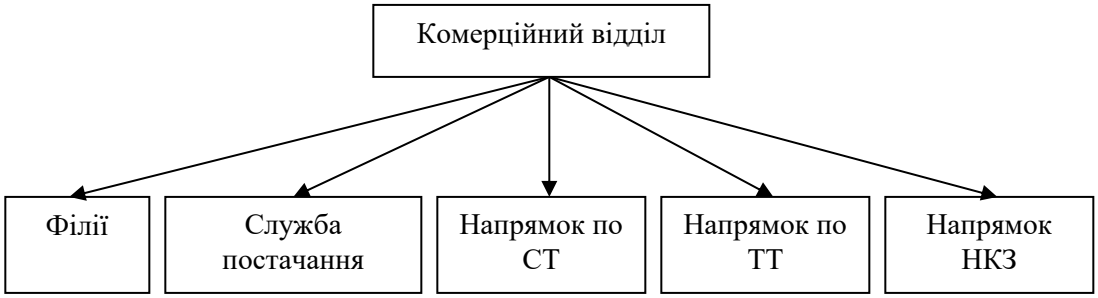

Рис. 1.2 Структура комерційного відділу компанії Товари-К

Основні функції, які виконуються у комерційному відділі – це робота з ключовими клієнтами – найбільші клієнти, які приносять найбільше прибутку, а саме: «АТБ», «Varus», «Spar» - напрямок по сучасній торгівлі (або канал СТ), напрямок по традиційній торгівлі, а саме роздрібні магазини (або канал ТТ), напрямок по нетрадиційним каналам збуту – навчальні заклади, аеропорти, вокзали (напрямок НКЗ).

Своєчасне забезпечення складів усією продукцією виконує служба постачання. У дистрибуції логістика є одною з найважливіших задач. Фахівці з логістики слідкують за наявністю продукції на кожному складі, формують заявки

на продукцію. Задача не з легких, тому що необхідно, щоб товару було завжди достатньо, але, в той же час, його не повинно бути багато, щоб уникнути добігання до кінця терміну придатності, потайки. У цьому фахівцям допомагає програма «1С», де відображаються залишки на складах, а також замовлення клієнтів у кожному місті. Після формування замовлення відправляється на обробку до КФУ у Київ, де завантажуються фури та доставляють продукти до необхідних складів.

Якщо у комерційному відділі виконується керування запасами на відстані, то пряма взаємодія з товаром виконується на філіях. Як було сказано раніше, компанія має 11 філій у Дніпропетровській, Кіровоградській та Запорізькій областях. Розглянемо детальніше структуру філії на прикладі Дніпропетровської.

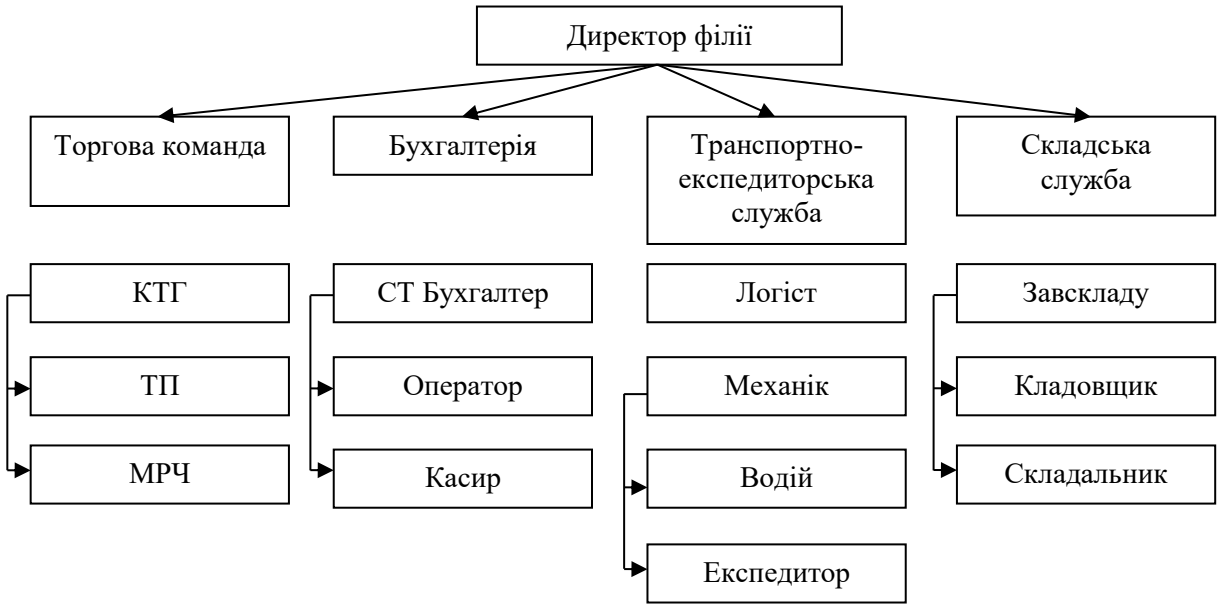

Рис. 1.3 Структура Дніпропетровської філії компанії Товари-К

Директор філії в першу чергу контролює процес продажів та роботу усіх служб. Торгова команда складається з КТГ – керівника торгової групи, ТП – торговельних представників, та МРЧ – мерчендайзерів. Торговельні представники та мерчендайзери співпрацюють напряму з торговими точками. До їх обов’язків

входить збір заявок за допомогою КПК, продаж продукції, встановлення «хот-зон». У кожного торговельного представника є план продажу на місяць, який формується згідно з плану продажу компанії за рік.

Керівник торгової групи (КТГ) слідкує за роботою торговельних представників, керує процесами продажів у своїй торговельній групі. Бухгалтерія займається оформленням замовлень клієнта та занесенням їх у базу. Ці функції виконують оператори та касири. Старший бухгалтер філії контролює роботу відділу та веде облік торгово-матеріальних цінностей. Наступна не менш важлива служба – це транспотро-експедиторська. Вона складається з логіста, який контролює постачання, декількох водіїв, які доставляють замовлення клієнтам, експедиторів та механіків.

Складська служба виконує зберігаючі функції, здійснює прийом, видачу та збірку товару, складається з завскладу, комірників та складальників. Ще один офіс, центральний, компанія має у Кривому Розі.

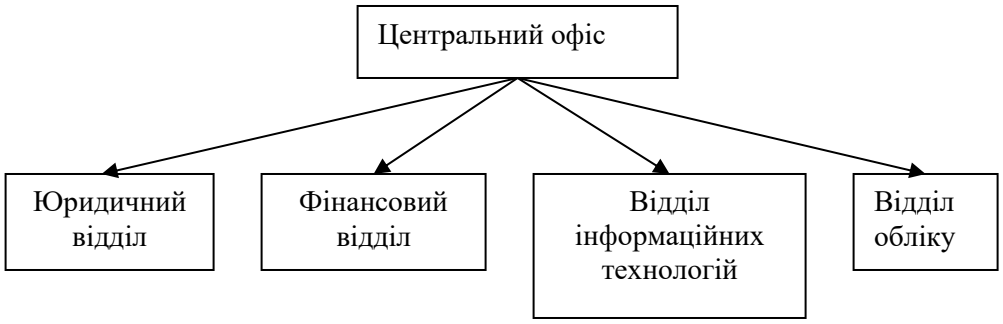

Рис. 1.4 Структура центрального офісу компанії Товари-К

Відділ інформаційних технологій займається підтримкою та вдосконаленням різноманітних баз компанії, 1С, розробкою сайта, усуненням неполадок, обслуговуванням КПК торговельних представників.

Відділ обліку веде облік товарно-матеріальних цінностей на всьому підприємстві.

Фінансовий відділ, окрім прямих фінансових операцій, займається заведенням акцій у «1С».

Юридичний відділ складається з двох працівників: юриста та його молодшого помічника. Вони виконують всі операції, зв`язані з юридичною діяльністю підприємства.

Як вже було сказано раніше, найважливіші стратегічні рішення приймає власник компанії разом з працівниками комерційного відділу, а саме менеджерами з ключових клієнтів, менеджером з традиційної торгівлі та нетрадиційних каналів збуту. Власник компанії вирішує глобальні питання, пов`язані з планом продажів, а також делегує свої повноваження менеджерам та контролює їх роботу.

План продажів на рік, надходить від компанії KFU. У цьому документі свідчить про те, який обсяг товару кожної категорії необхідно продати на ввіреній території. Згідно з цим документом в залежності від пори року та опираючись на досвід, план розбивають на 12 місяців та 11 філій.

В залежності від того, як добре чи погано продається певний товар, у комерційному відділі приймаються рішення щодо введення акцій та строку їх дій.

Окремим дуже важливим нюансом є слідкування за дебіторською заборгованістю. Майже завжди товар відгружається певному клієнтові в борг. По мірі продажу товару, клієнт зобов`язаний виплачувати гроші. Але існують ситуації, коли товар не продається у повному обсягу або клієнт не виплачує необхідну суму. У цьому випадку виникає дебіторська заборгованість. Відповідальність за неї повністю лягає на директора філії.

В загальному вигляді структура управління та прийняття рішень має наступний вигляд:

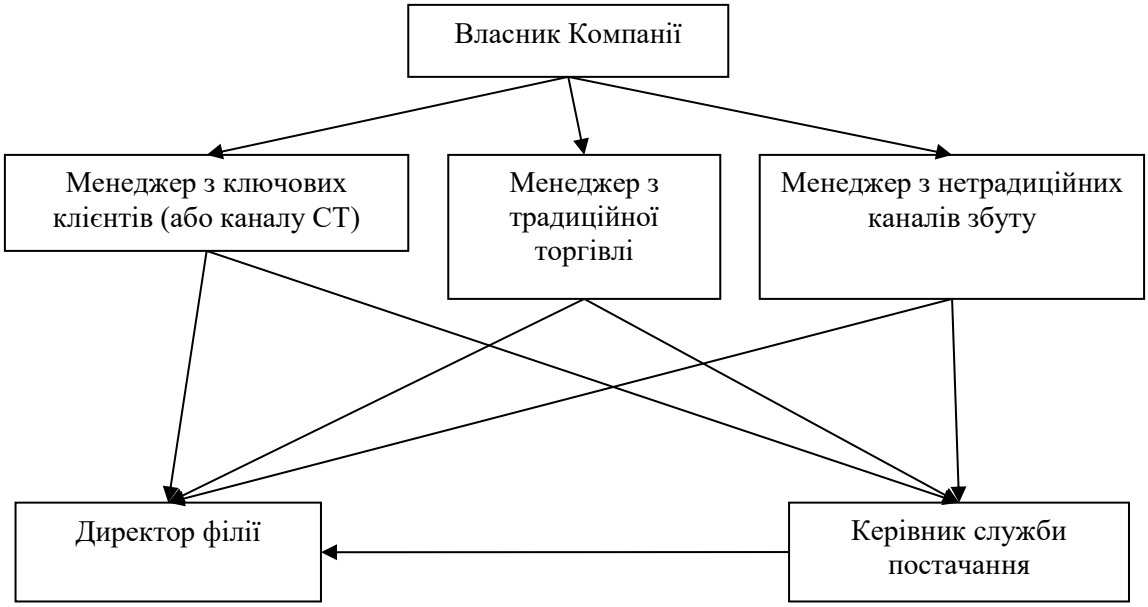

Рис. 1.5 Структура управління та прийняття рішень

Щодо продажів - все достатньо прозоро. Компанія має план та вона повинна його виконати, а краще перевиконати. Щодо прийняття рішень стосовно забезпечення продукцією складів – тут виникають певні питання та складності.

Кожна з 11-ти філій має у своєму розпорядженні склад, де у відповідних умовах зберігається товар. Задача наповнення складів вирішується у відділі постачання. Керування запасами для усіх складів здійснюють 2 логіста та їх керівник. У базі «1С» містяться дані про замовлення усіх клієнтів. Згідно з цим логісти, оцінюючи залишки на складах та реальну потребу у товарі, формують замовлення до KFU.

Замовлення товару на склади робиться з періодичністю у кожен тиждень по окремій філії.

Контроль запасів можна охарактеризувати як безперервний, тому що ведеться облік простроченого товару.

Управління запасами можна охарактеризувати як групове. Це пояснює той факт, що машина з замовленнями послідовно розвозить товар по всім потребуючим того філіям, а не є індивідуальною для кожного міста.

Замовлення ключових та великих клієнтів («АТБ», «Varus», «Spar» та ін..) доставляються на власних машинах компанії (ГАЗ 3301). Щодо доставки товару у роздрібні невеликі магазини та кіоски, то вона здійснюється самими торговельними представниками на їх власних авто.

Опираючись на те, що компанія нараховує 11 філій, 14 офісів, більш ніж 600 працівників та 15 000 торгових точок, можна тільки уявити, скільки складних задач доводиться вирішувати усім працівникам кожен день.

Окремою областю задач є процес формування продажів. Оскільки план продажів надходить від виробників – КФУ, то для компанії Товари-К стоїть нелегка задача оптимального розподілення обсягу товару, якого необхідно продати у певний місяць. Ця задача є достатньо проблемною, тому що продажі у окремому місяці залежать від багатьох факторів, таких як

- ціна конкурентів;
- власна ціна;
- пора року;
- престиж бренду та ін.

Отже є сенс вирішувати цю задачу як задачу багатокритеріальної оптимізації.

Другою проблемою є обмежений розмір вантажівки, яка доставляє товар до клієнтів. Часто виникають ситуації, коли вантажівки їздять наполовину порожні, що веде за собою додаткові витрати на пальне, втрату часу. Оскільки замовлення бувають достатньо великими, то є сенс вирішувати задачу багатономенклатурного керування запасами з обмеженням на розмір вантажівки.

Третьою проблемою, а точніше класом проблем, є задачі керування логістичними операціями. Саме ці питання й будуть вирішуватися у даній роботі.

1.2 CPG компанії та їх проблематика

CPG компанії, що займаються виробництвом та продажем товарів повсякденного попиту, таких як продукти харчування, напої, побутова хімія тощо, стикаються з рядом проблем у галузі логістики. Приведемо приклад таких проблем.

1.Проблеми в управлінні запасами. Компанії CPG мають справу з великою кількістю артикулів (одиниць зберігання запасів), що дуже ускладнює управління запасами. Необхідно точно знати, скільки продуктів знаходиться на складі, в дорозі і на полицях магазинів. Неправильний облік товарів може привести до недостач або надлишків товарів, що може знизити ефективність роботи компанії і привести до фінансових втрат.

Одним із способів управління товарними запасами є використання спеціалізованих систем управління складом, які дозволяють відстежувати рух товарів на складі, контролювати рівень запасів і прогнозувати попит. Це дозволяє CPG-компаніям більш точно планувати виробництво і управляти запасами, щоб мінімізувати надлишки товарів і знизити витрати.

Існують також інноваційні технології, такі як мітки RFID і безпілотні транспортні засоби, які можуть допомогти в управлінні запасами. RFID-мітки дозволяють швидко і точно знаходити товари на складі, а безпілотні транспортні засоби можуть забезпечити автоматичне і точне транспортування товарів між різними складами і пунктами призначення.

2.Оптимізація маршрутів доставки. Доставка товарів споживачам є однією з ключових задач CPG компаній. Однак оптимізація маршрутів доставки, щоб зменшити витрати на транспортування та збільшити швидкість доставки, може бути складною задачею.

Для оптимізації маршрутів доставки можуть бути використані спеціалізовані системи управління транспортом, які можуть розраховувати оптимальні маршрути доставки з урахуванням різних факторів, таких як відстань, кількість замовлень, наявність заторів тощо. Це дозволяє скоротити час доставки та витрати на транспортування товарів.

Також CPG компанії можуть використовувати маршрутні листи та GPS-навігацію, щоб оптимізувати маршрути.

Інший спосіб оптимізації маршрутів доставки - використання аналітики та машинного навчання. Аналітика може допомогти визначити найчастіше замовляні товари та найпопулярніші місця доставки, що дозволить оптимізувати маршрути доставки та скоротити час на доставку товарів. Машинне навчання може допомогти компаніям прогнозувати попит на продукти, що дозволить більш точно планувати маршрути доставки та скоротити кількість неуспішних спроб доставки.

3.Конкуренція на ринку та необхідність. CPG компанії повинні дотримуватись суворих вимог якості та безпеки продуктів, що встановлюються законодавством та ринком. Це вимагає дотримання різних стандартів та сертифікацій, що може бути складним та витратним процесом.

Щоб дотримуватись вимог якості та безпеки, CPG компанії можуть використовувати системи управління якістю, які дозволяють відстежувати виробничі процеси, контролювати якість та безпеку продуктів, а також взаємодіяти з контролюючими органами.

Також CPG компанії можуть використовувати інноваційні технології, такі як датчики та системи відеоспостереження, для контролю якості та безпеки продуктів. Датчики можуть допомогти відстежувати температуру та вологість у сховищах та транспорті, що може знизити ризик порушення продуктів. Системи відеоспостереження можуть забезпечити контроль за виробничими процесами та забезпечити швидку реакцію на будь-які проблеми, пов'язані з якістю та безпекою продуктів.

4.Складнощі у забезпеченні стійкості та екологічної відповідальності. CPG компанії повинні забезпечувати стійкість та екологічну відповідальність у своїй діяльності. Це означає, що компанії повинні вживати заходів щодо зниження екологічного впливу та використання стійких матеріалів та виробничих процесів.

Для забезпечення стійкості та екологічної відповідальністю CPG компанії можуть використовувати нові технології, такі як сенсори та системи моніторингу енергоспоживання, визначення точок споживання енергії та ідентифікації можливостей для підвищення ефективності. Також компанії можуть використовувати пакувальні матеріали, які можна переробляти або використати повторно.

Деякі компанії CPG також можуть використовувати так званий "зелений маркетинг", який орієнтований на екологічну відповідальність і стійкість. Це може включати просування екологічно чистих продуктів, упаковок, які можна переробляти, та інших інновацій у сфері стійкості та екологічної відповідальності.

5. Зростання конкуренції та необхідність інновацій. CPG компанії стикаються зі зростанням конкуренції на ринку, що вимагає від них розробки та впровадження інноваційних рішень, щоб залишатися конкурентоспроможними. Це може включати використання нових матеріалів, поліпшення якості продуктів, розробку нових продуктів і упаковок, а також використання нових технологій у виробничих процесах.

Для вирішення цих проблем CPG компанії можуть використовувати інноваційні підходи до виробництва та дистрибуції продуктів, такі як 3D-друк, автоматизація виробництва, дронові доставки та інші. Також компанії можуть зміцнити своє дослідження та розробку, щоб створювати нові продукти та упаковки, які відповідають очікуванням споживачів.

В цілому, CPG компанії стикаються з безліччю складнощів у галузі логістики, таких як управління запасами, оптимізація маршрутів доставки,

дотримання вимог якості та безпеки, забезпечення стійкості та екологічної відповідальності, а також зростання конкуренції та необхідність інновацій.

Однак, з використанням інноваційних підходів та технологій, CPG компанії можуть подолати ці проблеми та забезпечити ефективну логістику, яка відповідає потребам споживачів та підвищує їхню задоволеність. Наприклад, використання систем управління запасами та маршрутизації доставки може допомогти компаніям оптимізувати свої процеси та знизити витрати, а використання нових технологій, таких як дронові доставки та 3D-друк, може надати нові можливості для прискорення виробництва та доставки продуктів.

Крім того, важливо, щоб CPG компанії прагнули сталого розвитку та екологічної відповідальності, впроваджуючи інноваційні рішення в галузі стійкості та екологічної відповідальності. Це може покращити імідж компанії в очах споживачів та підвищити задоволеність та лояльність клієнтів.

Зрештою, зростання конкуренції на ринку вимагає від CPG компаній постійного інноваційного розвитку та пошуку нових можливостей для покращення якості продуктів та послуг. Інноваційні рішення, такі як дослідження та розробка нових продуктів, упаковок та технологій, можуть допомогти компаніям залишатися конкурентоспроможними на ринку та залучати нових клієнтів.

В цілому, CPG компанії стикаються з низкою складнощів у галузі логістики, але за допомогою інноваційних рішень та підходів вони можуть подолати ці проблеми та забезпечити більш ефективну логістику, яка відповідає потребам споживачів та підвищує їхню задоволеність.

1.3 Задача комівояжера в умовах CPG компаній

Задача комівояжера - одна з найвідоміших проблем в області оптимізації маршрутів. Це завдання полягає в тому, щоб знайти найкоротший маршрут, який проходить через заданий набір точок і повертається до вихідної точки. Завдання комівояжера має безліч застосувань в різних областях, включаючи логістику, транспорт, виробництво і маркетинг. Ми розглянемо застосування задачі комівояжера в компаніях CPG.

Ці компанії стикаються з низкою складних завдань, пов'язаних з управлінням виробничими процесами, логістикою та маркетингом. Задача комівояжера може допомогти CPG-компаніям оптимізувати логістику і знизити витрати на доставку товарів.

У CPG-компаніях існує безліч факторів, які впливають на ефективність логістичних операцій. Деякі з них включають відстань між складами і точками продажів, кількість позицій, які необхідно доставити, обмеження за часом і обсягом. Для вирішення цих завдань CPG-компанії можуть використовувати задачу комівояжера.

Застосування задачі комівояжера в CPG компаніях CPG може бути корисно в декількох областях. По-перше, задача комівояжера може допомогти оптимізувати маршрути доставки товарів, що дозволить скоротити транспортні витрати і скоротити терміни доставки. По-друге, використання задачі комівояжера може допомогти CPG компаніям визначити оптимальні місця розташування складів і точок продажів, що також може знизити транспортні витрати і поліпшити обслуговування клієнтів.

Щоб застосувати завдання комівояжера в компаніях CPG, необхідно виконати ряд кроків. По-перше, необхідно зібрати дані про відстані між складами і точками продажів, обсяги товару і терміни доставки.

Ці дані можна збирати і аналізувати за допомогою спеціалізованих програмних засобів, які дозволяють оптимізувати маршрут і розрахувати оптимальний час доставки вантажів.

Крім того, задачу комівояжера можна вирішити за допомогою штучного інтелекту (ШІ). Алгоритми ШІ можуть розраховувати оптимальні маршрути, беручи до уваги безліч факторів, таких як відстань, час, кількість замовлень і навіть прогноз погоди. Це дозволяє оптимізувати процес доставки і знизити транспортні витрати.

Для CPG-компаній задачі комівояжера має велике значення, так як доставка продукції є ключовим етапом в процесі продажів. Оптимізація маршрутів і скорочення термінів доставки дозволяє компаніям підвищити свою конкурентоспроможність, знизити транспортні витрати і підвищити обслуговування клієнтів.

Крім того, задача комівояжера може використовуватися для оптимізації процесів всередині компанії. Наприклад, CPG-компанії можуть використовувати задачу комівояжера, щоб впорядкувати процес комплектації замовлень на складах. Роботи, оснащені датчиками і камерами, можуть визначати оптимальні маршрути складання товарів на складі, що підвищує ефективність процесу і скорочує терміни доставки замовлень.

Також задача комівояжера може бути використана для оптимізації виробничих процесів. Наприклад, компанії CPG можуть використовувати задачу комівояжера для оптимізації маршрутів доставки сировини і матеріалів на виробничі лінії. Це дозволяє поліпшити виробничий процес, скоротити час на доставку матеріалів і знизити транспортні витрати.

На закінчення, задача комівояжера є важливим завданням для CPG-компаній, які прагнуть підвищити свою конкурентоспроможність і оптимізувати процеси.

Вона дозволяє оптимізувати маршрути доставки, знизити транспортні витрати, підвищити клієнтський сервіс і поліпшити імідж компанії.

Як вже було сказано, в задачі комівояжера входить мінімізація довжини шляху, а значить, і мінімізація витрат на доставку товару. За допомогою

алгоритмів оптимізації маршрутів компанії можуть скоротити час, необхідний для доставки товару і, отже, знизити транспортні витрати. Крім того, оптимізація маршрутів дозволяє збільшити кількість поставок в певний період часу, що може привести до підвищення рівня обслуговування клієнтів і зміцнити їх лояльність до компанії.

Наприклад, компанія Coca-Cola впровадила систему маршрутизації з використанням алгоритмів оптимізації, що дозволило скоротити кількість кілометрів, які необхідно проїхати при доставці напоїв, на 20%. Це призвело до значної економії витрат на паливо і машини, а також збільшення кількості замовлень, які компанія могла обробити за день.

Крім того, оптимізація маршрутів може допомогти компаніям поліпшити свій клієнтський досвід. Швидша та ефективніша доставка може призвести до підвищення задоволеності клієнтів, а це, у свою чергу, може зміцнити лояльність до їхнього бренду та призвести до збільшення продажів.

Нарешті, використання алгоритмів оптимізації маршрутів може поліпшити імідж компанії. Компанії, які можуть забезпечити швидку, ефективну та точну доставку, можуть сприйматися споживачами як надійні та професійні. Це може привести до підвищення довіри споживачів до бренду і підвищення його привабливості на ринку.

На закінчення, задача комівояжера важлива для CPG-компаній, яким необхідно оптимізувати процес доставки свого товару. З використанням алгоритмів оптимізації маршрутів компанії можуть швидко і ефективно вирішити задачу комівояжера. Вона дозволяє оптимізувати маршрути доставки, знизити транспортні витрати, підвищити обслуговування клієнтів і скоротити час на доставку товару.

1.4 Формальне визначення задачі комівояжера

Подання у вигляді графа

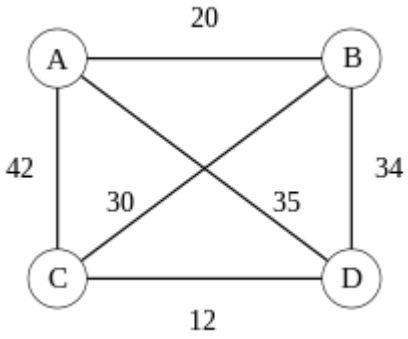

Рис 1.6 Приклад графа на 4 точки

Симетрична задача для чотирьох міст.

Для можливості застосування математичного апарату для розв'язання проблеми, її слід представити у вигляді математичної моделі. Проблему комівояжера можна представити у вигляді моделі на графі, тобто, використовуючи вершини та ребра між ними. Таким чином, вершини графа (на мал.: від A до D) відповідають містам, а ребра (I,J) між вершинами I та J сполучення між цими містами. У відповідність кожному ребру (I, J) можна зіставити вагу $C_{ij} \geqslant 0$ (на мал. : 20, 42, …), яку можна розуміти як, наприклад, відстань між містами, час або вартість подорожі. Маршрутом (також гамільтоновим маршрутом) називається маршрут на цьому графі до якого входить по одному разу кожна вершина графа. Задача полягає у відшуканні найкоротшого маршруту.

З метою спрощення задачі та гарантії існування маршруту, зазвичай вважається, що модельний граф задачі є повним, тобто, що між довільною парою вершин існує ребро. Це можна досягти тим, що в тих випадках, коли між окремими містами не існує сполучення, вводити ребра з максимальною вагою (довжиною, вартістю тощо). Через велику довжину таке ребро ніколи не потрапить до оптимального маршруту, якщо він існує.

В залежності від того, що зіставляється вазі ребер, розрізняють різні варіанти задачі, найважливішими з яких є симетрична та метрична задачі.

Асиметрична та симетрична задачі

В загальному випадку, *асиметрична задача комівояжера* відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.

У випадку *симетричної задачі* всі пари ребер між одними й тими самими вершинами мають однакову вагу, тобто, для ребра (I,j) ваги однакові $C_{ij} = C_{ij}$. Як наслідок, всі маршрути мають однакову довжину в обидва напрямки. В симетричному випадку кількість можливих маршрутів вдвічі менша за асиметричний випадок. Симетрична задача моделюється неорієнтованим графом (як показано на малюнку).

Насправді, задача комівояжера у випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів в залежності від напряму руху.

Метрична задача

Симетричну задачу комівояжера називають *метричною*, якщо відносно довжин ребер виконується нерівність трикутника. Умовно кажучи, в таких задачах обхідні шляхи довші за прямі, тобто, ребро від вершини (I) до вершини (j) ніколи не довше за шлях через проміжну вершину k:

$$c_{ij} \leqslant c_{ik} + c_{kj} \tag{1.1}$$

Така властивість довжини ребер визначає вимірний простір на множині ребер та міру віддалі, що задовольняє інтуїтивне розуміння відстані.

Поширені на практиці функції віддалі є також метриками і задовольняють нерівності трикутника:

- Евклідова відстань в *евклідовій задачі комівояжера*,

- Манхеттенська метрика (також квартальна метрика) *прямокутної задачі комівояжера*, в якій відстань між вершинами на ґратці дорівнює сумі відстаней вздовж осі ординат та абсцис,
- Відстань Чебишова, яка визначає віддаль між вершинами решітчастого графа як максимальне значення відстані вздовж осі ординат та абсцис.

Дві останні метрики знаходять застосування, наприклад, під час свердління отворів в друкованих платах коли верстат має зробити якнайбільше отворів за найменший час і може пересувати свердло незалежно в обох напрямах для переходу від одного отвору до наступного. Манхеттенська метрика відповідає випадку, коли пересування в обох напрямах відбувається послідовно, а максимальна — випадку коли пересування в обох напрямах відбувається синхронно а загальний час дорівнює максимальному часу пересування вздовж осі ординат або абсцис.

Не-метрична задача комівояжера може виникати, наприклад, у випадку мінімізації тривалості подорожі за наявності вибору транспортних засобів в різних напрямах. В такому випадку обхідний шлях літаком може бути коротший за пряме сполучення автомобілем.

Якщо, на практиці, в умовах задачі дозволяється відвідувати міста декілька раз, то симетричну задачу можна звести до метричної. Для цього задачу розглядають на так званому *графі відстаней*. Цей граф має таку саму множину вершин як і вихідний та, на доданок, є повністю зв'язним. Вага ребер C_{ij} між вершинами (i) та (j) на графі відстаней відповідає вазі найліпшого сполучення між вершинами (i) та (j)у вихідному графі. Для визначених в такий спосіб ваг C_{ij} виконується нерівність трикутника, і кожному маршруту на графі відстаней завжди відповідає маршрут з можливими повтореннями вершин у вихідному графі.

Формулювання у вигляді задачі дискретної оптимізації

Одним з підходів до розв'язання задачі є формулювання її у вигляді задачі дискретної оптимізації, при цьому розв'язки представляються у вигляді змінних, а зв'язки у вигляді відношень нерівності між ними. Таким чином, можливо декілька варіантів. Наприклад, симетричну задачу можна представити у вигляді множини ребер V. Кожному ребру $\{i, j\}$ зіставляється двійкова змінна $x_{ij} \in \{0,1\}$, яка дорівнює 1 якщо ребро належить маршруту та 0 в протилежному випадку. Довільний маршрут можна представити у вигляді значень множини змінних приналежності, але не кожна така множина визначає маршрут. Умовою того, що значення множини змінних визначають маршрут є описані далі лінійні нерівності.

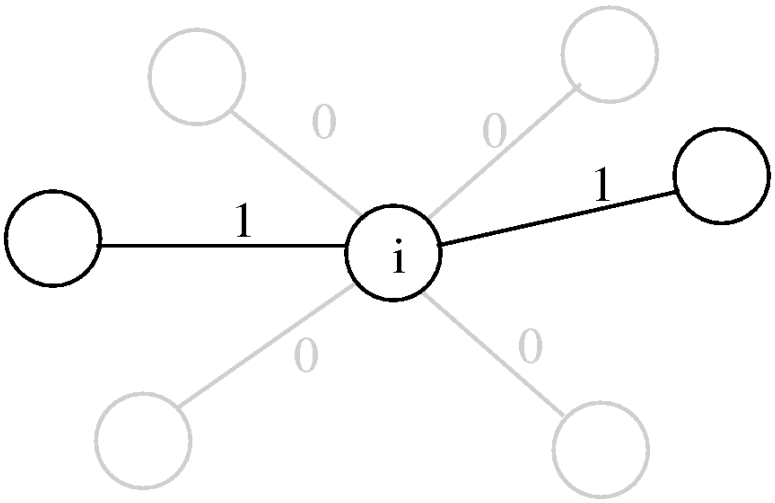

Рис 1.7 Умова кратності: Кожна вершина повинна мати один вхід і один вихідний край маршруту.

Кожна вершина має сполучатись через пару ребер з рештою вершин, тобто, через вхідне та вихідне ребро:

$$\forall i \in V, \sum_{j \in V \setminus \{i\}} x_{ij} = 2 \tag{1.2}$$

В сумі кожний доданок x_{ij} дорівнює або 1 (належить маршруту) або 0 (не належить). Тобто, отримана сума дорівнює кількості ребер в маршруті, таких що мають вершину (i) на одному з кінців. Вона дорівнює 2, оскільки кожна вершина має вхідне та вихідне ребро.

У наведеному малюнку вершина (i) показана з вхідним та вихідними ребрами, а ребра маршруту відмічено товстішими лініями. Поруч з ребрами вказано ваги x_{ij} що додаються до вказаної вище суми.

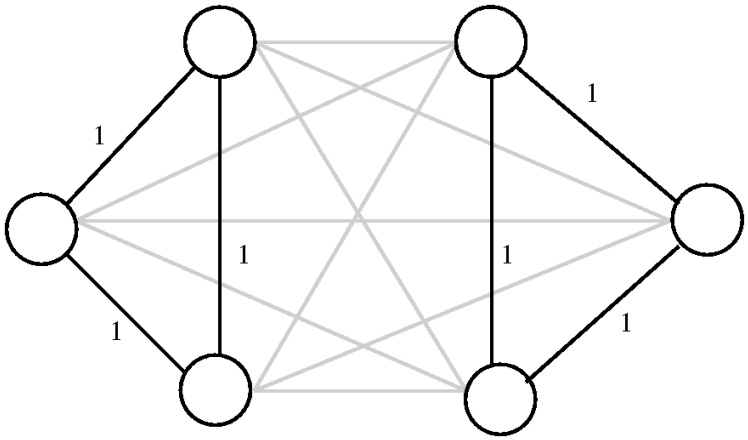

Рис 1.8 Петлі: змінні задовольняють умові кардинальності, але не визначають маршрут.

Описані раніше *умови кратності* виконуються не лише маршрутами, а й значеннями змінних, що відповідають відокремленим циклам, де кожна вершина належить лише одному циклу (див. малюнок). Аби уникнути подібні випадки, мають виконуватись так звані *нерівності циклів* (або *умови усунення підмаршрутів*), які було визначено Данциґом, Фалкерсоном та Джонсоном в 1954 році під назвою *умови петель* Цими нерівностями визначалась додаткова умова того, що кожна множина вершин $s \subset v$, є або порожньою, або містить всі вершини, що сполучається з рештою вершин через щонайменше два ребра:

$$\sum_{i \in S,\ j \notin S} x_{ij} \geq 2 \tag{1.3}$$

для всіх множин вершин S де $1 \leq |S| \leq |V| - 1$. Ця сума дорівнює сумі ваг ребер маршруту між вершиною $i \in S$ та вершиною $j \notin S$. Аби усунути зайві нерівності, можна обмежитись множинами вершин S з щонайменше двома та щонайбільше $|V| - 2$ вершинами. На малюнку(4) ребра $\{i, j\}$ з вагами $x_{ij} = 1$ відмічено товстішими лініями а решта ребер має вагу $x_{iJ} = 0$. Введення додаткових умов (мал 4) для множини вершин S, що складається з трьох лівих вершин, буде гарантувати, щоб S сполучалась через щонайменше два ребра маршруту з трьома вершинами справа, аби усунути обидва цикли. Кількість нерівностей усунення циклів відповідно до Данциґа, Фалкерсона та Джонсона дорівнює $2^n - 2(n - 1)$.

В 1960 році Міллєр Такер та Землін винайшли альтернативні умови усунення підшляхів шляхом введення n нових змінних, що визначають порядок відвіданих міст, що потребує лише $n^2 - n + 1$ додаткових нерівностей. Більше того, через двійковість x_{ij} у формулюванні Міллєра, Такера та Земліна задача комівояжера залишається NP-складною.

Так, кожен вектор $x = \left(x_{ij}\right)_{i,j \in V}$ з елементами, що дорівнюють 0 та 1, що задовольняє всім нерівностям визначає коректний маршрут, що є розв'язком переформульованої задачі комівояжера: обчислити

$$\min \left\{ \sum_{i \in V} \sum_{j \in V \setminus \{i\}} c_{ij} x_{ij} \mid x \text{ valid (1) (2)}, x_{ij} \in \{0, 1\} \right\} \tag{1.4}$$

Оскільки змінні x_{ij} мають значення лише 0 та 1, сума дорівнює загальній довжині C_{ij} ребер $\{i, j\}$ що належать маршруту.

Кількість нерівностей типу (1.3) зростає експоненційно разом зі збільшенням кількості міст, оскільки майже кожна з $2^{|v|}$ підмножин вузлів визначає одну нерівність. Цю проблему можна вирішити застосуванням методу

відсічення площиною завдяки якому нерівності додаються лише тоді, коли ці нерівності дійсно необхідні. З геометричної точки зору, лінійні нерівності можна представити як гіперплощину в просторі змінних. Множина векторів, що задовольняють цим нерівностям утворюють в такому просторі політоп (багатовимірний багатогранник), або багатовимірний багатокутник, точна форма визначається вагами C_{ij} і є в основному невідомим. Однак, можна показати, що умови (1.2) та (1.3) визначають грані (фацети) політопа, тобто, бічні поверхні політопа з найвищою розмірністю. Тому вони належать до найсильніших лінійних нерівностей, що можуть описувати маршрут. Також існує багато різних граней, що визначаються нерівностями, відомими лише у небагатьох випадках. Хоча (1.2) та (1.3) разом з обмеженнями повністю моделюють проблему лише для двійкових векторів, ці нерівності можуть використовуватись в методі гілок і меж аби відкинути розв'язки методами лінійної оптимізації з не цілими координатами.

1.5 Методи вирішення завдання комівояжера

Існує безліч методів вирішення завдання комівояжера, які використовуються в самих різних сферах, від логістики до біоінформатики. Однак в цілому можна виділити 4 основних методи вирішення завдання комівояжера:

1. Метод перебору
2. Метод найближчого сусіда
3. Спосіб вставки
4. Генетичний алгоритм

Розглянемо кожен спосіб окремо і поговоримо про кожен з них.

Метод перебору - один з найпростіших і інтуїтивно зрозумілих методів вирішення завдання комівояжера. Суть методу полягає в тому, що необхідно розглянути всі можливі варіанти маршрутів і вибрати оптимальний з них. Однак

цей метод має дуже високу обчислювальну складність, яка різко зростає з кількістю міст. Також повний пошук не гарантує знаходження оптимального рішення в розумні терміни при великій кількості міст. для розуміння Давайте розглянемо простий приклад. Припустимо, що у нас є 4 міста і нам потрібно знайти найкоротший маршрут, який проходить через всі міста і повертається в початкове місто. У цьому випадку всього можливо 4! (чотири факторіали) різних маршрутів, тобто 24 варіанти. Для кожного з цих варіантів необхідно розрахувати загальну відстань між усіма містами і вибрати найменший з них. Наступне завдання - визначити всі можливі маршрути. Це можна зробити, перебираючи всі перестановки багатьох міст, починаючи з початкового міста і повертаючись до нього. Наприклад, є 4! = 24 різних маршрути. Процес перебору починається з вибору першого міста. Потім з інших міст вибирається наступне найближче місто, і цей крок повторюється до тих пір, поки не будуть відвідані всі міста. Нарешті повертаємося до початкового міста. Для кожного можливого маршруту необхідно розрахувати його довжину. Це робиться шляхом підсумовування відстаней між кожною парою послідовних міст. Потім вибирається маршрут з найменшою протяжністю. Метод перебору може бути застосований до завдань комівояжера з будь-якою кількістю міст, однак складність обчислень зростає в геометричній прогресії з кількістю міст. Наприклад, для 20 міст потрібно розглянути 19! (факторіал 19) можливих маршрутів. Це становить близько 121 645 100 408 832 000 варіантів маршрутів, а це означає, що перерахувати всі можливі маршрути на практиці неможливо. Однак груба сила може бути використана для пошуку оптимального рішення для невеликих наборів даних. Цей метод заснований на тому, що для того, щоб знайти оптимальне рішення, потрібно розглянути всі можливі варіанти маршрутів і вибрати той, який має найкоротшу протяжність. Процес вирішення завдання комівояжера методом перебору складається з наступних етапів:

1. Створення матриці відстаней між містами. Ця матриця являє собою

таблицю, в якій кожна клітинка містить відстань між двома містами.

2. Генерація всіх можливих маршрутів. На цьому кроці генеруються всі можливі перестановки міст, тобто всі можливі маршрути.

3. Розрахуйте довжину кожного маршруту. Для кожного сформованого маршруту його довжина розраховується підсумовуванням відстаней між містами.

4. Вибір оптимального маршруту. З усіх згенерованих маршрутів вибирається той, який має найменшу довжину.

Щоб вирішити завдання комівояжера методом перебору, потрібно написати програму, яка буде виконувати вищеописані дії. Однак через величезну кількість можливих маршрутів цей метод працює лише для дуже невеликих наборів даних.

Наприклад, для набору даних з 5 міст метод грубої сили згенерує в цілому 120 можливих маршрутів, які можна зробити за розумний проміжок часу. Однак для набору даних з 20 міст метод грубої сили згенерує 1,2 * 10^16 можливих маршрутів, на виконання яких на звичайному комп'ютері потрібно кілька століть.

Метод найближчого сусіда - один з найпростіших і поширених алгоритмів вирішення завдання комівояжера. В його основі лежить принцип вибору найближчого міста на кожному етапі побудови маршруту.

Алгоритм починається з вибору довільного міста в якості відправної точки і додає найближче вільне місто до маршруту на кожному наступному кроці, поки всі міста не будуть відвідані. Потім маршрут повертається до початкової точки.

Для більш наочного розуміння того, як працює алгоритм, давайте розглянемо приклад на невеликому наборі даних. Припустимо, є 5 міст, заданих координатами на площині:

Таблиця 1.1

Координати міст

Місто	Координати
1	(0,0)
2	(1,1)
3	(2,0)
4	(1,2)
5	(3,1)

1. Виберемо в якості відправної точки довільне місто, наприклад, місто 1.

2. Давайте знайдемо найближче до міста 1 вільне місто. У цьому випадку це місто 2.

3. Перейдіть до міста 2 та повторіть крок 2, щоб знайти наступне найближче вільне місто. Найближче вільне місто до міста 2 - місто 4.

4. Перейдіть до міста 4 і повторіть крок 2. Найближче вільне місто до міста 4 - місто 3.

5. Перейдіть до міста 3 і повторіть крок 2. Найближче вільне місто до міста 3 - місто 5.

6. Перейдіть до міста 5 і повторіть крок 2. Найближче вільне місто до міста 5 - місто 1.

7. Їдемо в місто 1 і завершуємо маршрут.

Таким чином, маршрут буде мати наступну послідовність: 1 → 2 → 4 → 3 → 5 → 1.

Очевидною перевагою методу найближчого сусіда є його простота і швидкість роботи. При виборі найближчого міста на кожному кроці метод найближчого сусіда може привести до неоптимальних рішень, оскільки не враховує загальну протяжність маршруту і може привести до локальних мінімумів. Однак метод найближчого сусіда може бути ефективним при вирішенні

невеликих завдань комівояжера або в якості початкового наближення для більш складних алгоритмів.

Прикладом використання методу next-neighbor може бути оптимізація доставки CPG-продукції компанією в межах міста. Якщо міста знаходяться не так далеко один від одного, метод next-neighbor може бути швидшим і ефективнішим, ніж інші методи оптимізації маршрутів.

Одним із прикладів використання методу next-neighbor є Instacart, компанія, яка займається доставкою їжі. Для оптимізації маршрутів доставки компанія використовує комбінацію методу next-neighbor і більш складних алгоритмів, які враховують більш широкий спектр параметрів, таких як погодні умови, час доби, затори на дорогах та інші фактори.

Одним з головних переваг методу найближчого сусіда є його простота і швидкість. Вона реалізується відносно легко і не вимагає великих обчислювальних ресурсів, що може бути важливо при роботі з великими обсягами даних. Однак, як і у випадку з методом грубої сили, метод найближчого сусіда має свої недоліки і не завжди є оптимальним рішенням складних завдань комівояжера.

Взагалі, метод найближчого сусіда - це простий і швидкий алгоритм вирішення завдання комівояжера. Він може бути ефективним при роботі з невеликою кількістю даних і в якості початкового наближення для більш складних алгоритмів. Однак для більш складних завдань і критично важливих за часом проектів слід використовувати більш просунуті алгоритми оптимізації маршрутів.

Ще один метод вирішення завдання комівояжера - **метод вставки**. Цей метод заснований на тому, що рішення починається з міста, а потім будується маршрут шляхом поступового додавання нових міст.

1. Алгоритм методу вставки:
2. Вибирається початкове місто.
3. З решти міст вибирається найближчий до нинішнього міста.
4. Місто вставляється в маршрут таким чином, щоб загальна протяжність

маршруту була мінімальною.

5. Кроки 2 та 3 повторюються, доки до маршруту не буде додано всі міста.

Для реалізації цього методу необхідно за допомогою деяких евристик вибрати оптимальне місце для вставки міста в маршрут. Однією з таких евристик є метод найменшого збільшення довжини маршруту. Вона полягає в наступному:

1. Вибирається початкове місто.

2. З решти міст вибирається найближчий до нинішнього міста.

3. Для кожної пари міст поточного маршруту він розраховує, як зміниться загальна довжина маршруту, якщо між ними буде вставлене нове місто.

4. Місто вставляється в маршрут таким чином, щоб зміна довжини маршруту було мінімальним.

5. Кроки 2–4 повторюються, доки всі міста не будуть додані до маршруту.

На відміну від методу найближчого сусіда, метод вставки може забезпечити більш оптимальні рішення, але він також більш обчислювально дорогий.

Метод вставки широко використовується в логістиці і транспортних компаніях для оптимізації маршрутів доставки. Такий спосіб дозволяє скоротити кількість кілометрів, які необхідно проїхати при доставці товару, що в свою чергу скорочує витрати на паливо і терміни доставки.

Останнім популярним методом вирішення завдання комівояжера є генетичний алгоритм. Цей метод заснований на ідеях еволюційної біології і використовується для пошуку оптимального рішення в складній задачі оптимізації.

Генетичний алгоритм - це метод оптимізації, заснований на принципах природного відбору і генетичної рекомбінації. Вона починається з генерації початкової популяції рішень, де кожен член популяції представляє потенційне рішення проблеми. Потім відбувається ітеративний процес, який включає наступні

етапи:

1. Оцінка якості кожного рішення в популяції.
2. Вибір оптимальних рішень для схрещування.
3. Схрещування кращих рішень для створення нового потомства.
4. Мутація нащадків для створення різноманітності в популяції.
5. Оцінка якості кожного рішення в новій популяції.
6. Перевірка алгоритму умови зупинки.

Першим кроком генетичного алгоритму є генерація початкової популяції. Для вирішення завдання комівояжера це означає, що кожен член населення представляє порядок обходу міст. Наприклад, якщо міст 10, то кожен член населення буде перестановкою з 10 міст. Перше і останнє місто в перестановці фіксуються, так як маршрут повинен починатися і закінчуватися в одному місті.

Далі оцінюється якість кожного рішення в популяції. Для цього розраховується довжина маршруту, відповідна даній перестановці міст. Чим коротше довжина маршруту, тим краще якість рішення.

Потім підбираються кращі рішення для схрещування. У генетичному алгоритмі це називається «селекцією».

Існує кілька способів вибору оптимальних рішень, наприклад, вибір кращої фітнес-функції, вибір випадкових рішень з ймовірністю, пропорційною їх пристосованості і т. д. Після вибору оптимальних рішень проводиться їх схрещування, яке може відбуватися різними способами, наприклад, за допомогою одноточкового, двоточкового або багатоточкового перетину. Після схрещування відбувається мутація, яка випадковим чином змінює гени рішень. Це дозволяє не застрявати в локальних оптимумах і шукати нові рішення в просторі пошуку.

Після генетичних операцій отримують нові розчини, які порівнюють з існуючими в популяції. Кращі рішення зберігаються, а менш вдалі видаляються з популяції. Таким чином, популяція поступово поліпшується, і знайдено оптимальне рішення проблеми комівояжера.

Застосування генетичного алгоритму для вирішення задачі комівояжера має безліч переваг. По-перше, генетичний алгоритм може працювати з великими обсягами даних, що дає можливість вирішувати завдання комівояжера з великою кількістю міст. По-друге, генетичний алгоритм є ефективним методом вирішення завдання комівояжера, який дозволяє знайти оптимальне рішення в розумні терміни.

Однак у генетичного алгоритму є і свої недоліки. По-перше, він може застрягти в локальних оптимумах, що може привести до неоптимального рішення. По-друге, генетичний алгоритм вимагає великої кількості обчислювальних ресурсів, що може ускладнити його використання на слабких комп'ютерах.

Однак генетичний алгоритм є одним з найбільш ефективних методів вирішення завдання комівояжера, який може бути використаний в самих різних сферах, включаючи CPG-компанії.

1.5 Висновки до розділу

Проблема логістики є однією з найважливіших в організаційному управлінні. Але, як правило, в цій області не існує типових рішень - умови на кожному підприємстві або фірмі унікальні і включають безліч обмежень і різних особливостей. З цим пов'язані і проблеми, що виникають при розробці математичної моделі і визначенні оптимального шляху доставки товарів.

Об’єктом для дослідження були обрані логістичні процеси постачання товарів в мережі ТОВ «Товари-К».

Відповідно предметом дослідження є аналіз та оптимізація логістичних перевезень на підприємстві .

Аналіз господарської діяльності підприємства «Тавари-К» показав, що Логістика занадто перевантажена і працює неефективно, що є причиною великих

витрат, оптимізація яких вивільнить значну суму фінансів для інших напрямків

У собівартості продукції значну частину мають логістичні операції доставки товарів на магазини. Це обумовлює мету даної роботи: покращення техніко-економічних показників роботи мережі магазинів за рахунок оптимізації витрат на доставку товарів.

Для досягнення поставленої мети в спеціальному розділі пропонується вирішити дві математичні задачі.

1. Визначити послідовність обслуговування множини магазинів, вважаючи, що кожна з них обслуговується окремо і послідовно.
2. Створити програму для розбиття кластеру магазинів на рівні за дистанцією множини для оптимізації доставки

Для розв’язання задачі пропонується застосувати генетичний алгоритм.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Математичний опис задач TSP і VRP

Задача комівояжера (TSP) є однією з найвідоміших задач оптимізації маршрутів. Вона виникає, коли потрібно знайти найкоротший шлях, який проходить через всі дані міста і повертається до початкового міста. Ця задача має велике значення в багатьох галузях, таких як логістика, транспорт, маркетинг, телекомунікації та інші.

Математично задачу комівояжера можна сформулювати наступним чином:

1. Множина міст: Нехай ми маємо n міст, які позначені як V = {1, 2, 3, ..., n}.
2. Матриця відстаней: Задано матрицю відстаней d, де d[i][j] представляє відстань між містом i та містом j. Ця матриця може бути симетричною (d[i][j] = d[j][i]) або асиметричною, залежно від контексту задачі.
3. Цільова функція: Нашою ціллю є знайти такий гамільтонів цикл (цикл, що проходить через кожне місто рівно один раз), щоб сумарна відстань була мінімальною.
4. Обмеження: У кожному циклі потрібно відвідати кожне місто рівно один раз, і повернутися до початкового міста.

Метою задачі комівояжера є знайти перестановку міст, яка мінімізує сумарну відстань, проходячи через всі міста і повертаючись до початкового міста.

Задача маршрутизації транспорту (VRP) — це задача оптимізації, яка виникає при необхідності ефективного розподілу товарів або послуг з одного або декількох центральних складів до набору клієнтів. У цій задачі потрібно знайти оптимальні маршрути для транспортних засобів з метою мінімізації загальних витрат, таких як витрати на транспорт і час доставки.

Математично задачу маршрутизації транспорту можна сформулювати наступним чином:

1. Множина клієнтів: Нехай ми маємо n клієнтів, які позначені як V = {1, 2, 3, ..., n}.

2. Множина транспортних засобів: Нехай у нас є m транспортних засобів, позначених як K = {1, 2, 3, ..., m}.

3. Матриця відстаней: Задано матрицю відстаней d, де d[i][j] представляє відстань між клієнтом i та клієнтом j.

4. Вимоги клієнтів: Кожен клієнт має свої вимоги, такі як обсяг товарів, час доставки, обмеження на кількість візитів тощо.

5. Вартість: Вартість транспортування товарів або надання послуг для кожного маршруту залежить від відстані, часу та інших факторів.

Ціль задачі маршрутизації транспорту полягає в тому, щоб знайти оптимальний розподіл клієнтів між транспортними засобами і оптимальні маршрути для кожного засобу, задовольняючи вимоги клієнтів і мінімізуючи загальну вартість або час маршрутів.

Однією з головних відмінностей між завданнями TSP і VRP є підхід до оптимізації. TSP зосереджується на пошуку оптимального маршруту, який відвідує всі дані міста один раз і повертається до початкового міста, мінімізуючи загальну тривалість подорожі. У той час як у VRP основна увага приділяється оптимальному розподілу товарів між транспортними засобами та плануванню оптимальних маршрутів для кожного транспортного засобу з урахуванням різних обмежень.

Ще однією відмінністю є характер застосування завдань. TSP часто використовується для моделювання завдань планування маршруту та оптимізації логістичних процесів, таких як доставка вантажів або планування подорожей. Головна мета - знайти найкоротший по дорозі маршрут, який відвідає всі дані міста.

VRP, з іншого боку, знаходить своє застосування в оптимізації логістичних процесів, пов'язаних з розподілом товарів між різними клієнтами і визначенням оптимальних маршрутів для кожного транспортного засобу. Основна мета - ефективно розподіляти клієнтів між транспортними засобами і оптимально планувати маршрути з урахуванням різних обмежень, таких як місткість транспортного засобу і обмеження на кількість відвідувань клієнтів.

Таким чином, основна відмінність TSP від VRP завдань полягає в їх цілях і підходах до оптимізації. TSP фокусується на пошуку найкоротшого маршруту, який відвідує всі міста, тоді як VRP зосереджується на оптимальному розподілі товарів між транспортними засобами та плануванні оптимальних маршрутів для кожного транспортного засобу.

2.2 Використання API Google Distance Matrix

У цьому розділі показано, як використовувати API Google Distance Matrix для створення матриці відстаней для будь-якого набору місць, визначених адресами, широтами та довготами. API можна використовувати для розрахунку матриці відстаней для багатьох типів завдань маршрутизації.

Щоб використовувати API, вам знадобиться ключ API, який можна отримати з платформи Google.

Як приклад розглянемо програму на Python, яка створює матрицю відстаней для набору з 16 локацій у місті Дніпро, Україна. Матриця відстаней являє собою матрицю 16 x 16, запис якої i, j - відстань між розташуваннями i і j. Ось адреси місць.

1. вулиця Маршала Малиновського, 14А, Дніпро, Дніпропетровська область

2. вулиця Маршала Малиновського, 2, Дніпро, Дніпропетровська область
3. проспект Слобожанський, 1, Дніпро, Дніпропетровська область
4. бульвар Європейський, 4, Дніпро, Дніпропетровська область
5. вулиця Глінки, 2, Дніпро, Дніпропетровська область
6. вулиця Короленка, 3, Дніпро, Дніпропетровська область
7. проспект Пушкіна, 25/27, Дніпро, Дніпропетровська область
8. вулиця Шмідта, 2, Дніпро, Дніпропетровська область
9. вулиця Пастера, 6А, Дніпро, Дніпропетровська область
10. площа Вокзальна, буд. 2, Дніпро, Дніпропетровська область
11. вулиця Курчатова, 4, Дніпро, Дніпропетровська область
12. вулиця Робоча, 77, Дніпро, Дніпропетровська область
13. вулиця Титова, 13, Дніпро, Дніпропетровська область
14. вулиця Марії Кюрі, 5, Дніпро, Дніпропетровська область, 49010
15. пр-т.Гагаріна, буд. 35д, Дніпро, Дніпропетровська область, 49000
16. вулиця Набережна Перемоги 42Ц, Дніпро ,Дніпропетровська область, 49027

```
data['addresses'] = ['14A+Marshala+Malinovsky+St+ Dnipro', # depot
                     '2+Marshala+Malinovsky+St+Dnipro',
                     '1+Slobozhansky+Ave+Dnipro',
                     '4+European+Bou+Dnipro',
                     '2+Hlinka+St+Dnipro',
                     '3+Korolenko+St+Dnipro',
                     '25/27+Pushkin+Ave+Dnipro',
                     '2+Schmidt+St+Dnipro',
                     '6A+Pastera+St+Dnipro',
                     '2+Station+Squ+Dnipro',
                     '77+Working+St+Dnipro',
                     '13+Titova+St+Dnipro',
                     '4+Kurchatov+St+Dnipro',
                     '5+Maria+Curie+St+Dnipro',
                     '35+Gagarin+Ave+Dnipro',
                     '42+Victory+embankment+St+Dnipro'
                    ]
```

Рис 2.1 Код з блоком адрес

Запити API

API матриці відстаней - це довгий рядок, який містить наступне:

1. API адреса: https://maps.googleapis.com/maps/api/distancematrix/json? Кінець запиту, json, запитує відповідь у форматі JSON.

2. Пропозиції запитів: у цьому прикладі в полі units=imperial вказується англійська мова відповіді.

3. Адреси відправлення: початкові пункти подорожі. Наприклад, &origins=14A+Marshala+Malinovsky+Dnipro Пробіли в адресі замінюються символом + . Кілька адрес розділяються кнопкою | .

4. Адреса назначения: конечные пункты путешествия. Например &destinations=42+Victory+embankment+St+Dnipro

5. Ключ API: Облікові дані для запиту у формі &key=YOUR_API_KEY.

Відповідь міститиме відстань (у милях та метрах) та тривалість подорожі (у хвилинах та секундах) між двома адресами.

Для обчислення матриці відстаней ми хотіли б відправити один запит, що містить всі 16 адрес відправника та отримувача. Однак ми не можемо, оскільки для цього потрібно 16x16 = 256 пар відправник-призначення, тоді як API обмежений 100 такими парами на запит. Тому нам потрібно зробити кілька запитів.

Оскільки кожен рядок матриці містить 16 елементів, ми можемо обчислити максимум шість рядків на запит (потрібно 6x16=96 пар). Ми можемо обчислити всю матрицю в трьох запитах, які повертають 6 рядків, 6 рядків і 4 рядки.

Наступний код обчислює матрицю відстаней наступним чином:

1. Розділіть 16 адрес на дві групи по шість адрес і одну групу по чотири адреси.
2. Для кожної групи створіть і надішліть запит на вихідні адреси в групі та всі адреси призначення. См. Створіть і надішліть запит.
3. Використовуйте відповідь, щоб побудувати відповідні рядки матриці та об'єднати рядки

Наступна функція створює і відправляє запит для заданого набору адрес відправника і одержувача.

```
def create_distance_matrix(data):
  addresses = data["addresses"]
  API_key = data["API_key"]
  # Distance Matrix API only accepts 100 elements per request, so get rows in multiple requests.
  max_elements = 100
  num_addresses = len(addresses) # 16 in this example.
  # Maximum number of rows that can be computed per request (6 in this example).
  max_rows = max_elements // num_addresses
  # num_addresses = q * max_rows + r (q = 2 and r = 4 in this example).
  q, r = divmod(num_addresses, max_rows)
  dest_addresses = addresses
  distance_matrix = []
  # Send q requests, returning max_rows rows per request.
  for i in range(q):
    origin_addresses = addresses[i * max_rows: (i + 1) * max_rows]
    response = send_request(origin_addresses, dest_addresses, API_key)
    distance_matrix += build_distance_matrix(response)

  # Get the remaining remaining r rows, if necessary.
  if r > 0:
    origin_addresses = addresses[q * max_rows: q * max_rows + r]
    response = send_request(origin_addresses, dest_addresses, API_key)
    distance_matrix += build_distance_matrix(response)
  return distance_matrix
```

Рис 2.2 . Функція створення матриці відстаней

build_address_string об’єднує адреси, розділені чертою | .

Код, що залишився у функції збирає описані вище частини запиту і відправляє запит. Відповідь рядка = json. Loads(jsonResult) перетворює необроблений результат на об’єкт Python.

```
def send_request(origin_addresses, dest_addresses, API_key):
    """ Build and send request for the given origin and destination addresses."""
    def build_address_str(addresses):
        # Build a pipe-separated string of addresses
        address_str = ''
        for i in range(len(addresses) - 1):
            address_str += addresses[i] + '|'
        address_str += addresses[-1]
        return address_str

    request = 'https://maps.googleapis.com/maps/api/distancematrix/json?units=imperial'
    origin_address_str = build_address_str(origin_addresses)
    dest_address_str = build_address_str(dest_addresses)
    request = request + '&origins=' + origin_address_str + '&destinations=' + \
                        dest_address_str + '&key=' + API_key
    jsonResult = urllib.urlopen(request).read()
    response = json.loads(jsonResult)
    return response
```

Рис 2.3 Збір параметрів і відправка запитів

Наступна функція будує рядки матриці відстаней, використовуючи відповідь, повернуту функцією send_request.

```
def build_distance_matrix(response):
    distance_matrix = []
    for row in response['rows']:
        row_list = [row['elements'][j]['distance']['value'] for j in range(len(row['elements']))]
        distance_matrix.append(row_list)
    return distance_matrix
```

Рис 2.4 Функція будує рядки матриці

Рядок row_list = [row['elements'][j]['distance']['value'] для j in range(len(row['elements']))] витягує відстані між місцями для рядка відповіді. Ви можете порівняти це з частиною відповіді (перетворена за допомогою json.loads) для того самого джерела та споживача, як показано нижче.

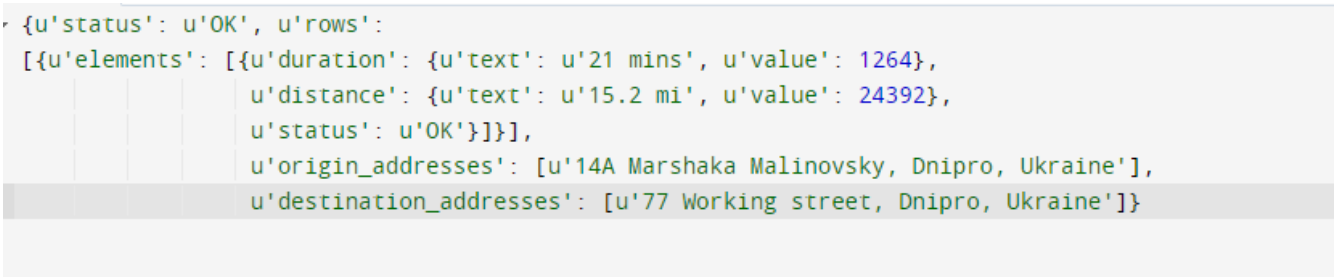

Рис 2.5

Якщо ви хочете створити матрицю часу, що містить час у дорозі між місцями, замініть " distance" на " duration" у функції build_distance_matrix.

2.3 Особливості побудови початкових рішень і локального пошуку

Перша стратегія рішення - це метод, який вирішувач використовує для пошуку початкового рішення. У таблиці нижче наведено параметри first_solution_strategy.

Таблиця 2.1

Варіанти функції першого рішення

Варіант	Опис
AUTOMATIC	Дозволяє програмі визначити, яку стратегію використовувати відповідно до моделі, що вирішується автоматично.
PATH_CHEAPEST_ARC	Починаючи з «стартового» вузла маршруту, з'єднайте його з вузлом, який створює найдешевший відрізок маршруту, потім продовжте маршрут, повторивши останній вузол, доданий до маршруту.
PATH_MOST_CONSTRAINED_ARC	Подібно до PATH_CHEAPEST_ARC, але дуги оцінюються за допомогою селектора на основі порівняння, який спочатку віддає перевагу найбільш обмеженій дузі.
EVALUATOR_STRATEGY	Аналогічно PATH_CHEAPEST_ARC, за винятком того, що вартість дуг оцінюється за допомогою функції,

	переданої SetFirstSolutionEvaluator()
SAVINGS	Алгоритм заощаджень Кларка і Райта
SWEEP	Алгоритм розгортки Рена і Холлідей
CHRISTOFIDES	Алгоритм Хрістофіда. Працює з типовими моделями маршрутизації транспортних засобів, продовжуючи маршрут до тих пір, поки в нього не будуть вставлені вузли.
ALL_UNPERFORMED	Робить всі вузли неактивними. Знаходить рішення, тільки якщо вузли необов'язкові
BEST_INSERTION	Ітеративно створює рішення, вставляючи найдешевший вузол в його найдешевше положення; Вартість вставки базується на глобальній витратній функції моделі маршрутизації.
PARALLEL_CHEAPEST_INSERTION	Ітеративно створює рішення, вставляючи найдешевший вузол у найдешевше положення; вартість вставки базується на функції вартості дуги. Швидше, ніж BEST_INSERTION.
LOCAL_CHEAPEST_INSERTION	Ітеративно створює рішення, вставляючи кожен вузол у найдешевшу позицію; вартість вставки базується на функції вартості дуги. Відрізняється від вузла PARALLEL_CHEAPEST_INSERTION обраного для вставки; Тут вузли розглядаються в тому порядку, в якому вони створюються. Він працює швидше, ніж PARALLEL_CHEAPEST_INSERTION.
GLOBAL_CHEAPEST_ARC	Ітеративно з'єднуються два вузли, які створюють найдешевший сегмент маршруту.
LOCAL_CHEAPEST_ARC	Вибирає перший вузол з непов'язаним наступником і з'єднує його з вузлом, який створює найдешевший сегмент маршруту.
FIRST_UNBOUND_MIN_VALUE	Вибирає перший вузол з непов'язаним наступником і підключає його до першого доступного вузла. Це еквівалентно стратегії CHOOSE_FIRST_UNBOUND в поєднанні з ASSIGN_MIN_VALUE

Найбільш перспективними і швидкими методами є PATH_CHEAPEST_ARC і Христофід, але для первинної перевірки найкраще використовувати автоматичний метод, хоча він і не гарантує найбільшої швидкості.

Тепер давайте розглянемо варіанти локального пошуку, показані в таблиці нижче

Таблиця 2.2

Варіанти функції локального пошуку

Варіант	Опис
AUTOMATIC	Дозволяє програмі автоматично вибирати параметр локального пошуку.
GREEDY_DESCENT	Приймає поліпшення (зниження витрат) локальних пошукових сусідів до тих пір, поки не буде досягнутий локальний мінімум.
GUIDED_LOCAL_SEARCH	Використовує керований локальний пошук, щоб уникнути локальних мінімумів. Це, як правило, найефективніша метаевристика для маршрутизації транспортних засобів.
SIMULATED_ANNEALING	Використовує імітацію отжига, щоб уникнути локальних мінімумів
TABU_SEARCH	Використовує табу- пошук, щоб обійти місцеві мінімуми
GENERIC_TABU_SEARCH	Використовує табу-пошук за цільовим значенням рішення, щоб уникнути локальних мінімумів.

З усіх методів локального пошуку найбільш перспективними і швидкими є метод контрольованого локального пошуку, метод табу і метод відпалу. Нижче ми розглянемо кожен з цих методів.

Метод керованого локального пошуку - евристичний алгоритм, який використовується для вирішення комбінаторних задач оптимізації, включаючи задачу комівояжера (TSP) та інші. Він поєднує в собі переваги локального пошуку з використанням інформації про проблему і попередніх рішень для управління пошуком і досягнення більш оптимальних рішень.

Принцип роботи керованого локального методу пошуку складається з декількох етапів. Давайте розглянемо його докладніше:

1. Ініціалізація: початкове рішення створюється випадковим чином або вибирається з попередніх рішень.

2. Локальний пошук: застосовує локальний пошук для покращення поточного рішення. Локальним пошуком може бути базовий алгоритм, наприклад 2-OPT або 3-OPT для TSP, або будь-який інший алгоритм, здатний поліпшити поточне рішення.

3. Руйнування: Руйнування поточного розчину здійснюється для створення декількох «зруйнованих» розчинів. Робиться це шляхом видалення або зміни деяких частин розчину.

4. Відновлення: Для кожного зруйнованого рішення відновлення виконується шляхом застосування локального пошуку або інших операторів для отримання нового вдосконаленого рішення.

5. Інтенсифікація: інтенсифікація спрямована на поліпшення поточного рішення шляхом посилення його переважно хороших показників. Сюди можна віднести зміцнення хороших ребер жорсткості або перестановку хороших елементів.

6. Диверсифікація: диверсифікація зосереджена на диверсифікації рішень та вивченні нових областей пошуку. Цього можна досягти, змінюючи параметри алгоритму або вносячи випадкові зміни в рішення.

7. Перевірка критерію зупинки: перевіряє, чи виконано критерій зупинки, наприклад обмеження часу або кількість ітерацій. Якщо критерій зупинки не досягнутий, алгоритм повертається до кроку 2 і продовжує ітеративно покращувати рішення.

Метод керованого локального пошуку поєднує в собі інтенсивне і широке дослідження простору рішень, що дозволяє досягти більш оптимальних результатів. Використання інформації про попередні рішення і знань про проблему дозволяє більш ефективно управляти процесом пошуку і скорочувати витрати часу на пошук оптимального рішення.

В результаті керований метод локального пошуку являє собою евристичний алгоритм оптимізації, який поєднує локальний пошук з використанням інформації про проблему і попередніх рішеннях для управління пошуком і досягнення більш оптимальних результатів. Він широко застосовується для вирішення комбінаторних задач оптимізації, в тому числі і задачі комівояжера, і показує хороші результати при вирішенні складних завдань.

Tabu Search також є евристичним алгоритмом оптимізації, який використовується для вирішення комбінаторних задач. Він заснований на ідеї пошуку в просторі рішень за допомогою списку заборон (список табу), щоб уникнути повторного відвідування вже пройдених станів і руху до кращого рішення.

Принцип пошуку табу включає наступні етапи:

1. Ініціалізація: початкове рішення створюється випадковим чином або вибирається з попередніх рішень. Список табу та інші допоміжні структури даних також ініціалізовані.

2. Генерація сусідніх рішень: Сусідні рішення генеруються з поточного рішення, які отримуються шляхом застосування певних операторів, таких як обмін двома містами в задачі TSP.

3. Оцінка рішення: Кожне сусіднє згенероване рішення обчислюється на основі цільової функції, такої як сума відстаней у задачі TSP. Мета - вибрати найкраще сусіднє рішення.

4. Застосування правил табу: перевіряється, чи знаходиться сусіднє рішення в списку табу, який містить інформацію про заборонені кроки. Якщо сусіднє рішення знаходиться в списку табу, застосовуються правила табу, що дозволяють все-таки вибрати це рішення, щоб рухатися до кращого результату.

5. Оновити список табу: Після того, як ви виберете сусіднє рішення для застосування, список табу оновлюється, щоб включити інформацію про вибране переміщення та видалити застарілі елементи.

6. Перевірка критерію зупинки: перевіряє, чи виконано критерій зупинки, наприклад обмеження часу або кількість ітерацій. Якщо критерій зупинки не досягнутий, алгоритм повертається до кроку 2 і продовжує ітеративно покращувати рішення.

Пошук табу дозволяє уникнути застрягання в місцевих оптимумах, забороняючи повторні відвідування минулих штатів, а також застосовуючи

правила табу для переходу до кращого рішення, навіть якщо це порушує деякі обмеження. Це дозволяє алгоритму ефективніше досліджувати простір рішень і знаходити кращі результати.

Табуйований пошук є потужним і ефективним методом оптимізації і широко використовується для вирішення комбінаторних задач оптимізації. Він дозволяє досягати більш оптимальних рішень і долати локальні оптимуми, що робить його корисним інструментом для різних областей застосування, які вимагають оптимального планування і маршрутизації.

Метод модельованого відпалу - це стохастичний алгоритм оптимізації, який використовується для вирішення комбінаторних задач. Він заснований на аналогії з фізичним процесом відпалу металу, де система охолоджується поступово для досягнення мінімальної енергії.

Принцип роботи методу відпалу наступний:

1. Ініціалізація: вихідний розв'язок вибирається випадковим чином або може бути виведений з попередніх рішень. Також визначаються параметри алгоритму, включаючи температуру і швидкість охолодження.

2. Генерація сусідніх рішень: сусідні рішення генеруються з поточного рішення за допомогою певних операторів, таких як обмін двох міст у задачі комівояжера.

3. Оцінка рішення: Кожне сусіднє згенероване рішення оцінюється на основі цільової функції, такої як сума відстаней у задачі комівояжера. Мета - вибрати найкраще сусіднє рішення.

4. Прийняття або відхилення рішення: Рішення може бути прийнято, навіть якщо воно гірше поточного рішення. Імовірність прийняття такого рішення залежить від поточної температури і різниці цільових функцій.

5. Оновлення параметра: температура знижується відповідно до встановленої функції охолодження. Це дозволяє алгоритму поступово рухатися до кращих рішень і зменшувати ймовірність прийняття гірших рішень при зниженні

температури.

6. Перевірка критерію зупинки: перевіряє, чи виконано критерій зупинки, наприклад обмеження часу або кількість ітерацій. Якщо критерій зупинки не досягнутий, алгоритм повертається до кроку 2 і продовжує ітеративно покращувати рішення.

Метод відпалу дозволяє алгоритму змоделювати фізичний процес відпалу, при якому система поступово занурюється в мінімальну енергію. Це дозволяє алгоритму ефективно досліджувати простір рішень і знаходити глобальні оптимуми, долаючи локальні оптимуми.

Основною перевагою методу відпалу є його здатність обходити локальні оптимуми і досліджувати різні ділянки простору розчину. Однак для досягнення оптимальних результатів потрібно правильно підібрати параметри алгоритму, включаючи початкову температуру, швидкість охолодження і функцію охолодження.

2.4 Побудова програми для пошуку оптимальних маршрутів.

Проаналізувавши алгоритм побудови програм за допомогою сервісу Google OR-tools і методи, які він може використовувати, можна приступати до розробки програми на його основі.

Для цього проекту буде створено дві програми:

1. Підібрати оптимальний маршрут доставки вантажів за допомогою 1 автомобіля

2. Знайти оптимальний маршрут доставки можна, розділивши магазини на кластери і використовуючи кілька машин.

Для початку розглянемо побудову програми для пошуку оптимального маршруту доставки на базі сервісу Google OR-Tools.

```
import math
import time
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp
```

Рис 2.6 Бібліотеки програми

На рисунку 2.6 зображені бібліотеки, необхідні для запуску цієї програми.

```
def create_data_model():
    """Stores the data for the problem."""
    data = {}
    # Locations in block units
    data['locations'] = [
       (24748.3333,50840.0000),
(24758.8889,51211.9444),
(24827.2222,51394.7222),
(24904.4444,51175.0000),
(24996.1111,51548.8889),
(25010.0000,51039.4444),
(25030.8333,51275.2778),
(25067.7778,51077.5000),
(25100.0000,51516.6667),
(25103.3333,51521.6667),
(25121.9444,51218.3333),
```

Рис 2.7 Частина масиву координат

На рисунку 2.7 зображена частина координатного масиву у вигляді точок по широті і довготі на карті, саме по ним і буде орієнтуватися програма. Повний масив буде у додатоку разом з повним кодом програми.

```
def compute_euclidean_distance_matrix(locations):
    """Creates callback to return distance between points."""
    distances = {}
    for from_counter, from_node in enumerate(locations):
        distances[from_counter] = {}
        for to_counter, to_node in enumerate(locations):
            if from_counter == to_counter:
                distances[from_counter][to_counter] = 0
            else:
                # Euclidean distance
                distances[from_counter][to_counter] = (int(
                    100*math.hypot((from_node[0] - to_node[0]),
                                   (from_node[1] - to_node[1]))))
    return distances
```

Рис 2.8 Обчислення відстані

На рисунку 2.8 представлена функція, що відповідає за обчислення відстані між точками.

```
def print_solution(manager, routing, solution):
    """Prints solution on console."""
    print('Objective: {}'.format(solution.ObjectiveValue()))
    index = routing.Start(0)
    plan_output = 'Route:\n'
    route_distance = 0
    while not routing.IsEnd(index):
        plan_output += ' {} ->'.format(manager.IndexToNode(index))
        previous_index = index
        index = solution.Value(routing.NextVar(index))
        route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
    plan_output += ' {}\n'.format(manager.IndexToNode(index))
    print(plan_output)
    plan_output += 'Objective: {}m\n'.format(route_distance)
```

Рис 2.9 Побудування графу

Ця функція будує граф, який з'єднує точки на карті між собою і наочно показує загальний логістичний маршрут

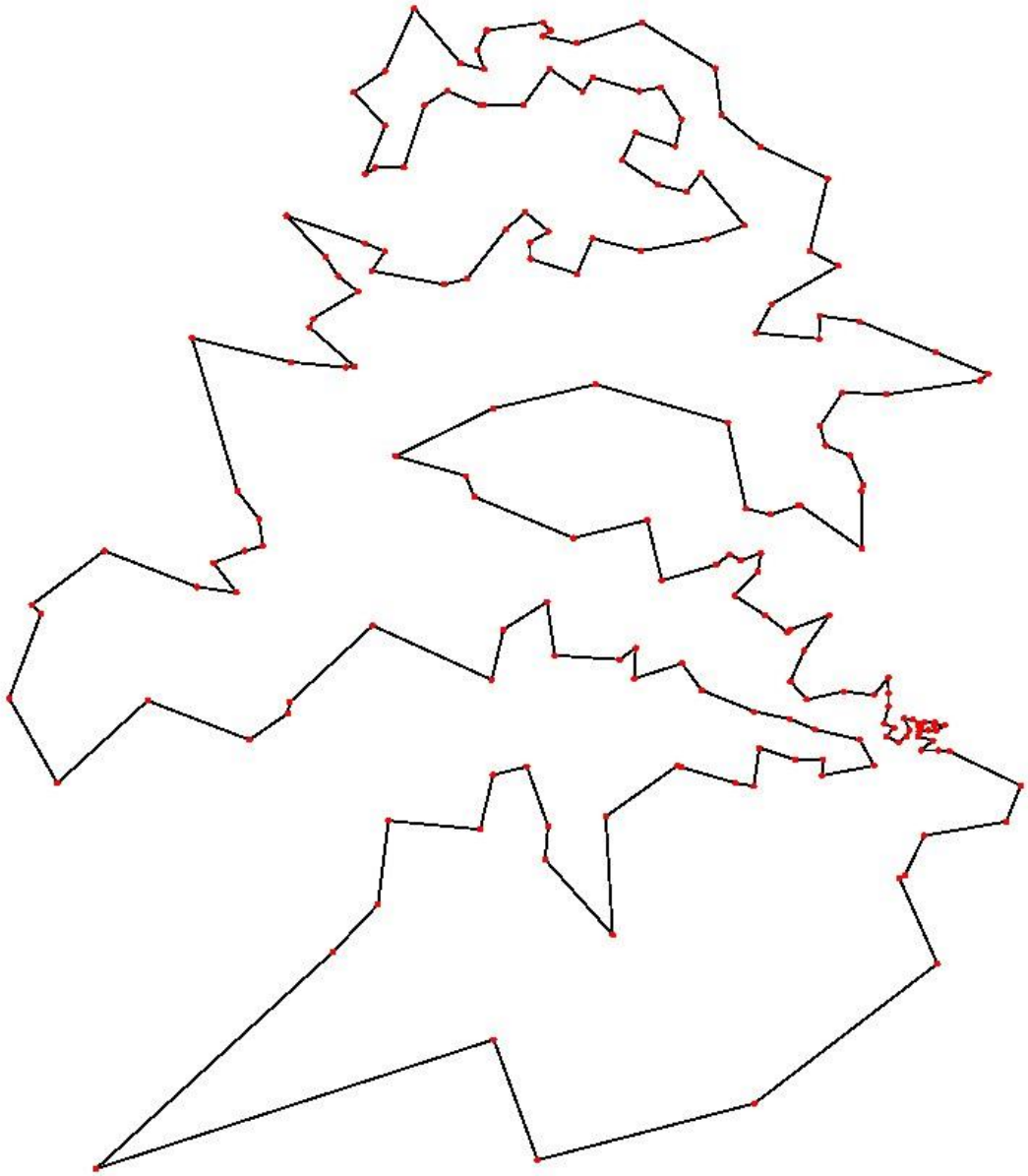

Рис 2.10 Граф

На рисунку 2.10 показаний загальний логістичний маршрут, червоними крапками позначені точки доставки, а лінії, які їх з'єднують, не довільні, а дороги.

```
def main():
    """Entry point of the program."""
    # Instantiate the data problem.
    data = create_data_model()
    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data['locations']),
                                           data['num_vehicles'], data['depot'])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)
    distance_matrix = compute_euclidean_distance_matrix(data['locations'])

    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return distance_matrix[from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic = (
        routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.seconds = 20
    search_parameters.log_search = False
```

Рис 2.11 Основна частина програми

Основна частина програми відповідає за розрахунок найкращого способу доставки товару, враховуючи, що обслуговувати його буде тільки 1 машина.

Знаючи принципи побудови завдання на розподіл точок на 1 машину, ми можемо побудувати програму, яка допоможе нам оптимізувати доставку,

використовуючи кілька машин на маршруті і розбиваючи її так, щоб вони проїхали рівну відстань.

```
data['distance_matrix'] = compute_euclidean_distance_matrix(data['locations'])
data['num_vehicles'] = 3
data['depot'] = 0
return data
```

Рис 2.12 Розподіл на кількість автомобілів

```
# Setting first solution heuristic.
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
search_parameters.local_search_metaheuristic = (
    routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
search_parameters.time_limit.seconds = 30
search_parameters.log_search = False
```

Рис 2.13 Виклик алгоритму пошуку

На рисунках 2.12 та 2.13 представлені ключові відмінності цих програм. Як можна побачити на рисунку 2.14 ідеальнім розподілом буде розподіл на 3 автомобілі, що ми і зробили, зазначивши це у третій строчці на рисунку 2.12. А вже на рисунку 2.13, мі можемо побачити, що для цього використовується алгоритм PATH_CHEAPEST_ARC, як найефективніший.

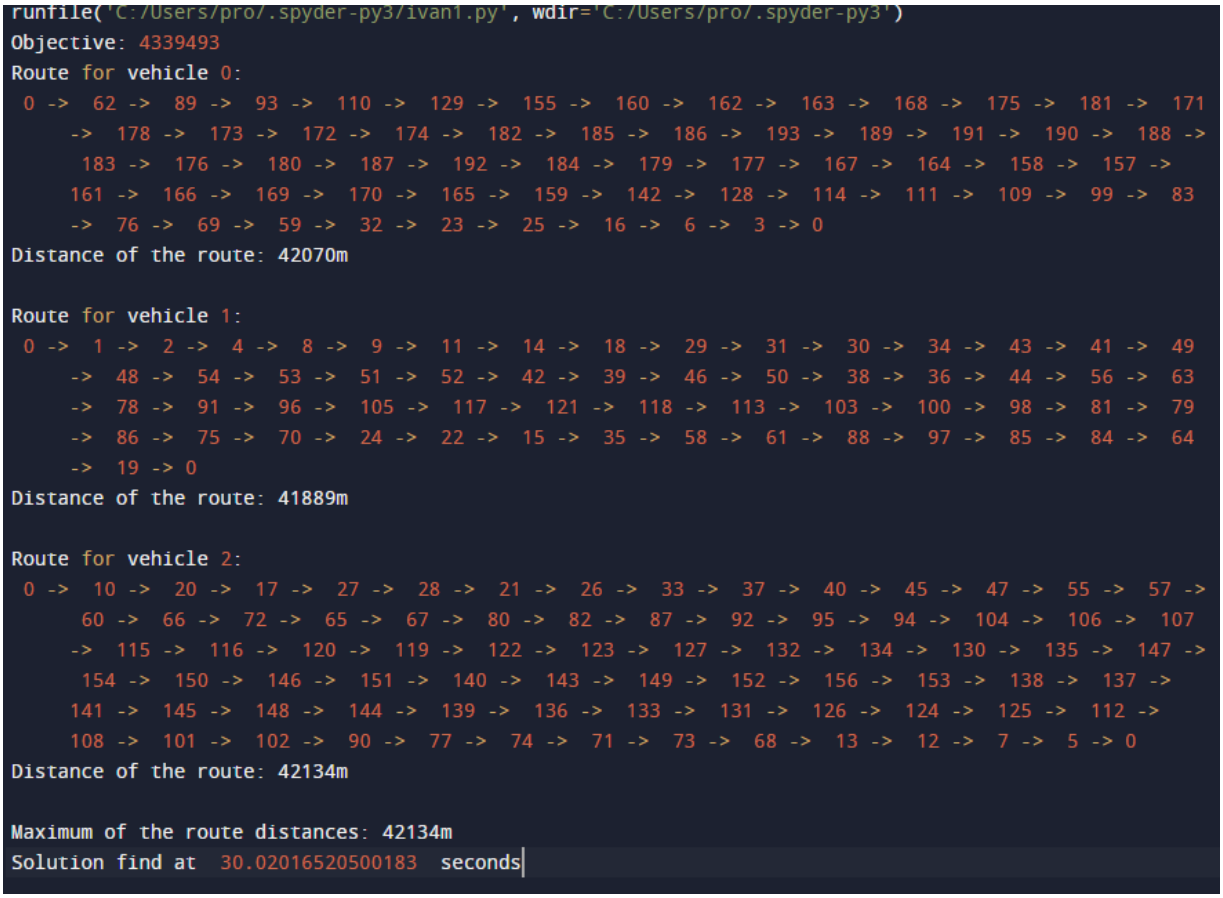

Рис 2.14 Результат роботи програми

На рисунку 2.14 представлена відповідь на запит до цієї програми и як ми можемо побачити, розбиття на 3 маршрути є найефективнішим і машини проходять майже однакову дистанцію у приблизно 42 кілометри.

2.5 Додаткові варіанти розширення можливостей ПЗ

Перераховані вище особливості програмного забезпечення не єдині, якщо завдання вимагає дозволу більш точних умов його дозволу, код може бути доповнений деякими елементами, які ми розберемо нижче.

Почнемо аналіз можливих додаткових умов з умови ліміту часу. Якщо в одній з точок маршруту нам терміново потрібен товар, але при цьому нам все одно

потрібно відвідати всі точки на маршруті, ми можемо скористатися цією умовою. Ви навіть можете встановити кожному пункту період часу, в який повинна здійснюватися доставка.

Такі обмеження можна враховувати, встановлюючи lower/upper атрибути на кожному вузлі, а також активувавши атрибут time_windows на True. Крім того, ви можете обліковувати час обслуговування кожного вузла, встановивши атрибут service_time.

```
>>> G.nodes[1]["lower"] = 0
>>> G.nodes[1]["upper"] = 10
>>> G.nodes[2]["lower"] = 5
>>> G.nodes[2]["upper"] = 9
>>> G.nodes[1]["service_time"] = 1
>>> G.nodes[2]["service_time"] = 2
>>> prob.time_windows = True
>>> prob.solve()
```

Рис 2.15 Умови обмежень за часом доставки

В якості наступної можливої умови можна розглядати обмеження на максимальний час у дорозі і на мінімальну кількість точок, які повинен відвідати автомобіль. Ця умова дуже хороша, якщо у нас обмежений парк транспортних засобів, що ми не можемо об'їхати всі точки за один день. З огляду на норми закону, згідно з якими-водій кур'єр не може перебувати за кермом довше певної кількості часу, ця умова дозволяє істотно оптимізувати маршрут.

```
>>> G.edges["Source",1]["time"] = 5
>>> G.edges["Source",2]["time"] = 4
>>> G.edges[1,2]["time"] = 2
>>> G.edges[1,"Sink"]["time"] = 6
>>> G.edges[2,"Sink"]["time"] = 1
>>> prob.duration = 9
>>> prob.solve()
>>> prob.num_stops = 1
```

Рис 2.15 Умови обмеження часу в дорозі

В якості останньої умови можна розглядати умову пропускної здатності.

Маючи справу з великоваговими або великогабаритними вантажами, ми повинні розуміти, що потрібно регулювати, скільки вантажу може перевезти один автомобіль і чим менше вантажу він може перевезти, тим більше автомобілів потрібно бути на маршруті. Ось для чого потрібна ця умова.

```
>>> from networkx import DiGraph
>>> from vrpy import VehicleRoutingProblem
>>> G = DiGraph()
>>> G.add_edge("Source", 1, cost=1)
>>> G.add_edge("Source", 2, cost=2)
>>> G.add_edge(1, "Sink", cost=0)
>>> G.add_edge(2, "Sink", cost=2)
>>> G.add_edge(1, 2, cost=1)
>>> G.nodes[1]["demand"] = 2
>>> G.nodes[2]["demand"] = 3
>>> prob = VehicleRoutingProblem(G, load_capacity=10)
>>> prob.solve()
```

Рис 2.16 Умови обмеження ваги

2.6 Висновки до розділу 2

Метою спеціального розділу було побудова ПЗ для оптимального і швидкого розрахунку логістичних маршрутів для компанії. За основу для побудови такого програмного забезпечення було узято досконалий та сучасний застосунок Google OR-Tools.

Як ми можемо побачити, додаток Google OR-Tools є дуже потужним застосунком для оптимізації задач TSP та VRP, завдяки багатьом методом для пошуку початкового рішення, а також для методів локального пошуку. Він дає дуже велику гнучкість як для розробників програмного забезпечення на його основі, так і для користувачів цього ПО.

Проаналізувавши більшу частину методів які використовую застосунок можна сказати, що для пошуку стратегії першого рішення найкраще себе показують методи - AUTOMATIC та CHRISTOFIDES. Перший через те, що перебирає усі варіанти та вибирає серед них найліпший, але за це доведеться платити швидкістю, а другий через свою досконаліть, але може будувати і не найоптимальніший маршрут, але навіть у такому разі похибка не буде сягати навіть кількох процентів.

Серед методів локального пошуку найліпше себе показує SIMULATED_ANNEALING та TABU_SEARCH, серед яких перевагу у даній роботі було віддано все ж першому варіанту, через цікавий і більш досконалий алгоритм.

ВИСНОВКИ

CPG компанії займають сильне, важливе і невід'ємне місце в світовій економіці, але, тим не менш, більшість цих компаній мають ряд подібних проблем, особливо в сфері логістики. Метою даної кваліфікаційної роботи був аналіз діяльності підприємства и покращення економічних показників роботи підприємства за рахунок мінімізації логістичних витрат.

Проаналізувавши роботу підприємства і його економічні показники, ми прийшли до висновку, що, як і для багатьох CPG-компаній, існує ряд проблем, які призводять до витрат, які можна скоротити і тим самим збільшити чистий прибуток підприємства.

Основною проблемою, як ми і очікували, є проблема логістики, через її надмірне перевантаження вона не могла працювати максимально ефективно. Проаналізувавши різні методи вирішення цієї проблеми, ми відкинули багато з них з різних причин. Деякі з них не задовольняли нас в плані швидкості їх роботи, що дуже важливо при наявності великої кількості точок, а деякі через занадто негнучкої системи налаштувань. Зрештою, наш вибір припав на доповнення Google OR-Tools. Розібравшись з його роботою, ми поставили перед собою наступні завдання:

1. Визначити послідовність обслуговування множини магазинів, вважаючи, що кожна з них обслуговується окремо і послідовно.

2. Створити програму для розбиття кластеру магазинів на рівні за дистанцією множини для оптимізації доставки

Для вирішення обох завдань ми склали програми на базі сервісу Google OR-Tools. Що є потужною і змінною основою для вирішення таких завдань. Вона має як свої методи базового вирішення даного завдання, так і її можна доповнити умовами, щоб зробити програму більш точно настроюється.

Можна резюмувати, що використання сервісу Google OR-Tools спільно з

доповненнями дозволяє істотно знизити витрати на доставку товару. Скоротити кількість автомобілів, водіїв і персоналу, який їх обслуговує. А також знизити витрати на логістичний відділ компанії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M. Fischetti, J.J. Salazar-Gonzalez, and P. Toth. A Branch-and-Cut algorithm for the symmetric generalized traveling salesman problem. Operations Research 45 (3) (1997), 378—394.
2. D. Ben-Arieh, G. Gutin, M. Penn, A. Yeo, and A. Zverovitch. Transformations of generalized ATSP into ATSP, Operations Research Letters 31 (2003), 357—365.
3. 6. Arash Behzad, Mohammad Modarres (2002). A New Efficient Transformation of Generalized Traveling Salesman Problem into Traveling Salesman Problem
4. L.V. Snyder and M.S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. European Journal of Operational Research 174 (2006), 38−53.
5. J. Silberholz and B. Golden. The Generalized Traveling Salesman Problem: a new Genetic Algorithm approach. Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies, 2007, 165−181.
6. G. Gutin and D. Karapetyan. Gregory Gutin and Daniel Karapetyan. A Memetic Algorithm for the Generalized Traveling Salesman Problem. Natural Computing 9(1), pages 47-60, Springer 2010. (недоступная ссылка)
7. Lau, T.L. & Tsang, E.P.K., Solving the processor configuration problem with a mutation-based genetic algorithm, International Journal on Artificial Intelligence Tools (IJAIT), World Scientific, Vol.6, No.4, December 1997, 567-585
8. Mills, P. & Tsang, E.P.K., Guided local search for solving SAT and weighted MAX-SAT problems, Journal of Automated Reasoning, Special Issue on Satisfiability Problems, Kluwer, Vol.24, 2000, 205-223
9. Tsang, E.P.K. & Voudouris, C., Fast local search and guided local search and their application to British Telecom's workforce scheduling problem, Operations Research Letters, Elsevier Science Publishers, Amsterdam, Vol.20, No.3, March

1997

10. Voudouris, C. & Tsang, E.P.K., Guided local search joins the elite in discrete optimisation, DIMACS Series in Discrete Mathematics and Theoretical Computer Science Volume 57, 2001, 29-39
11. Fred Glover (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". Computers and Operations Research. 13 (5): 533–549.
12. F. Glover, M. Laguna & R. Marti (2000). "Fundamentals of Scatter Search and Path Relinking". Control and Cybernetics. 29 (3): 653–684.
13. D. Gamboa, C. Rego & F. Glover (2005). "Data Structures and Ejection Chains for Solving Large Scale Traveling Salesman Problems". European Journal of Operational Research. 160 (1): 154–171
14. Moscato, Pablo (June 1993). "An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search". Annals of Operations Research. 41 (2): 85–121.
15. Granville, V.; Krivanek, M.; Rasson, J.-P. (1994). "Simulated annealing: A proof of convergence". IEEE Transactions on Pattern Analysis and Machine Intelligence. 16 (6): 652–656
16. Černý, V. (1985). "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm". Journal of Optimization Theory and Applications. 45: 41–51

ДОДАТКИ

ДОДАТОК А
Відомості матеріалів дипломної роботи

		Позначення	Найменування	Кількість	Примітка
1					
2			Документація		
3					
4		САУ.КР.23.10.ПЗ	Пояснювальна записка	77	Формат А4
5					
6		САУ.КР.23.10.ДМ	Демонстраційні матеріали	15	А4
7					
8		САУ.КР.23.10.КР	Копія роботи	1	(Диск CD-R)

					САУ.КР.23.10.ДА.ПЗ			
Зм	Лист	№ докум.	Підпис	Дата				
Розроб.		Молчанов			**Матеріали кваліфікаційної роботи**	Літ	Аркуш	Аркушів
К. розд.		Желдак						
Керівн.		Желдак				НТУ «ДП», 12; 124-19-1		
Н. контр.		Хом’як						
Зав. каф.		Желдак						

Додаток Б

Відгук
на кваліфікаційну роботу бакалавра
студента гр. 124-19-1
Молчанова Івана Володимировича

Тема кваліфікаційної роботи: «Системний аналіз та оптимізація логістичних процесів в умовах підприємства роздрібної торгівлі господарчими товарами».

Обсяг кваліфікаційної роботи 76 стор.

Мета кваліфікаційної роботи: покращення економічних показників роботи підприємства за рахунок мінімізації логістичних витрат.

Актуальність теми обумовлена необхідністю розв’язання оптимізаційних задач в логістичних процесах постачання продукції на об’єкті дослідження.

Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки предметом дослідження є теоретичні, методичні та прикладні аспекти побудови оптимальних маршрутів руху транспорту в умовах торговельного підприємства при наявності обмежень.

Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра. Оригінальність наукових рішень полягає у використанні сучасного програмного забезпечення мовою Python, яке за адресами магазинів вираховує маршрути, використовуючи хмарні можливості компанії Гугл.

Практичне значення результатів кваліфікаційної роботи полягає в тому, що запропонована розроблена система скорочує час на побудову оптимального маршруту та скорочує витрати компанії.

Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі.

У роботі відзначено такі **недоліки**:

1. В роботі не наведені математичні записи використаних алгоритмів та способів урахування обмежень, які використовує програмне забезпечення.
2. В роботі наведено вирішення лише однієї задачі з постійною кількістю магазинів, хоча щодня ситуація може бути різна.

Кваліфікаційна робота в цілому заслуговує оцінки: «добре» (80 балів).

З урахуванням висловлених зауважень автор заслуговує присвоєння освітньої кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи,
к.т.н., доц., завідувач кафедри САУ Желдак Т.А.

Додаток В

Рецензія

на кваліфікаційну роботу бакалавра

студента групи 124 – 19 – 1 спеціальності 124 Системний аналіз

Молчанова Івана Володимировича

Тема кваліфікаційної роботи: «Системний аналіз та оптимізація логістичних процесів в умовах підприємства роздрібної торгівлі господарчими товарами»

Обсяг кваліфікаційної роботи: 76 стор.

Виконані в кваліфікаційній роботі дослідження та отримані в підсумку результати відповідають завданню, та освітньо-професійній програмі спеціальності оскільки предметом дослідження є теоретичні, методичні та прикладні аспекти побудови оптимальних маршрутів руху транспорту в умовах торговельного підприємства при наявності обмежень.

Кваліфікаційна робота виконана на високому виконавському рівні, у відповідності до нормативно-методичної літератури. В роботі застосовані сучасні програмні розробки та передові методи побудови оптимальних обходів графів.

Позитивні сторони кваліфікаційної роботи: автор використовує сучасне програмне забезпечення від компанії Google, яке дозволяє будувати замкнені та незамкнені маршрути для довільних адрес на карті, вирішувати задачі комівояжера з обмеженнями і додатковими умовами

Основні недоліки кваліфікаційної роботи:

1. Рішення для маршрутів, названі «оптимальними», хоча ніде не перевіряється і наперед не відома їх оптимальність.
2. Автор використовує одну з відомих реалізацій відомих методів і каже, що це добре, не порівнюючи з іншими методами та реалізаціями.

Кваліфікаційна робота в цілому заслуговує оцінки: «добре»

З урахуванням висловлених зауважень автор заслуговує присвоєння освітньої кваліфікації «бакалавр з системного аналізу».

Рецензент, к.т.н., доцент _____ / Шедловський І.А..

Додаток Г

Повний код програми для пошуку оптимального шляху доставки:

```
import math
import time
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp


def create_data_model():
    """Stores the data for the problem."""
    data = {}
    # Locations in block units
    data['locations'] = [
       (24748.3333,50840.0000), (24758.8889,51211.9444), (24827.2222,51394.7222),
(24904.4444,51175.0000), (24996.1111,5 1548.8889), (25010.0000,51039.4444),
(25030.8333,51275.2778), (25067.7778,51077.5000), (25100.0000,51516.6667),
(25103.3333,51521.6667), (25121.9444,51218.3333), (25150.8333,51537.7778),
(25158.3333,51163.6111), (25162.2222,51220.8333), (25167.7778,51606.9444),
(25168.8889,51086.3889), (25173.8889,51269.4444), (25210.8333,51394.1667),
(25211.3889,51619.1667), (25214.1667,50807.2222), (25214.4444,51378.8889),
(25223.3333,51451.6667), (25224.1667,51174.4444), (25233.3333,51333.3333),
(25234.1667,51203.0556), (25235.5556,51330.0000), (25235.5556,51495.5556),
(25242.7778,51428.8889), (25243.0556,51452.5000), (25252.5000,51559.1667),
(25253.8889,51535.2778), (25253.8889,51549.7222), (25256.9444,51398.8889),
(25263.6111,51516.3889), (25265.8333,51545.2778), (25266.6667,50969.1667),
(25266.6667,51483.3333), (25270.5556,51532.7778), (25270.8333,51505.8333),
(25270.8333,51523.0556), (25275.8333,51533.6111), (25277.2222,51547.7778),
(25278.3333,51525.5556), (25278.3333,51541.3889), (25279.1667,51445.5556),
(25281.1111,51535.0000), (25281.3889,51512.5000), (25283.3333,51533.3333),
(25283.6111,51546.6667), (25284.7222,51555.2778), (25286.1111,51504.1667),
(25286.1111,51534.1667), (25286.6667,51533.3333), (25287.5000,51537.7778),
(25288.0556,51546.6667), (25290.8333,51528.3333), (25291.9444,51424.4444),
(25292.5000,51520.8333), (25298.6111,51001.6667), (25300.8333,51394.4444),
(25306.9444,51507.7778), (25311.9444,51003.0556), (25313.8889,50883.3333),
(25315.2778,51438.6111), (25316.6667,50766.6667), (25320.5556,51495.5556),
(25322.5000,51507.7778), (25325.2778,51470.0000), (25326.6667,51350.2778),
(25337.5000,51425.0000), (25339.1667,51173.3333), (25340.5556,51293.6111),
(25341.9444,51507.5000), (25358.8889,51333.6111), (25363.6111,51281.1111),
(25368.6111,51226.3889), (25374.4444,51436.6667), (25377.7778,51294.7222),
(25396.9444,51422.5000), (25400.0000,51183.3333), (25400.0000,51425.0000),
(25404.7222,51073.0556), (25416.9444,51403.8889), (25416.9444,51457.7778),
(25419.4444,50793.6111), (25429.7222,50785.8333), (25433.3333,51220.0000),
(25440.8333,51378.0556), (25444.4444,50958.3333), (25451.3889,50925.0000),
(25459.1667,51316.6667), (25469.7222,51397.5000), (25478.0556,51362.5000),
(25480.5556,50938.8889), (25483.3333,51383.3333), (25490.5556,51373.6111),
(25492.2222,51400.2778), (25495.0000,50846.6667), (25495.0000,50965.2778),
(25497.5000,51485.2778), (25500.8333,50980.5556), (25510.5556,51242.2222),
(25531.9444,51304.4444), (25533.3333,50977.2222), (25538.8889,51408.3333),
(25545.8333,51387.5000), (25549.7222,51431.9444), (25550.0000,51433.3333),
(25560.2778,51158.6111), (25566.9444,51484.7222), (25567.5000,50958.8889),
(25574.7222,51486.3889), (25585.5556,51151.3889), (25609.4444,51092.2222),
(25610.2778,51475.2778), (25622.5000,51454.4444), (25645.8333,51450.0000),
(25650.0000,51372.2222), (25666.9444,51174.4444), (25683.8889,51505.8333),
(25686.3889,51468.8889), (25696.1111,51260.8333), (25700.8333,51584.7222),
(25708.3333,51591.6667), (25716.6667,51050.0000), (25717.5000,51057.7778),
(25723.0556,51004.1667), (25734.7222,51547.5000), (25751.1111,51449.1667),
(25751.9444,50920.8333), (25758.3333,51395.8333), (25765.2778,51019.7222),
(25772.2222,51483.3333), (25775.8333,51023.0556), (25779.1667,51449.7222),
(25793.3333,51409.4444), (25808.3333,51060.5556), (25816.6667,51133.3333),
```

```
(25823.6111,51152.5000), (25826.6667,51043.8889), (25829.7222,51245.2778),
(25833.3333,51072.2222), (25839.1667,51465.2778), (25847.7778,51205.8333),
(25850.0000,51033.3333), (25856.6667,51083.3333), (25857.5000,51298.8889),
(25857.5000,51441.3889), (25866.6667,51066.6667), (25867.7778,51205.5556),
(25871.9444,51354.7222), (25872.5000,51258.3333), (25880.8333,51221.3889),
(25883.0556,51185.2778), (25888.0556,51386.3889), (25900.0000,51000.0000),
(25904.1667,51201.6667), (25928.3333,51337.5000), (25937.5000,51313.3333),
(25944.7222,51456.3889), (25950.0000,51066.6667), (25951.6667,51349.7222),
(25957.7778,51075.2778), (25958.3333,51099.4444), (25966.6667,51283.3333),
(25983.3333,51400.0000), (25983.6111,51328.0556), (26000.2778,51294.4444),
(26008.6111,51083.6111), (26016.6667,51333.3333), (26021.6667,51366.9444),
(26033.3333,51116.6667), (26033.3333,51166.6667), (26033.6111,51163.8889),
(26033.6111,51200.2778), (26048.8889,51056.9444), (26050.0000,51250.0000),
(26050.2778,51297.5000), (26050.5556,51135.8333), (26055.0000,51316.1111),
(26067.2222,51258.6111), (26074.7222,51083.6111), (26076.6667,51166.9444),
(26077.2222,51222.2222), (26078.0556,51361.6667), (26083.6111,51147.2222),
(26099.7222,51161.1111), (26108.0556,51244.7222), (26116.6667,51216.6667),
(26123.6111,51169.1667), (26123.6111,51222.7778), (26133.3333,51216.6667),
(26133.3333,51300.0000), (26150.2778,51108.0556),
    ]  # yapf: disable
    data['num_vehicles'] = 1
    data['depot'] = 0
    return data


def compute_euclidean_distance_matrix(locations):
    """Creates callback to return distance between points."""
    distances = {}
    for from_counter, from_node in enumerate(locations):
        distances[from_counter] = {}
        for to_counter, to_node in enumerate(locations):
            if from_counter == to_counter:
                distances[from_counter][to_counter] = 0
            else:
                # Euclidean distance
                distances[from_counter][to_counter] = (int(
                    100*math.hypot((from_node[0] - to_node[0]),
                                   (from_node[1] - to_node[1]))))
    return distances


def print_solution(manager, routing, solution):
    """Prints solution on console."""
    print('Objective: {}'.format(solution.ObjectiveValue()))
    index = routing.Start(0)
    plan_output = 'Route:\n'
    route_distance = 0
    while not routing.IsEnd(index):
        plan_output += ' {} ->'.format(manager.IndexToNode(index))
        previous_index = index
        index = solution.Value(routing.NextVar(index))
        route_distance += routing.GetArcCostForVehicle(previous_index, index, 0)
    plan_output += ' {}\n'.format(manager.IndexToNode(index))
    print(plan_output)
    plan_output += 'Objective: {}m\n'.format(route_distance)


def main():
    """Entry point of the program."""
    # Instantiate the data problem.
    data = create_data_model()
    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data['locations']),
                                           data['num_vehicles'], data['depot'])
    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)
```

```
    distance_matrix = compute_euclidean_distance_matrix(data['locations'])

    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return distance_matrix[from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic = (
        routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.seconds = 20
    search_parameters.log_search = False




    # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print solution on console.
    if solution:
        print_solution(manager, routing, solution)


if __name__ == '__main__':
    t = time.time()
    main()
    elapsed = time.time() - t
    print('Solution find at ',elapsed,' seconds')
```

Додаток Д

Повний код програми яка розбиває маршрут на кластери.

```
import math
import time
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

def compute_euclidean_distance_matrix(locations):
    """Creates callback to return distance between points."""
    distances = []
    for from_counter, from_node in enumerate(locations):
        distances.append ([])
        for to_counter, to_node in enumerate(locations):
            if from_counter == to_counter:
                distances[from_counter].append(0)
            elif from_counter < to_counter:
                # Euclidean distance
                distances[from_counter].append(int(
                    10*math.hypot((from_node[0] - to_node[0]),
                                  (from_node[1] - to_node[1]))))
            else:
                distances[from_counter].append(distances[to_counter][from_counter])
    return distances


def create_data_model():
    """Stores the data for the problem."""
    data = {}
    # Locations in block units
    data['locations'] = [
       (24748.3333,50840.0000), (24758.8889,51211.9444), (24827.2222,51394.7222),
(24904.4444,51175.0000), (24996.1111,51548.8889), (25010.0000,51039.4444),
(25030.8333,51275.2778), (25067.7778,51077.5000), (25100.0000,51516.6667),
(25103.3333,51521.6667), (25121.9444,51218.3333), (25150.8333,51537.7778),
(25158.3333,51163.6111), (25162.2222,51220.8333), (25167.7778,51606.9444),
(25168.8889,51086.3889), (25173.8889,51269.4444), (25210.8333,51394.1667),
(25211.3889,51619.1667), (25214.1667,50807.2222), (25214.4444,51378.8889),
(25223.3333,51451.6667), (25224.1667,51174.4444), (25233.3333,51333.3333),
(25234.1667,51203.0556), (25235.5556,51330.0000), (25235.5556,51495.5556),
(25242.7778,51428.8889), (25243.0556,51452.5000), (25252.5000,51559.1667),
(25253.8889,51535.2778), (25253.8889,51549.7222), (25256.9444,51398.8889),
(25263.6111,51516.3889), (25265.8333,51545.2778), (25266.6667,50969.1667),
(25266.6667,51483.3333), (25270.5556,51532.7778), (25270.8333,51505.8333),
(25270.8333,51523.0556), (25275.8333,51533.6111), (25277.2222,51547.7778),
(25278.3333,51525.5556), (25278.3333,51541.3889), (25279.1667,51445.5556),
(25281.1111,51535.0000), (25281.3889,51512.5000), (25283.3333,51533.3333),
(25283.6111,51546.6667), (25284.7222,51555.2778), (25286.1111,51504.1667),
(25286.1111,51534.1667), (25286.6667,51533.3333), (25287.5000,51537.7778),
(25288.0556,51546.6667), (25290.8333,51528.3333), (25291.9444,51424.4444),
(25292.5000,51520.8333), (25298.6111,51001.6667), (25300.8333,51394.4444),
(25306.9444,51507.7778), (25311.9444,51003.0556), (25313.8889,50883.3333),
(25315.2778,51438.6111), (25316.6667,50766.6667), (25320.5556,51495.5556),
(25322.5000,51507.7778), (25325.2778,51470.0000), (25326.6667,51350.2778),
(25337.5000,51425.0000), (25339.1667,51173.3333), (25340.5556,51293.6111),
(25341.9444,51507.5000), (25358.8889,51333.6111), (25363.6111,51281.1111),
(25368.6111,51226.3889), (25374.4444,51436.6667), (25377.7778,51294.7222),
(25396.9444,51422.5000), (25400.0000,51183.3333), (25400.0000,51425.0000),
(25404.7222,51073.0556), (25416.9444,51403.8889), (25416.9444,51457.7778),
(25419.4444,50793.6111), (25429.7222,50785.8333), (25433.3333,51220.0000),
(25440.8333,51378.0556), (25444.4444,50958.3333), (25451.3889,50925.0000),
(25459.1667,51316.6667), (25469.7222,51397.5000), (25478.0556,51362.5000),
(25480.5556,50938.8889), (25483.3333,51383.3333), (25490.5556,51373.6111),
(25492.2222,51400.2778), (25495.0000,50846.6667), (25495.0000,50965.2778),
```

```
(25497.5000,51485.2778), (25500.8333,50980.5556), (25510.5556,51242.2222),
(25531.9444,51304.4444), (25533.3333,50977.2222), (25538.8889,51408.3333),
(25545.8333,51387.5000), (25549.7222,51431.9444), (25550.0000,51433.3333),
(25560.2778,51158.6111), (25566.9444,51484.7222), (25567.5000,50958.8889),
(25574.7222,51486.3889), (25585.5556,51151.3889), (25609.4444,51092.2222),
(25610.2778,51475.2778), (25622.5000,51454.4444), (25645.8333,51450.0000),
(25650.0000,51372.2222), (25666.9444,51174.4444), (25683.8889,51505.8333),
(25686.3889,51468.8889), (25696.1111,51260.8333), (25700.8333,51584.7222),
(25708.3333,51591.6667), (25716.6667,51050.0000), (25717.5000,51057.7778),
(25723.0556,51004.1667), (25734.7222,51547.5000), (25751.1111,51449.1667),
(25751.9444,50920.8333), (25758.3333,51395.8333), (25765.2778,51019.7222),
(25772.2222,51483.3333), (25775.8333,51023.0556), (25779.1667,51449.7222),
(25793.3333,51409.4444), (25808.3333,51060.5556), (25816.6667,51133.3333),
(25823.6111,51152.5000), (25826.6667,51043.8889), (25829.7222,51245.2778),
(25833.3333,51072.2222), (25839.1667,51465.2778), (25847.7778,51205.8333),
(25850.0000,51033.3333), (25856.6667,51083.3333), (25857.5000,51298.8889),
(25857.5000,51441.3889), (25866.6667,51066.6667), (25867.7778,51205.5556),
(25871.9444,51354.7222), (25872.5000,51258.3333), (25880.8333,51221.3889),
(25883.0556,51185.2778), (25888.0556,51386.3889), (25900.0000,51000.0000),
(25904.1667,51201.6667), (25928.3333,51337.5000), (25937.5000,51313.3333),
(25944.7222,51456.3889), (25950.0000,51066.6667), (25951.6667,51349.7222),
(25957.7778,51075.2778), (25958.3333,51099.4444), (25966.6667,51283.3333),
(25983.3333,51400.0000), (25983.6111,51328.0556), (26000.2778,51294.4444),
(26008.6111,51083.6111), (26016.6667,51333.3333), (26021.6667,51366.9444),
(26033.3333,51116.6667), (26033.3333,51166.6667), (26033.6111,51163.8889),
(26033.6111,51200.2778), (26048.8889,51056.9444), (26050.0000,51250.0000),
(26050.2778,51297.5000), (26050.5556,51135.8333), (26055.0000,51316.1111),
(26067.2222,51258.6111), (26074.7222,51083.6111), (26076.6667,51166.9444),
(26077.2222,51222.2222), (26078.0556,51361.6667), (26083.6111,51147.2222),
(26099.7222,51161.1111), (26108.0556,51244.7222), (26116.6667,51216.6667),
(26123.6111,51169.1667), (26123.6111,51222.7778), (26133.3333,51216.6667),
(26133.3333,51300.0000), (26150.2778,51108.0556),
    ]  # yapf: disable
    data['distance_matrix'] = compute_euclidean_distance_matrix(data['locations'])
    data['num_vehicles'] = 4
    data['depot'] = 0
    return data


def print_solution(data, manager, routing, solution):
    """Prints solution on console."""
    print(f'Objective: {solution.ObjectiveValue()}')
    max_route_distance = 0
    for vehicle_id in range(data['num_vehicles']):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for vehicle {}:\n'.format(vehicle_id)
        route_distance = 0
        while not routing.IsEnd(index):
            plan_output += ' {} -> '.format(manager.IndexToNode(index))
            previous_index = index
            index = solution.Value(routing.NextVar(index))
            route_distance += routing.GetArcCostForVehicle(
                previous_index, index, vehicle_id)
        plan_output += '{}\n'.format(manager.IndexToNode(index))
        plan_output += 'Distance of the route: {}m \n'.format(route_distance)
        print(plan_output)
        max_route_distance = max(route_distance, max_route_distance)
    print('Maximum of the route distances: {}m'.format(max_route_distance))


def main():
    """Entry point of the program."""
    # Instantiate the data problem.
    data = create_data_model()
```

```
    # Create the routing index manager.
    manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
                                           data['num_vehicles'], data['depot'])

    # Create Routing Model.
    routing = pywrapcp.RoutingModel(manager)


    # Create and register a transit callback.
    def distance_callback(from_index, to_index):
        """Returns the distance between the two nodes."""
        # Convert from routing variable Index to distance matrix NodeIndex.
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return data['distance_matrix'][from_node][to_node]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)

    # Define cost of each arc.
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # Add Distance constraint.
    dimension_name = 'Distance'
    routing.AddDimension(
        transit_callback_index,
        0,  # no slack
        100000,  # vehicle maximum travel distance
        True,  # start cumul to zero
        dimension_name)
    distance_dimension = routing.GetDimensionOrDie(dimension_name)
    distance_dimension.SetGlobalSpanCostCoefficient(100)

    # Setting first solution heuristic.
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = (
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
    search_parameters.local_search_metaheuristic = (
        routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)
    search_parameters.time_limit.seconds = 30
    search_parameters.log_search = False

    # Solve the problem.
    solution = routing.SolveWithParameters(search_parameters)

    # Print solution on console.
    if solution:
        print_solution(data, manager, routing, solution)
    else:
        print('No solution found !')


if __name__ == '__main__':
    t = time.time()
    main()
    elapsed = time.time() - t
    print('Solution find at ',elapsed,' seconds')
```