МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних систем_____

«До захисту в ЕК»	**«До захисту допущено»**
Директор інституту(декан факультету)	Завідувач кафедри
_____ Андрій Форсюк	_____ Сергій Чумаченко
(підпис) (ім'я та прізвище)	(підпис) (ім'я та прізвище)
«___» _____ 2022 р.	«___» _____ 2022 р.

КВАЛІФІКАЦІЙНА РОБОТА
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ МАГІСТРА

зі спеціальності 122 "Комп'ютерні науки"
(код та назва спеціальності)
освітньо-професійної програми Інформаційні управляючі системи та технології

на тему: Дослідження та розроблення інформаційної системи підтримки організації ефективних перевезень

Виконав: здобувач 2 курсу, групи ІУС-2-3М

Устимук Іван Васильович _____
(прізвище, ім'я, по батькові повністю) (підпис)

Керівник Грибков Сергій Віталійович _____
(прізвище , ім'я та по батькові повністю) (підпис)

Консультанти _____ _____
(ім'я та прізвище) (підпис)

_____ _____
(ім'я та прізвище) (підпис)

Рецензент Лідія Власенко _____
(ім'я та прізвище) (підпис)

Я як здобувач(ка) Національного університету харчових технологій розумію і підтримую політику університету з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволеної допомоги під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач_____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ХАРЧОВИХ ТЕХНОЛОГІЙ

Інститут (факультет) Автоматизації і комп'ютерних систем
Кафедра Інформаційних систем
Освітній ступінь магістр
Спеціальність 122 «Комп'ютерні науки»
(код і назва)
Освітньо-професійна програма Інформаційні управляючі системи та технології
(назва)

ЗАТВЕРДЖУЮ
Завідувач
кафедри Інформаційних систем

Чумаченко С. М.
"____" _____2022 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Устимук Іван Васильович
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та розроблення інформаційної системи організації ефективних перевезень
керівник роботи Грибков Сергій Віталійович, д.т.н., доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 11 " 11 2021 року № 884-кс
2. Строк подання здобувачем роботи 01.02.2022
3. Вихідні дані до роботи Інформація про ФОП «Устимук В. С.». Положення про логістичний ФОП «Устимук І. В.». Посадові інструкції усіх працівників відділу логістики. Звітна документація відділу логістики про виконання замовлень. Сучасні публікації вітчизняних та зарубіжних авторів по створенню та розвитку інформаційних продуктів та алгоритмів рішень задач маршрутизації транспорту.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1. Вступ; 2. Дослідження предметної області та постановка задачі; 3. Дослідження методів, алгоритмів та програмних засобів розв'язання задачі; 4. Розробка та апробація інформаційної системи; 5. Основні висновки та результати; 6. Список використаних джерел; 7. Додатки

5. Перелік графічного матеріалу Діаграми функціональної моделі. Діаграми моделі даних. Скріншоти системи.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3	Грибков С. В., доцент	11.11.21	30.01.22

7. Дата видачі завдання 11.11.2021

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної області та постановка задачі	11.12.2021-	Виконано
2	Дослідження методів, алгоритмів та програмних засобів розв'язання задачі	30.12.2021-	Виконано
3	Розробка та апробація інформаційної системи	30.12.2021-	Виконано
4	Оформлення роботи	30.01.2021-	Виконано
5	Підготовка автореферату	01.02.2021-	Виконано
6	Підготовка презентації		

Здобувач _____ Устимук І. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Грибков С. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Устимук І. В. «Дослідження та розроблення інформаційної системи підтримки організації ефективних перевезень».

Магістерська робота на здобуття ступеню магістра за спеціальністю 122 Комп'ютерні науки – Національний університет харчових технологій. Київ. 2022 рік.

В даній роботі були проаналізовані та досліджені види задач маршрутизації транспорту, досліджені методи рішення ЗМТ, застосовано евристичний алгоритм вирішення задачі маршрутизації транспорту після чого він був інтегрований до СППР логіста ФОП "Устимук В. С.".

Отриманий в результаті програмний продукт, удосконалить процес прийняття рішень логіста при виборі автомобілів для виконання замовлень та прокладання маршруту для виконання замовлень за рахунок використання інформаційних технологій.

КЛЮЧОВІ СЛОВА: ЕВРИСТИЧНИЙ АЛГОРИТМ, ЗАДАЧА МАРШРУТИЗАЦІЇ ТРАНСПОРТУ, ПРОКЛАДАННЯ МАРШРУТУ.

ANNOTATION

Ustymuk I.V. “Research and development of information system to support the organization of efficient transportation”.

Master work for the master's degree in 122 Computer science — National University of Food Technology. Kyiv. 2022.

In this paper, the types of transport routing problems were analyzed and investigated, methods of solving the MRT were investigated, a heuristic algorithm for solving the transport routing problem was applied, after which it was integrated into the DSS of «Ustimuk V.S».

The resulting software product will improve the logistician's decision-making process when choosing cars to fulfill orders and paving the route for fulfilling orders through the use of information technology.

KEYWORDS: HEURISTIC ALGORITHM, PROBLEM OF ROUTING OF TRANSPORT, LAYING OF ROUTE.

ЗМІСТ

ВСТУП

Актуальність теми. Кожен рік попит на транспортно-логістичні послуги підвищується по всьому світі. В наслідок цього, проблема вирішення задач маршрутизації транспорту (ЗМТ) стає все більш актуальною і затребуваною. Головною ціллю кожного підприємства, яке займається вантажними і не тільки вантажними перевезеннями є — побудова оптимальних маршрутів для своїх транспортних засобів для виконання замовлень.

Кількість видів задач маршрутизації транспорту є дуже великим, як і алгоритмів, які їх вирішують. Це точні, евристичні та метаевристичні алгоритми. Так як задача маршрутизації транспорту є NP-складною[1], очевидно, що точні підходи є сенс використовувати тільки при задачах з малою к-стю вхідних даних. Тому, для рішення задач маршрутизації транспорту частіше надають перевагу евристичним алгоритмам. Вони створюють рішення, наближені до оптимального, але за менший час (в порівнянні з точними методами). Також ще одною особливістю евристичних алгоритмів являється різноманітність рішень, які отримуються під час роботи алгоритму з одним набором даних.

Тема магістерської кваліфікаційної роботи тісно переплітається з темою бакалаврської кваліфікаційної роботи, оскільки, обидві роботи виконані під час дослідження логістичного відділу ФОП “Устимук В. С.” для часткової автоматизації роботи даного відділу. Ті задачі, які були виконані в ході виконання магістерської кваліфікаційної роботи, по суті, є продовженням тієї автоматизації та оптимізації, яка була розпочата під час виконання бакалаврської кваліфікаційної роботи. Після створення системи підтримки прийняття рішень логіста під час виконання бакалаврської роботи, значно спростилась робота логіста, а її швидкість виросла, через легкий доступ до всіх необхідних даних через СППР, а також її допомогу в прийнятті рішень логіста. На теперішній час, найбільшим запитанням залишилось побудова

оптимальних маршрутів, та вибір транспортних засобів, для виконання всіх замовлень, при найменших витратах.

Зв'язок роботи з науковими програмами, планами, темами. Наукова робота виконується згідно з планом та програмою наукових досліджень на кафедрі інформаційних систем Національного університету харчових технологій (НУХТ): № ДР 0117U003475 «Дослідження та впровадження інформаційних технологій у галузях харчової промисловості та освіті» (на червень 2017 — травень 2022 р.).

Об'єкт дослідження: Процеси прийняття рішень при виборі автомобілів та прокладання маршруту для виконання замовлень в логістичних фірмах..

Предмет дослідження: Методи, технології та інструментальні засоби удосконалення процесу прийняття рішень при управлінні логістичних фірм.

Мета й завдання дослідження. Метою кваліфікаційної магістерської роботи є дослідження видів ЗМТ, ідентифікація, до якого з видів належить ЗМТ підприємства, вибір алгоритму для реалізації, розробка алгоритму вирішення ЗМТ для подальшої інтеграції алгоритму до СППР логіста ФОП "Устимук В. С.".

Для досягнення зазначеної мети необхідно розв'язати наступні завдання:

- дослідити предметну область для виявлення основних видів ЗМТ;
- дослідити існуючі алгоритми вирішення ЗМТ та обрати найбільш ефективний для обраної предметної області;
- удосконалити існуючу СППР за рахунок використання обраних алгоритмів та провести апробацію для оцінки ефективності їх використання для побудови маршрутів в реальних логістичних завданнях.

Методи дослідження: метод структурно-функціонального аналізу для дослідження предметної області; методи моделювання та побудови баз даних, для аналізу та удосконалення бази даних системи; евристичні та метаевристичні алгоритми для розв'язання задачі побудови маршрутів; об'єктно-орієнтоване програмування для створення елементів системи.

Наукова новизна одержаних результатів. Наукове значення роботи полягає в удосконаленні діючих підходів до вирішення задач маршрутизації транспортних засобів.

Практичне значення отриманих результатів. Практичне значення результатів магістерських досліджень полягає у створенні алгоритму вирішення задачі маршрутизації транспорту, для подальшої інтеграції даного алгоритму в існуючі інформаційні системи для автоматизації процесу побудови маршрутів та вибору автомобілів.

Структура та обсяг магістерської роботи. Магістерська робота складається із вступу, трьох розділів, загальних висновків, бібліографічного списку та додатків. Повний обсяг роботи складається з 80 сторінок, списку використаних джерел із 25 найменувань, який викладено на 2 сторінках, та додатків на 1 сторінці.

Наукова робота містить 43 рисунки та 4 таблиці.

РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СУЧАСНОЇ СХЕМИ ОРГАНІЗАЦІЇ РОБОТИ ЛОГІСТИЧНОГО ВІДДІЛУ ФОП ”УСТИМУК В.С.” ТА ШЛЯХИ ЙОГО УДОСКОНАЛЕННЯ

1.1. Загальна характеристика ФОП “Устимук В.С.” та схеми роботи логістичного відділу

ФОП “Устимук В.С.” — логістичне підприємство, з ланки малого бізнесу, яке займається вантажними перевезеннями на території України та за її межами. Організаційно-функціональна схема представлена на рисунку 1.

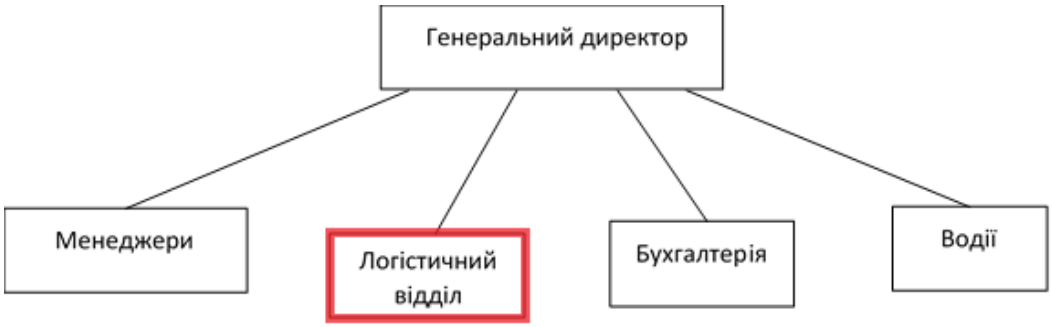

Рис. 1. Організаційно-функціональна схема ФОП “Устимук В. С.”

Це класичне мале логістичне підприємство, яких зараз безліч по всій Україні, а особливо їх багато в західних регіонах України, адже левова частка логістики припадає на доставку вантажів з України в ЄС та навпаки. Дуже часто ці підприємства починають свій шлях з того, що всі ролі виконує одна людина і поступово розвивається. Більше детально дане підприємство було розглянуто при виконані бакалаврської кваліфікаційної роботи.

На даний момент логістичний відділ працює в створеній раніше системі підтримки прийняття рішень логіста і основні задачі логістичного відділу можна побачити, розглянувши діаграму “Вибір замовлень для виконання”, представлену на рисунку 2.

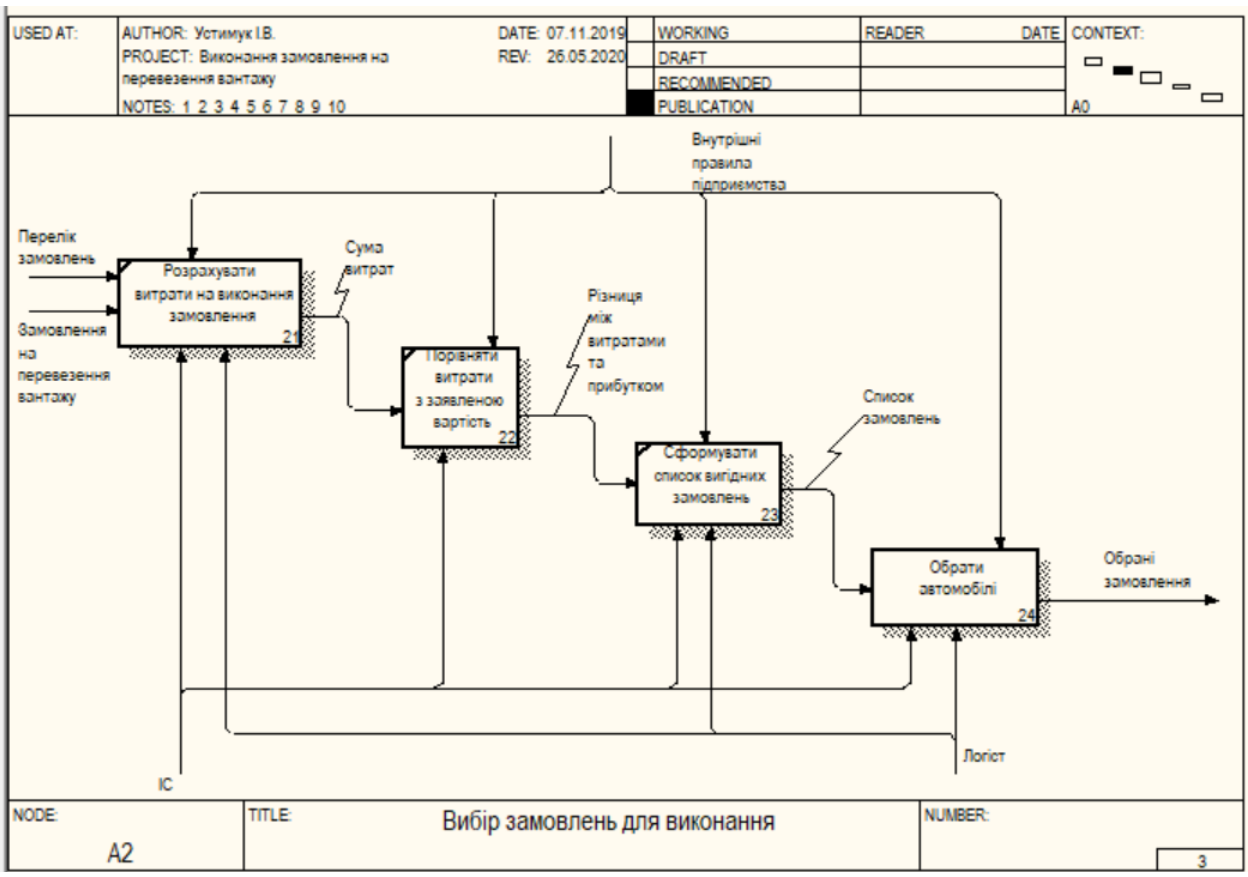

Рис. 2. Схема роботи логістичного відділу

Інтеграція СППР логіста, дала видиме покращення роботи логістичного відділу, та підприємства в цілому, а саме:

- Виросла швидкість обробки замовлень;
- Частина роботи логіста була автоматизована;
- Документообіг перейшов в цифровий вигляд.

На даний момент, найслабшою ланкою роботи системи являється вибір автомобіля для виконання замовлення, а також побудови маршрутів для виконання замовлень.

У результаті дослідження системи підтримки прийняття рішень логіста, було вирішено розробити алгоритм для побудови маршрутів та вибору автомобілів для виконання замовлень, для подальшої інтеграції цього алгоритму до існуючої СППР.

Для реалізації поставленої задачі необхідно розробити алгоритм вирішення задачі маршрутизації транспорту.

1.2. Задача маршрутизації транспорту

Задачі маршрутизації транспорту (Vehicle Routing Problems, VRP) — це задачі комбінаторної оптимізації, в яких для парку транспортних засобів, розміщених в одному або декількох депо, необхідно визначити набір маршрутів до декількох віддалених точок-клієнтів (рис. 3).

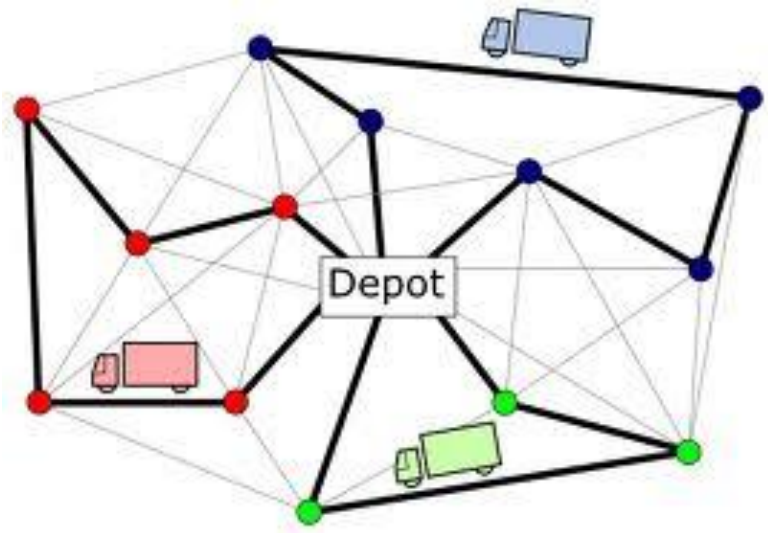

Рис. 3. Графічне представлення ЗМТ

Інтерес до ЗМТ викликаний її практичною значимістю при значній складності самої задачі.

VRP — добре відома задача цілочисельного програмування, яка відноситься до класу NP-складних задач, що означає, складність обрахунку задачі залежить від розміру вхідних даних.

Для таких задач зазвичай достатньо знайти наближені рішення, які знаходяться достатньо швидко і є достатньо точними для необхідних цілей. Зазвичай це досягається різними евристичними методами.

Задачі маршрутизації транспорту лежать на пересіченні двох добре досліджених задач: задачі комівояжера та задачі про пакування в ємності.

Задача комівояжера (Traveling Salesman Problem, TSP): якщо вантажомісткість кожного транспортного засобу вважається нескінченною, або точніше достатньою, то задача VRP здводиться до множинної задачі комівояжера (Multiple Traveling Salesman Problem, MTSP) шляхом додавання до початкового графа k -1 (де k – к-сть маршрутів) копій нульової вершини і її ребер (між цими копіями ребра відсутні) (рис. 4).

Рис. 4. Комівояжер(подорожучий торговець)

Задача про пакування в ємності (Bin Packing Problem, BPP) (Рис. 5): рішенням цієї задачі тотожно рішенню задачі VRP при умові, що всі відстані вважаються рівними нулю (тобто ефективність всіх підходящих рішень буде однакова).

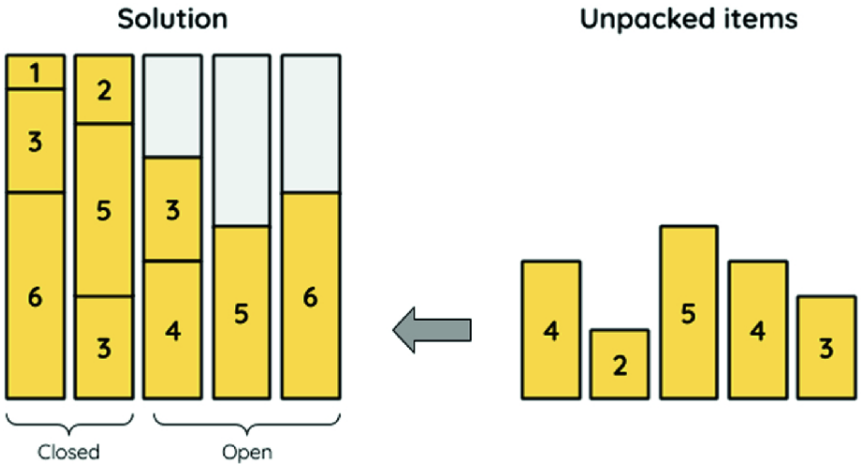

Рис. 5. Задача про пакування в ємності

Не дивлячись на те, що ЗМТ являється розвитком задачі комівояжера (ЗК), і ці задачі споріднені, варто відмітити, що в ЗМТ прийнята термінологія, яка відрізняється від термінології ЗК. У випадку якщо в ЗК використовують такі поняття як “комівояжер”, “місто”, “шлях”, то в ЗМТ “комівояжер” замінений на “транспортний засіб”, “місто” на “вершину”, “шлях” на “маршрут”. В окремих варіантах “вершина” заміняються синонімом “клієнт”. Окрім того, в ЗМТ з’являється ще одне допоміжне слово — “депо”.

Це така особлива вершина, де розпочинаються і закінчуються маршрути абсолютно всіх транспортних засобів. Як правило, депо являє собою склад зберігання продукції, необхідної для доставки замовлень клієнтів. У всіх видах ЗМТ передбачається наявність як мінімум одного депо.

Окремо акцентується увага на варіації задачі, в якій розглядається випадок коли існує більше одного депо. У випадку, коли в задачі існує тільки одне депо, то всі транспортні засоби повинні починати свій шлях з депо і закінчувати свій шлях повертаючись до його. Іншими словами, всі маршрути повинні включати в себе вершину депо.

Якщо ж депо декілька, то кожен маршрут повинен включати вершину тільки одного з них. В задачі присутнє таке поняття, як вартість об’їзду послідовності вершин транспортним засобом. Це ключовий параметр, який потребує оптимізації. На практиці він показує будь-які витрати, пов’язані з відвідуванням клієнтів і може враховувати як вартість палива, яке використовується, так і час, необхідний для виконання роботи.

Для дослідження алгоритмів нам не дуже важливо, які саме витрати ховаються за цим параметром в дійсності. Будемо вважати його як узагальнення всіх видів витрат на переміщення. Але обов’язково треба враховувати, що вартість проїзду не може бути від’ємною.

Задачі маршрутизації являються ключовими в сферах вантажних перевезень, переміщення і логістики. В багатьох частинах ринку доставка товару додає до його вартості суму, яку можна порівняти з вартістю цього

товару. Не зважаючи на це, використання комп’ютерних методів оптимізації доставки товару часто дає змогу зекономити 5-20% від загальної його вартості.

1.3. Види задачі маршрутизації транспорту

Зазвичай в реальних задачах оптимізації виникає маса додаткових обмежень і варіацій, найбільш важливі з них представлені на рисунку 6.

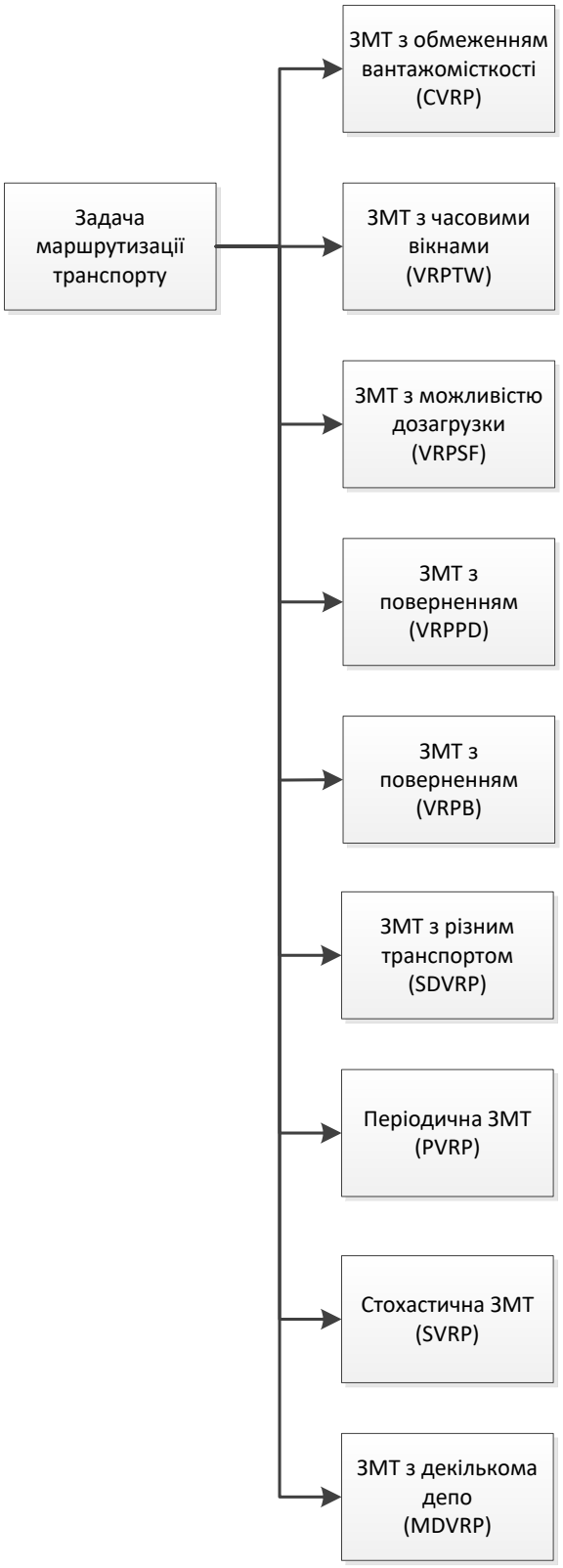

Рис. 6. Види обмежень ЗМТ

Маршрутизація з урахуванням вантажомісткості (Capacitated VRP, CVRP)

В задачах цього типу вводиться додаткове обмеження: обсяг вантажів на кожному маршруті не повинен перевищувати заданої величини Q (однакової для всіх авто).

Основна мета: Мінімізувати парк автомобілів, необхідних для виконання завдання, а також загальний час виконання задачі.

Маршрутизація з часовими вікнами (VRP with Time Windows, VRPTW)

Ця задача подібна до класичної ЗМТ з основною додатковою умовою: для виконання кожного запиту кожного клієнта існує відомий часовий інтервал - розмічений горизонт (scheduling horizon) (рис. 7).

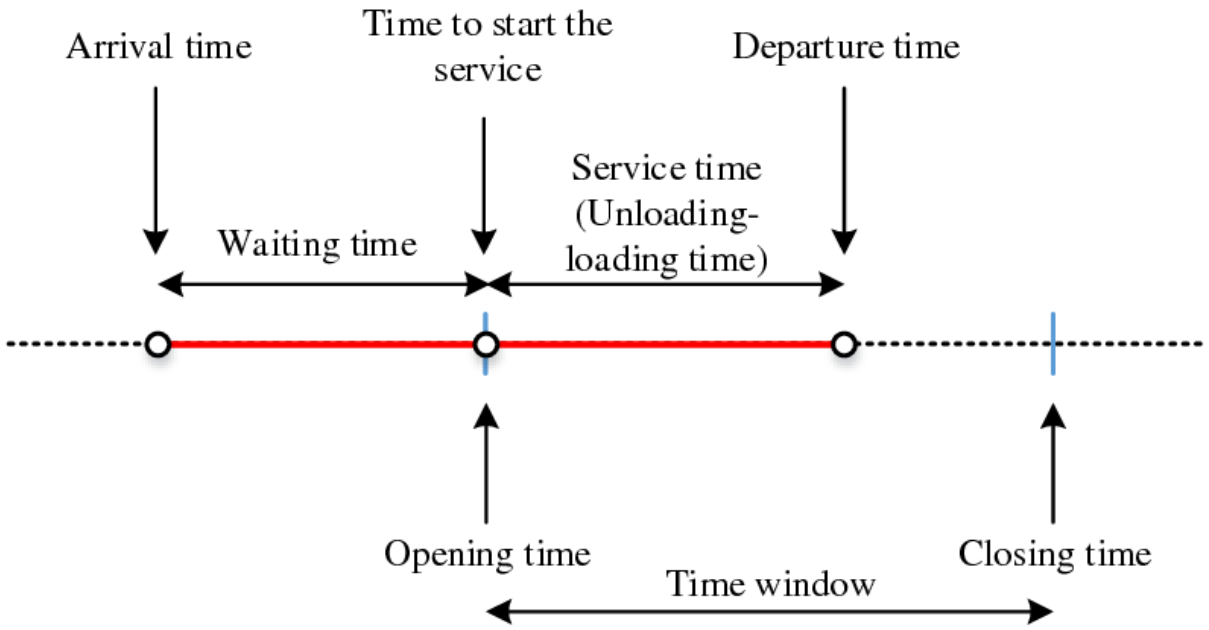

Рис. 7. Розмічений горизонт

В порівнянні з класичною ЗМТ, в задачах цього типу додаються наступні умови. Розв’язок задачі незадовільний, якщо:

- клієнт обслуговується після верхньої часової межі;

- машина, яка прибула раніше нижньої часової межі очікує, коли вона наступить;
- як варіант, запізнення не впливає на якість рішення, але додає деяке додаткове штрафне значення до цільової функції.

Отримавши рішення задачі такого типу, можна окрім всього іншого, точніше підібрати час відправлення автомобіля з депо і тим самим уникнути непотрібних простоїв.

Основна мета: Мінімізувати к-сть автомобілів, загальний час в дорозі і очікування, необхідні для обробки запитів клієнтів у визначені інтервали часу.

Маршрутизація з багатьма депо (Multiple Depot VRP, MDVRP)

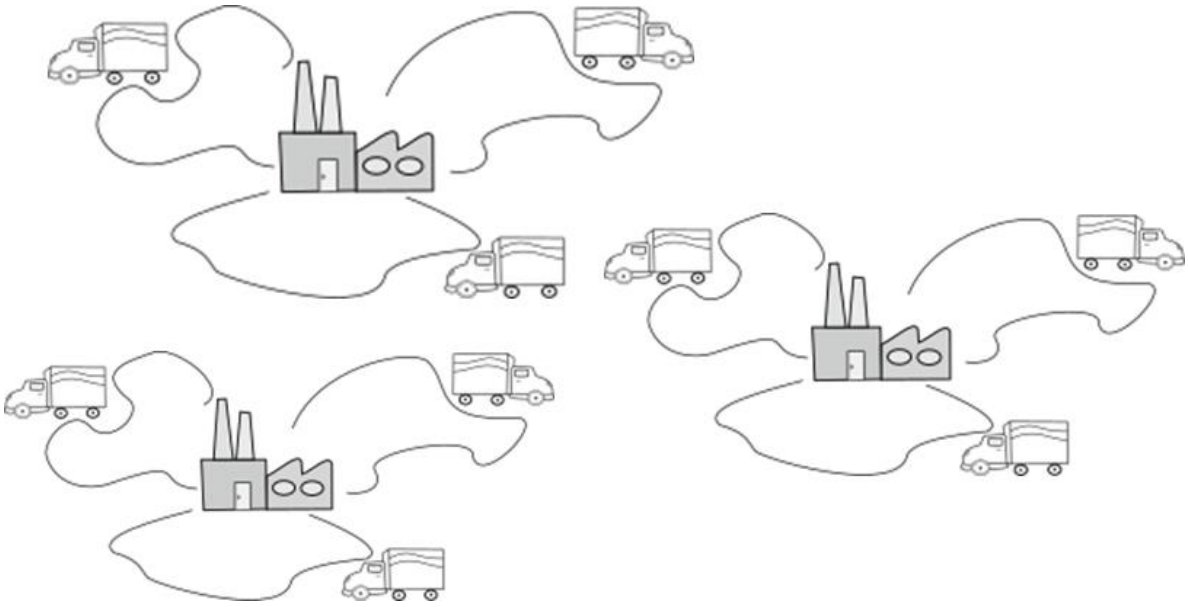

Рис. 8. Маршрутизація з багатьма депо

Існують випадки, коли логістична компанія має декілька депо, які обслуговують клієнтів.

У випадку, коли клієнти згруповані довкола кожного депо, задача може бути розбита на декілька незалежних задач, втім, коли клієнти і депо розташовані хаотично, необхідно вирішити ЗМТ з багатьма депо.

Задача потребує розподілення клієнтів по різним депо. В кожному депо знаходиться свій парк автомобілів. Кожне авто відправляється з депо, де воно розташоване , обслуговує клієнтів і повертається до того ж депо.

Обмеження: Кожен маршрут повинен задовольняти стандартні обмеження ЗМТ, а також розпочинатися і закінчуватися в одному і тому ж депо.

Основна мета: Мінімізувати парк автомобілів і загальний час в дорозі.

Маршрутизація з поверненням товарів (VRP with Pick-Ups and Deliveries, VRPPD)

Задача маршрутизації з можливістю повернення і доставки товару розширює класичну ЗМТ тим, що необхідна доставка певної к-сті товару від клієнтів назад в депо або до постачальника (іншого клієнта). Це означає, що необхідно бути впевненим, що повернені товари не будуть перевищувати загальну вантажомісткість автомобіля.

Це обмеження ускладнює планування задачі і може привести до непродуктивного використання вантажомісткості транспорту, збільшення загального пройденого шляху і к-сті одиниць транспорту.

Для простоти зазвичай розглядаються задачі з додатковими обмеженнями, наприклад, коли всі запити на доставку товарів починаються в депо, і всі запити на повернення закінчуються в депо, тобто, не відбувається обмін товарами між клієнтами.

Інший спосіб полягає в тому, що всі клієнти повинні бути відвідані лише один раз.

Існує ще одне спрощення — вважати, що кожен автомобіль спочатку розвозить всі товари, а потім починає приймати товари від клієнтів (VRP with Backhauls).

Обмеження: кількість товару, який потрібно доставити користувачам і кількість товару, який необхідно забрати від користувачів в депо, не повинен перевищувати вантажомісткість автомобіля ні в одній з точок маршруту.

Основна мета: Мінімізувати парк автомобілів і загальний час переміщень.

Маршрутизація з поверненням товарів (VRP with Backhauls, VRPB)

В попередній задачі (VRPPD) необхідно прийняти до уваги, що товари які повернуть клієнти повинні поміститися в автомобілі. Відмінність від VRPPD полягає в тому, що всі товари повинні бути доставлені, перед тим як відбудеться будь-яке повернення.

Це обмеження випливає з того факту, що всі автомобілі завантажуються виключно через задні двері і перестановка вантажів не являється економічно вигідною і доречною.

Кількість товару, який необхідно доставити і забрати, фіксована і відома заздалегідь.

Обмеження: повернення товару відбувається тільки після доставки всіх товарів. Обсяги товарів при доставці і поверненні не повинен перевищувати вантажомісткість автомобілів.

Основна мета: Знайти такий набір маршрутів, щоб мінімізувати загальну пройдену відстань.

Маршрутизація з різним транспортом(Split Delivery VRP, SDVRP)

Ця задача розширює ЗМТ, дозволяючи обслуговувати одного клієнта різними автомобілями, якщо це зменшує загальну вартість задачі.

Цей варіант типовий для ситуації, коли обсяг замовлення можна порівняти по величині з вантажомісткістю авто.

Як правило, для таких задач отримати оптимальне рішення складніше, ніж для класичної ЗМТ.

Обмеження: на відміну від класичної ЗМТ, в задачах SDVRP знімається обмеження на те, що клієнт повинен бути обслуженим тільки одним автомобілем. Окрім того, парк транспорту включає автомобілі різної вантажомісткості.

Задача SDVRP зводиться до VRP розбиттям кожного замовлення на декілька неділимих замовлень.

Основна мета: Мінімізувати парк автомобілів, і загальний час обслуговування всіх клієнтів.

Періодична маршрутизація (Periodic VRP, PVRP)

В класичній ЗМТ період планування один день, в задачах з періодичною маршрутизацією період планування розширюється до декількох днів.

Якщо період планування M=1, задача зводиться до класичної. Кожен клієнт в даній задачі повинен бути відвіданий k разів, також $1 \leq k \leq M$.

В класичному варіанті PVRP, щоденне замовлення клієнта завжди фіксоване. Періодичну ЗМТ можна розглядати як задачу компонування групи маршрутів на кожен день, також маршрути повинні відповідати накладеним обмеження і загальна вартість задачі повинна бути мінімальна.

Обмеження: ті ж самі шо і в ЗМТ. Окрім того, автомобіль може повернутися в депо не в той самий день. Через M-денний період, кожен клієнт повинен бути відвіданий як мінімум хоча б один раз.

Ціль: Мінімізувати парк автомобілів, і загальний час обслуговування всіх клієнтів.

Маршрутизація з випадковими даними (Stochastic VRP, SVRP)

В цьому варіанті один або декілька компонентів задачі можуть мати випадкову поведінку.

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

Випадкові замовлення: замовлення кожного клієнта — випадкова величина.

Випадковий час поїздки: час поїздки (відстань між клієнтами) — випадкові величини.

Такі задачі вирішуються в два підходи. Перший етап дає рішення без врахування випадкових змінних. А на другому етапі коли змінні стають відомими, відбувається корекція отриманого раніше рішення.

Обмеження: коли деякі данні невідомі, стає неможливим виконання всіх обмежень для всіх випадкових змінних. Таким чином, може бути необхідним

виконання деяких умов із заданою ймовірністю, або, створення корекційної моделі, яка виконується при порушення якихось обмежень.

Наприклад, в задачі SVRP з поверненням товарів і урахуванням вантажомісткості, можливими способами корекції будуть наступні:

- Повернутися в депо, коли автомобіль заповниться, з ціллю розвантажитися, після чого продовжити шлях маршрутом;
- Повернутися в депо, коли автомобіль заповниться і знову оптимізувати частину шляху, що залишився;
- Запланувати повернення в депо, навіть якщо автомобіль не до кінця заповнений. В цьому випадку, рішення може залежати від к-сті зібраного вантажу і відстані до депо. Автомобіль може повернутися в депо, якщо відомо, що вантаж наступного клієнта перевищить вантажомісткість автомобіля.

Мета задачі: Мінімізувати парк автомобілів, і загальний час обслуговування всіх клієнтів.

Маршрутизація з можливістю до завантаження (VRP with Satellite Facilities, VRPSF)

В класичній задачі VRP вважається, що кожен маршрут починається і закінчується в депо.

Однією з причин повернення в депо може бути обмежена вантажомісткість. Коли автомобіль розвіз всі товари, він повинен повернутися в депо за новою порцією товарів.

Однак інколи в деяких випадках буває вигіднішим виконати до завантаження прямо на маршруті, без повернення в депо, за допомогою інших автомобілів.

Типовим є випадок, коли більшість клієнтів очікують регулярних поставок від одного центрального постачальника.

Обмеження: товар на складі клієнта не повинен закінчуватися.

Задача маршрутизації з можливістю до завантаження являється частиною задачі розподілення товару (IRP, Inventory Routing Problem), в зв'язку з чим, повний опис цієї задачі виходить за рамки даної роботи.

Ціль: Мінімізувати витрати на доставку товарів за визначений термін (можливо, що, враховуючи витрати на додаткові машини, загальна вартість рішення задачі в короткочасній перспективі буде вище, ніж наприклад, при рішенні класичної VRP) [2].

1.4. Математична модель задачі

Виходячи зі статті [3], постановка математичної моделі ЗМТ може бути представлена у вигляді неорієнтованого графу (рис. 9).

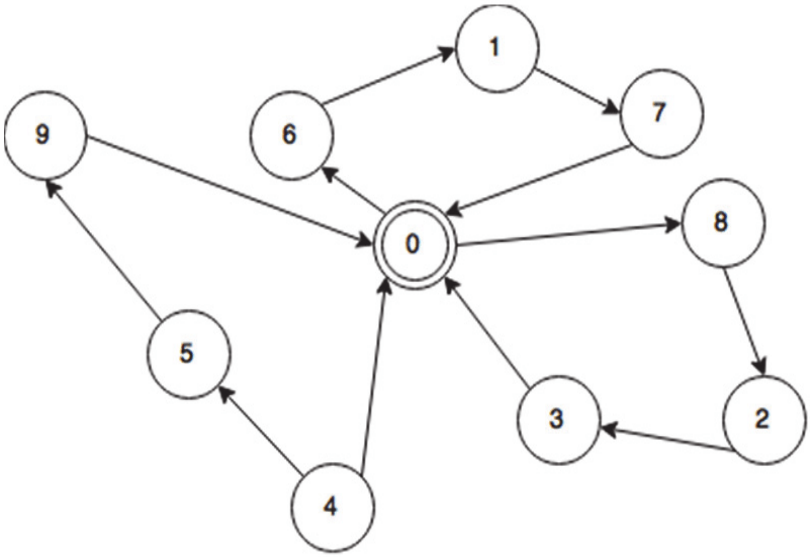

Рис. 9. ЗМТ на графі

Задача маршрутизації транспорту визначена на повному неорієнтованому графі G = (V, E).

Множина V = {0, … , n} є множиною вершин. Кожна вершина i ∈ V / {0} являється клієнтом, який має невід'ємний попит q_i, а вершині 0 відповідає депо, як на рисунку 10.

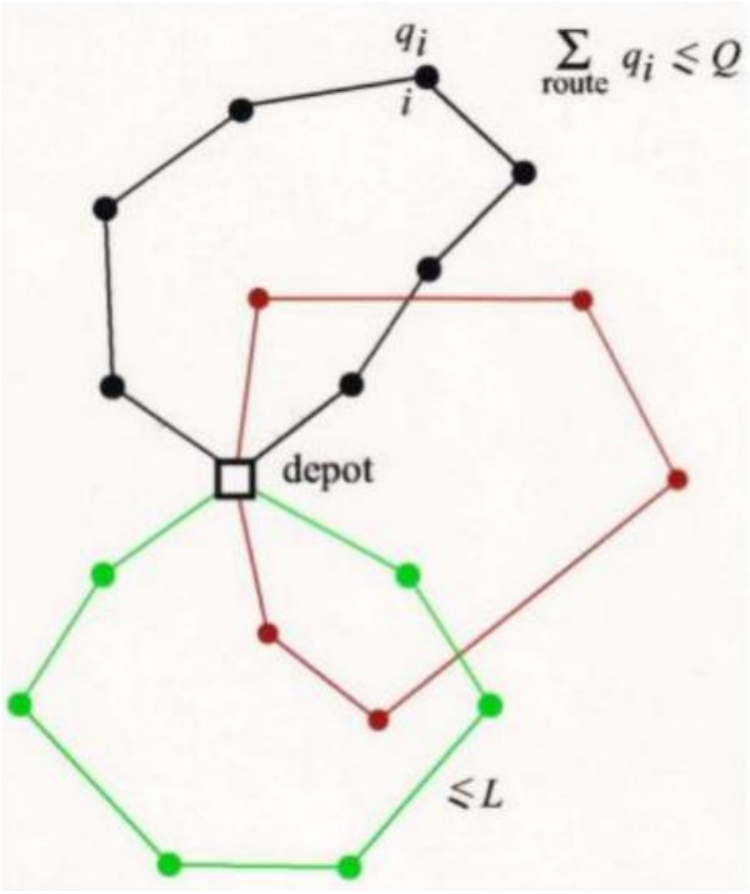

Рис. 10. Граф задачі маршрутизації транспорту

Кожному ребру e ∈ E = {(i, j) : i, j ∈ V, i < j} співвідноситься вартість поїздки c_e або c_{ij}.

На складі є фіксований парк з однакових транспортних засобів, кожен з яких має вантажомісткість Q. ЗМТ потребує визначення набору маршрутів m, вартість яких в цілому зводиться до мінімуму і така, що:

1. кожен клієнт відвідується рівно один раз за один маршрут;
2. кожен маршрут починається і закінчується на складі;
3. загальні потреби клієнтів, які обслуговуються на маршруті, не перевищує пропускну здібність (вантажомісткість) Q;
4. довжина кожного маршруту не перевищує встановлений рівень L.

Зазвичай, прийнято припускати постійну швидкість, для того, щоб відстань, час в дорозі і витрати на проїзд вважались однаковими. Рішення

можна розглядати як набір із m циклів, що розділяють загальну вершину на складі.

Далі слідує цілочислове формулювання лінійного програмування ЗМТ з урахуванням вантажомісткості, де для кожного ребра e ∈ E цілочислова змінна X_e вказує кількість пересічень ребра e в рішенні.

Нехай r(S) є мінімальною кількістю транспортних засобів, необхідних для обслуговування клієнтів підмножини S клієнтів. Значення r(S) можна визначити, вирішивши пов'язану проблему упаковки в ємності (BPP) з набором елементів S і ємністю ємкості Q. Нарешті, для S ⊂ V нехай δ (S) = {(i,j): i ∈ S, j∉ S або i ∉ S, j ∈ S}. Якщо S = {i}, тоді ми просто будемо писати δ(i).

Формула ЗМТУВ, запропонована Laporte et al.(1985):

Minimize $_{e \in E} C_e X_e$, (4)

$_{e \in \delta(i)} X_e = 2, i \in V \setminus \{0\}$ (5)

$_{e \in \delta(0)} Xe = 2m,$ (6)

$_{e \in \delta(S)} X_e >= 2r(S), S \subseteq V\setminus\{0\}, S \neq 0,$ (7)

$X_e \in 0,1, e \notin \delta\ 0,$ (8)

$X_e \in 0,1,2\ e \in \delta\ 0.$ (9)

Обмеження степені (5) вказують, що кожен клієнт відвідується рівно один раз, тоді як обмеження степені депо (6) означає, що m маршрутів створюється.

Обмеження пропускної здібності (7) накладає як зв'язність рішення, так і вимоги до вантажомісткості транспортного засобу, заставляючи достатню кількість ребер ввійти в кожну підмножину вершин.

Обмеження (8) і (9) говорять, що кожне ребро між двома клієнтами проходить не частіше одного разу, як і кожне ребро, з'єднане з депо, проходить не більше двох разів. В останньому випадку автомобіль виконує маршрут, відвідуючи лише одного клієнта.

РОЗДІЛ 2. РОЗГЛЯД АЛГОРИТМІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧ МАРШРУТИЗАЦІЇ ТРАНСПОРТУ

2.1. Види алгоритмів рішення ЗМТ

Як можна побачити в попередньому розділі, різновидів задачі маршрутизації транспорту існує дуже велика кількість. Також існує і велика кількість видів алгоритмів, за допомогою яких можна вирішувати ці задачі. На рисунку 11 представлені види алгоритмів, які використовуються для вирішення ЗМТ, та деякі їх представники.

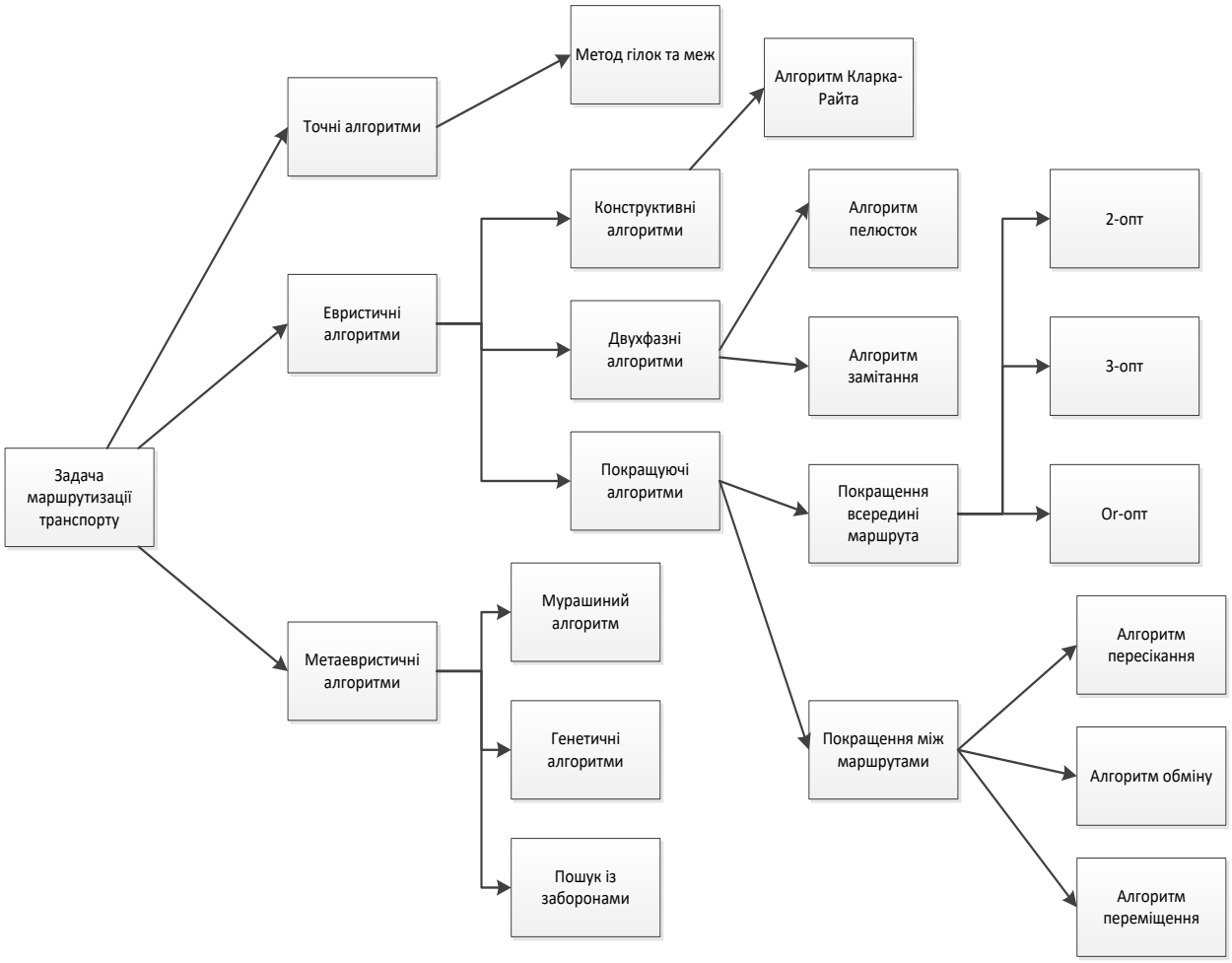

Рис. 11. Алгоритми рішення ЗМТ

Розглянемо детальніше ці алгоритми. Розпочнемо з класу точних алгоритмів.

2.2.Точні алгоритми рішення ЗМТ

Ці алгоритми пропонують знайти всі можливі рішення, доки не буде досягнуто краще з них. Через те, що задача відноситься до класу NP-складної, точні алгоритми можуть використовуватися лише для задач з малою кількістю вхідних даних.

У зв'язку з цим, на практиці частіше використовують інші класи алгоритмів. Розглянемо деякі точні алгоритми.

Графічне представлення методу гілок і меж наведено на рисунку 12.

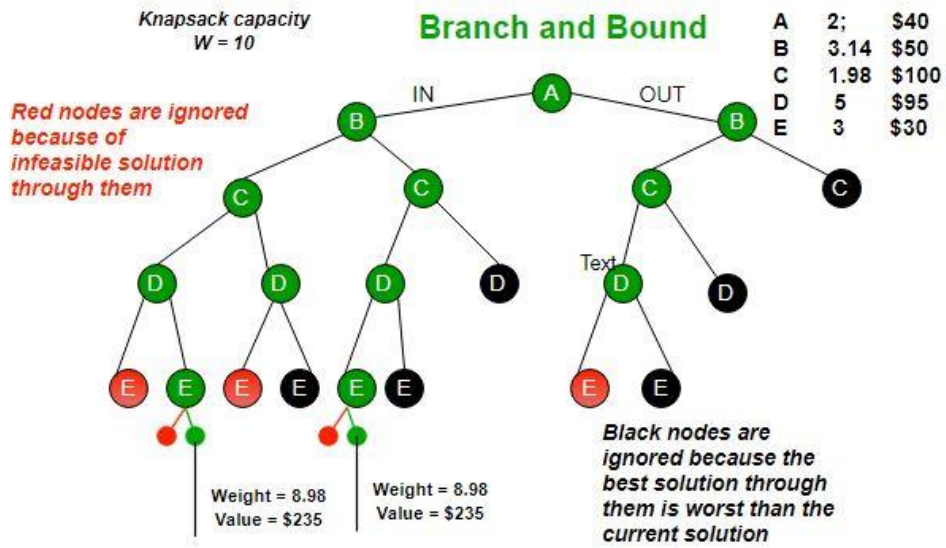

Рис. 12. Метод гілок і меж

Метод гілок і меж має наступні кроки для знаходження оптимального цілочислового рішення:

1) знайти оптимальне рішення моделі лінійного програмування з релаксацією цілих чисел;
2) на вузлу 1 нехай релаксоване рішення являється верхньою межею, а округлене цілочислове рішення — нижньою межею;

3) обрати змінну з найбільшою дробовою часткою для розгалуження. Створити два обмеження для цієї змінної, які відображають значення секційного цілого. Результатом буде нове обмеження типу “≤” і нове обмеження типу “≥”;
4) створити два нових вузла: один для обмеження типу “≤” і один для обмеження типу “≥”;
5) вирішити вільну модель лінійного програмування з новим обмеженням, доданим в кожному з цих вузлів;
6) розслабленим рішенням являється верхня межа в кожному вузлу, а існуюче максимальне цілочислове рішення (на будь-якому вузлу) являється нижньою межею;
7) якщо процес створює допустиме цілочислове рішення з найбільшим значенням верхньої межі будь-якого кінцевого вузла, досягається оптимальне цілочислове рішення. Якщо можливе цілочислове рішення не виникає, переходьте від вузла з найбільшою верхньою межею.
8) Повернутися до кроку 3.

Для моделі мінімізації розслаблені рішення заокруглюються, а верхня і нижня межі змінюються на протилежні.

Другим методом розглянемо метод **гілок і обмежень** [4].

Метод гілок і обмежень — це метод гілок і меж, в якому відсікання генеруються при рішенні релаксійної задачі лінійного програмування у всіх (або тільки деяких) вузлах дерева пошуку.

На відміну від “чистих” методів відсікання, ми тепер не надіємось, що одних обмежень буде достатньо для отримання оптимального рішення.

Метод базується на наступній ідеї. Якщо нижня межа значень функції на частині області A дерева пошуку більше, ніж верхня межа на якій-небудь раніше переглянутій частині області B, то A може бути виключена з подальшого розгляду (правило відсіву). Як правило, мінімальну з отриманих верхніх оцінок записують в перемінну глобального типу з назвою m. Будь-

який вузол дерева рішення, нижня границя якого > ніж m, виключається з розгляду.

Якщо нижня границя для вузла співпадає з верхньою межею, то це значення являється мінімумом функції і досягається на відповідній області.

2.3. Метаевристичні алгоритми рішення ЗМТ

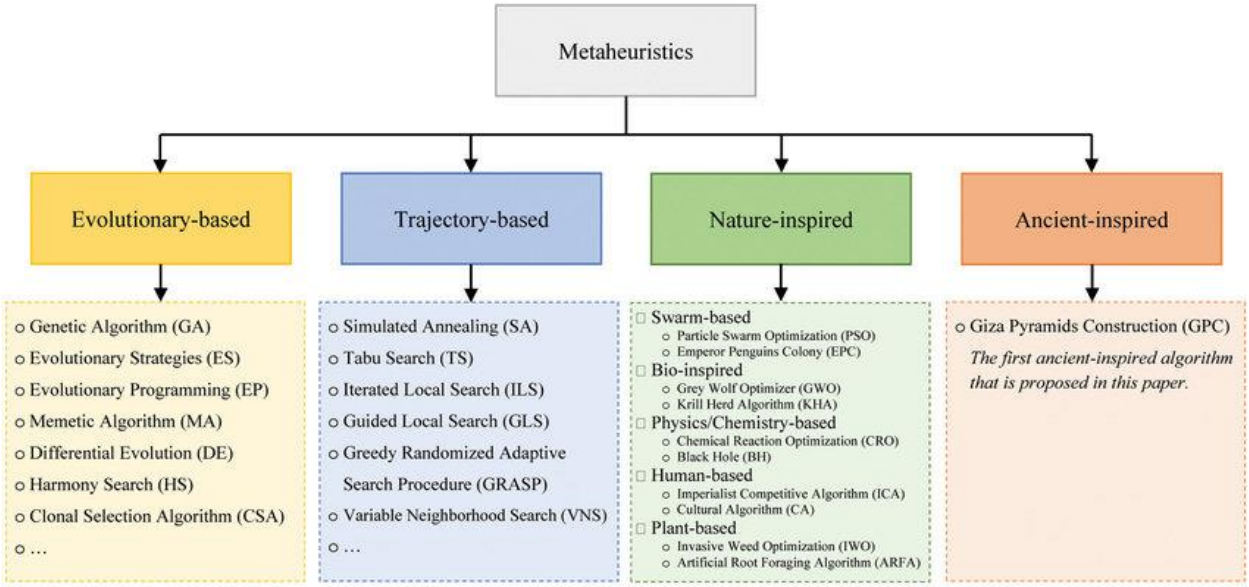

Рис.13. Метаевристичні алгоритми

Метаевристики — досить невдала назва для опису великого підрозділу, в дійсності основного, в стохастичній оптимізації (stochastic optimization).

Стохастична оптимізація є великим класом алгоритмів і методів, які так чи інакше використовують випадковість для пошуку оптимального (або досяжного оптимального) рішення складних завдань. Метаевристики — найзагальніші алгоритми з цього класу і застосовуються для вирішення широкого спектра задач.[5]

Метаевристичні алгоритми, тобто методи оптимізації, розроблені у відповідності до стратегій, викладених в метаевристичний структурі, як слідує з назви, завжди евристичні по своїй природі.

Цей факт відрізняє їх від точних методів, які приходять з доказом того, що оптимальне рішення буде знайдено в скінченний (хоча і часто в непомірно

великий) час. Тому метаевристика розроблена спеціально для того, щоб знайти рішення, яке буде “достатньо хорошим” за час обрахунку, який буде “достатньо малим”. В результаті вони не залежні від комбінаторного вибуху — явища, в якому розрахунковий час, необхідний для знаходження оптимального рішення NP-складних задач, росте як експоненціальна функція розміру задачі.

Наукове суспільство продемонструвало метаевристику як життєздатну і часто більш перспективну альтернативу більш традиційним (точним) методам змішаної цільної оптимізації, таким як галузеве і динамічне програмування. Спеціально для складних проблем, метаевристика часто може запропонувати кращий компроміс між якістю рішення і часом, затраченим на його знаходження. Більш того, метаевристика більш гучка, ніж точні методі, двома важливими аспектами.

По-перше, оскільки метаевристичні рамки визначені в загальних рисах, метаевристичні алгоритми можуть бути адаптовані для задоволення потреб більшості задач оптимізації в реальному часі з точки зору очікуваної якості рішення і дозволеного часу обрахунку, яке може сильно відрізнятися в різних задач і в різних ситуаціях.

По-друге, метаевристика не виставляє ніяких умов до формулювання задачі оптимізації (наприклад, умови обмежень або цільова функція повинна виражатися як лінійні функції змінних рішення).

Проте, ця гнучкість тягне за собою великі затрати на значну адаптацію алгоритмів цього типу до конкретної проблеми для досягнення гарної продуктивності.

Найчастіше використовують такі метаевристичні алгоритми:

1) Мурашиний алгоритм (Ant Algorithm);
2) Імітація відпалу (Deterministic Annealing);
3) Генетичні алгоритми (Genetic Algorithm);
4) Табу-пошук (Tabu Search).

Мурашиний алгоритм (Ant Algorithm)

Мурашиний алгоритм — один з ефективних поліноміальних алгоритмів для знаходження наближених розв'язки задачі комівояжера, а також аналогічних завдань пошуку маршрутів на графах. Підхід запропонований бельгійським дослідником Марко Доріго. Суть підходу полягає в аналізі та використанні моделі поведінки мурах, що шукають шляхи від колонії до їжі.

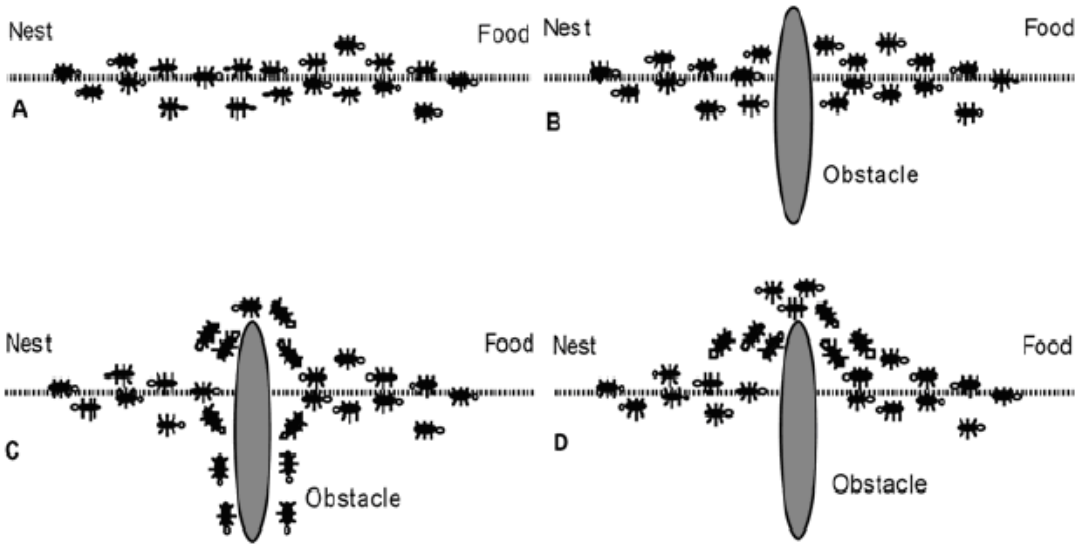

Рис. 14. Мурашина колонія знаходить оптимальних шлях до їжі

У основі алгоритму лежить поведінка мурашиної колонії – маркування вдалих доріг великою кількістю феромону. Робота починається з розміщення мурашок у вершин графу (містах), потім починається рух мурашок – напрям визначається імовірнісним методом, на підставі формули:

$$P_i = \frac{l_i^q * f_i^p}{\sum_{k=0}^{N} l_k^q * f_k^p}$$

де:

P_i — ймовірність переходу шляхом i,

l_i — довжина і-го переходу,

f_i — к-сть феромонів на і-у переході,

q — величина, яка визначає “жадібність” алгоритму,

p — величина, яка визначає “стадність” алгоритму і q + p = 1

Результат не є точним і навіть може бути одним із гірших, проте, в силу ймовірності рішення, повторення алгоритму може видавати (досить) точний результат.

Висхідна діяльність у цьому напрямі призвела до проведення конференцій, присвячених виключно штучним мурахам, а також численним комп'ютерним програмам від спеціалізованих компаній, таких як AntOptima. Як приклад, мурашиний алгоритм є класом алгоритмів оптимізації за зразком колонії мурашок. Штучні «мурашки» знаходять оптимальні варіанти, рухаючись простором параметрів, який представляє усі можливі рішення.

Реальні мурахи виділяють феромон, щоб спрямувати один одного до ресурсів та досліджувати своє оточення. Модельовані «мурахи» аналогічно фіксують свої позиції та якість своїх рішень, таким чином, в подальших ітераціях моделювання мурахи приймають кращі рішення. Одним з варіантів цього підходу є бджолиний алгоритм, який аналогічний структурі роботи медоносних бджіл, іншої соціальної комахи.

В природі, мурахи (початково) блукають довільним чином, і по знаходженні їжі повертаються до колонії, залишаючи по собі слід з феромонів. Якщо інші мурахи знаходять такий шлях, вони схильні припинити свої блукання, натомість слідувати позначеним шляхом, посилюючи його під час повернення у разі знаходження їжі.

Однак, з часом, шляхи з феромонів випаровуються, тоді привабливість шляхів зменшується. Чим більше часу потрібно мурасі, щоб подолати дорогу, тим більше часу мають феромони, щоб випаруватись. Натомість, короткий шлях проходиться частіше, отже щільність феромонів стає більшою на короткому шляху. Випаровування феромонів також надає перевагу уникнення локально найкращих шляхів. Якби випаровування не відбувалось взагалі,

шляхи обрані першою мурахою тяжіли б стати вкрай привабливими для наступних. В цьому разі, розвідка можливих шляхів була б обмежена.

Таким чином, коли мураха знаходить вдалий (тобто короткий) шлях з колонії до джерела їжі, інші мурахи швидше слідуватимуть йому, і позитивний зворотний зв'язок зрештою призведе до обрання цього шляху всіма мурахами.

Ідея мурашиного алгоритму полягає в наслідуванні поведінки з «симулятором мурахи», що прогулюється графом, який представляє проблему, що треба розв'язати (Рис. 15).

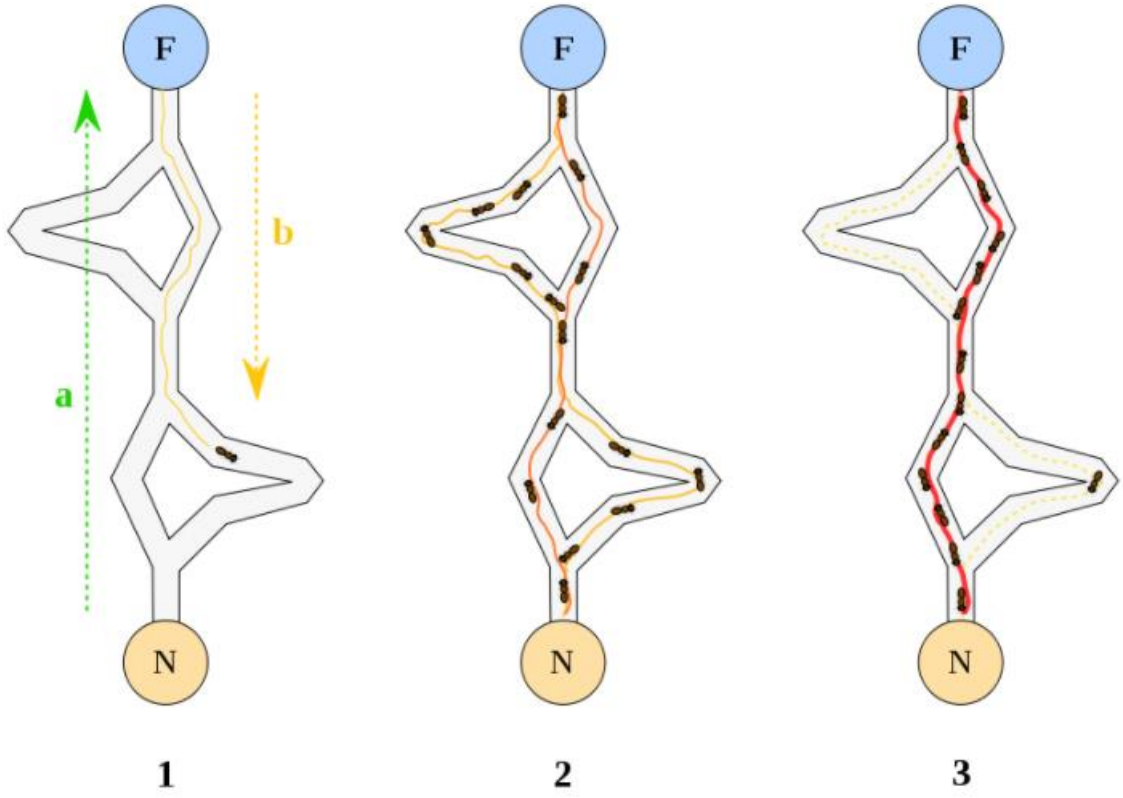

Рис. 15. Схема роботи мурашиного алгоритму

Первісна ідея прийшла зі спостереження за використанням харчових ресурсів серед мурах, де мурахи, окремо обмежені в своїх пізнавальних можливостях, колективно здатні знайти найкоротший шлях між джерелом їжі і гніздом.

1. Перша мураха знаходить джерело їжі (Д), через якийсь шлях (а), тоді повертається до гнізда (Г), залишивши позаду слід з феромонів (б).
2. Мурахи без розбору обирають всі чотири шляхи, але підсилення основної стежки робить її привабливішою як найкоротший шлях.
3. Мурахи обирають коротший шлях, довгі шляхи втрачають щільність сліду з феромонів.

В серії дослідів на колонії мурах з вибором між двома шляхами різної довжини, які ведуть до джерела їжі, біологи спостерігали, що мурахи тяжіють до використання найкоротшого шляху. Модель, що пояснює таку поведінку така:

1. Мураха (звана «бліц») рухається більш-менш випадково по колонії;
2. Якщо вона знаходить джерело їжі, вона повертається прямо до гнізда, залишаючи після себе слід з феромонів;
3. Ці феромони заманливі, ближні мурахи схилятимуться до слідування, більш чи менш точно, цим шляхом;
4. Повертаючись до колонії, мурахи підсилюватимуть маршрут;
5. Якщо наявні два шляхи до одного й того самого джерела їжі тоді, за певний час, коротший шлях пройде більше мурах ніж довший;
6. Короткий шлях все більш посилюватиметься, і таким чином ставатиме привабливішим;
7. Довгий шлях з часом зникне, бо феромони вивітряться;
8. Зрештою, всі мурахи це визначатимуть і через це обиратимуть найкоротший шлях.

Мурахи використовують навколишнє середовище як посередник для зв'язку. Вони обмінюються інформацією непрямо, через відкладання феромонів, які уточнюють статус їхньої роботи. Інформація розповсюджувана через феромони має місцеву дію, лише мурахи розташовані поруч із відкладеними феромонами помічають їх. Така система трапляється в багатьох спільнотах соціальних тварин (її вивчали на прикладі розбудови стовпів у гніздах термітів).

Спільне розв'язання проблем занадто складних для однієї мурахи є хорошим прикладом власно організованої системи. Система покладається на позитивний зворотний зв'язок (відкладення феромонів приваблює інших мурах, які підсилять їх у свою чергу) і негативний (зникнення маршруту через випаровування забезпечує оптимальність роботи системи). Теоретично, якщо щільність феромонів залишатиметься постійною на всіх відтинках, жоден із шляхів не стане головним.

Однак, через зворотний зв'язок, малі відмінності на частинах маршрутах підсилюватимуть силу феромонів і дозволятимуть зробити вибір. Алгоритм зсуватиметься з нестабільного стану, де жоден з частин маршруту не привабливіший за інші, у бік стабільного стану, де маршрут утворений з найкращих частин маршрутів [6].

Генетичні алгоритми (Genetic Algorithm)

Генетичний алгоритм — це еволюційний алгоритм пошуку, що застосовується для вирішення завдань оптимізації і моделювання за допомогою послідовного підбору, комбінування і зміни критеріїв, які знаходяться з використанням патернів, подібних до біологічної еволюції.

Особливістю генетичного алгоритму є акцент на використання оператора «схрещення», який виконує операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещення в живій природі. «Батьком-засновником» генетичних алгоритмів вважається Джон Голланд, книга якого «Адаптація в природних і штучних системах» є фундаментальною в цій сфері досліджень [7].

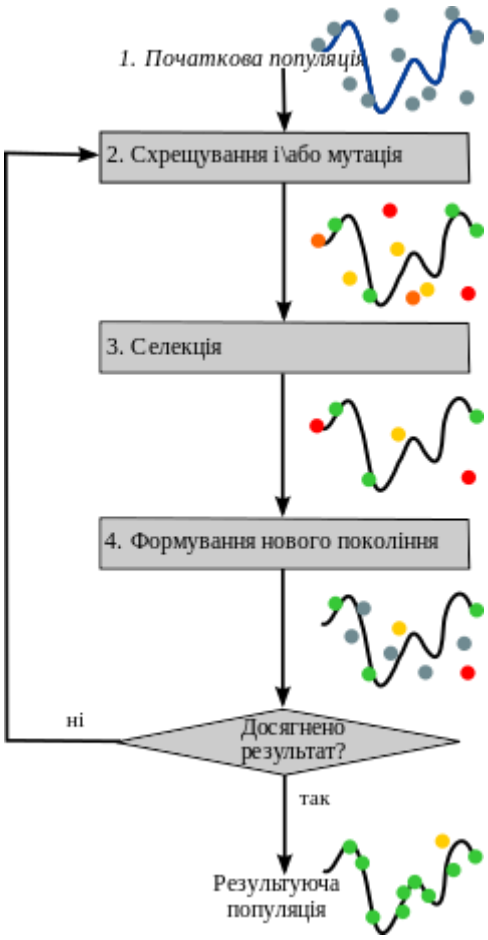

Рис. 16. Генетичний алгоритм

Задача кодується таким чином, щоб її вирішення могло бути представлено у вигляді масиву подібного до інформації складу хромосоми [7].

Цей масив часто саме так і називають «хромосома». Випадковим чином в масиві створюється деяка кількість початкових елементів «осіб», або початкова популяція. Особи оцінюються з використанням функції придатності, в результаті якої кожній особі присвоюється певне значення придатності, яке визначає можливість виживання особи. Після цього з використанням отриманих значень придатності обираються особи, допущені до схрещення (*селекція*) [7].

До осіб застосовується «генетичні оператори» (в більшості випадків це оператор схрещення (crossover) і оператор мутації (mutation)), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і

мутація. Так моделюється еволюційний процес, що продовжується декілька ітерацій, поки не буде досягнуто стоп-критерію алгоритму. Ним може бути [7]:

- знаходження глобального, або найкращого вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Генетичні алгоритми можуть використовувати для пошуку рішень в дуже великих і важких просторах пошуку [7].

Можна виділити такі етапи генетичного алгоритму [7]:

1. Створення початкової популяції:
2. Обчислення функції придатності для осіб популяції (оцінювання);
3. Повторювання до виконання критерію зупинки алгоритму:
    1. Вибір індивідів із поточної популяції (селекція)
    2. Схрещення або/та мутація
    3. Обчислення функції допасованості для всіх осіб
    4. Формування нового покоління

Створення початкової популяції

Перед першим кроком необхідно випадковим чином створити деяку початкову популяцію. Навіть якщо популяція виявиться абсолютно неконкурентоздатною, генетичний алгоритм все одно достатньо швидко переведе її в придатну для життя популяцію. Таким чином, на першому кроці можна не старатися зробити надто допасованих осіб, достатньо, щоб вони відповідали формату осіб популяції, і на них можна було порахувати функцію придатності. Наслідком першого кроку є популяція H, що налічує N осіб [7].

Відбір

На етапі відбору необхідно із всієї популяції вибрати її певну долю, яка залишиться в «живих» на цьому етапі популяції. Є декілька способів провести відбір. Ймовірність виживання особи h повинна залежати від значення її придатності Fitness(h). Сама ж доля відібраних s зазвичай є параметром генетичного алгоритму, і її просто задають заздалегідь. Внаслідок відбору із N

осіб популяції H повинні залишитись sN осіб, які ввійдуть в наступну популяцію H'. Решта осіб «загине» [7].

Розмноження

Розмноження в генетичних алгоритмах зазвичай статеве – щоб «народити» нащадку, необхідно декілька батьків, зазвичай потрібна участь двох. Розмноження в різних алгоритмах описується по різному – воно, звісно, залежить від формату осіб. Головна вимога до розмноження – щоб нащадок чи нащадки мали можливість успадкувати риси всіх батьків, «змішавши» їх якимось достатньо розумним чином.

Розмноження або оператор рекомбінації застосовують відразу ж після оператора відбору батьків для отримання нових особин-нащадків. Сенс рекомбінації полягає в тому, що створені нащадки повинні наслідувати генну інформацію від обох батьків. Розрізняють дискретну рекомбінацію і кросинговер.

Приклад операції розмноження: Вибрати (1-s)p/2 пар гіпотез із H і провести з ними розмноження, отримавши по два нащадки від кожної пари (якщо розмноження описано так, щоб давати одного нащадку, необхідно обрати (1–s)p пар), і додати цих нащадків в H'. В результаті H' буде складатися з N осіб [7].

Особи для розмноження зазвичай вибираються із всієї популяції H, а не із тих, що вижили на першому кроці (хоча останній варіант теж має право на існування). Справа в тому, що головна проблема генетичних алгоритмів – недостача різноманітності (diversity) в особах. Достатньо швидко виділяється єдиний генотип, який являє собою локальний максимум і згодом всі елементи популяції програють йому в відборі, і вся популяція «забивається» копіями цієї особи. Існують різні способи боротьби із таким небажаним ефектом; один з них – вибір для розмноження не з самих «допасованих», а взагалі зі всіх осіб [7].

Мутації

До мутацій відноситься все те ж, що і до розмноження: є деяка доля мутантів m, що є параметром генетичного алгоритму, і на кроці мутацій необхідно вибрати mN осіб, а згодом змінити їх згідно з заздалегідь заданими операціями мутації [7].

Імітація відпалу (Deterministic Annealing)

Алгоритм імітації - загальний алгоритмічний метод розв'язання задачі глобальної оптимізації, особливо дискретної та комбінаторної оптимізації, в якому процедура пошуку глобального розв'язку імітує фізичний процес відпалу. Він часто використовується, коли пошук наближеного глобального оптимуму важливіший, ніж пошук точного локального оптимуму за встановлений проміжок часу [8].

Графічне представлення процесу відпалу наведено на рисунку 17.

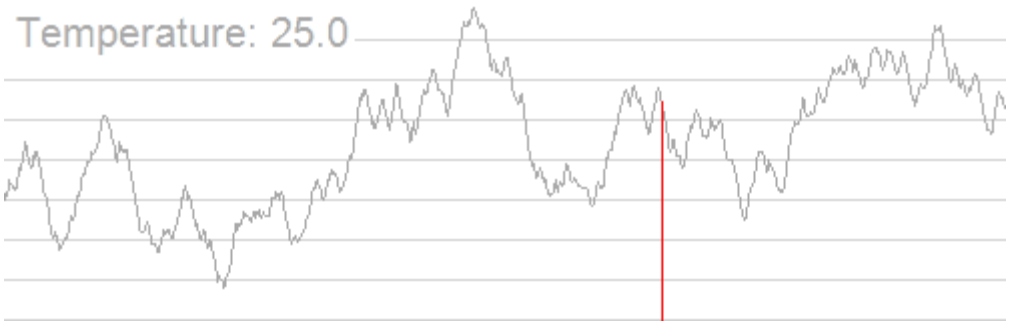

Рис. 17. Імітація відпалу

Алгоритм імітації відпалу — загальний алгоритмічний метод розв'язання задачі глобальної оптимізації, особливо дискретної та комбінаторної оптимізації. У галузі фізики конденсованих середовищ, *відпалом* називається тепловий процес отримання низьких енергетичних станів тіла в тепловій бані (термостаті). Цей процес складається з двох кроків (Kirkpatrick та ін., 1983) [8]:

- підвищення температури теплової бані до максимального значення, до твердих розплавів
- повільне зниження температури теплової бані, поки частинки не впорядкуються в основний стан тіла.

У рідкій фазі частинки розташовуються випадковим чином, а в основному стані твердого тіла всі частинки розташовані в добре структуровані ґратки, для яких відповідна енергія мінімальна. Основний стан твердого тіла отримується тільки тоді, коли максимальне значення температури досить високе і охолодження здійснюється досить повільно. В іншому випадку, тіло застигне в метастабільному стані, а не в істинному стані [8].

Алгоритм ґрунтується на імітації фізичного процесу, який відбувається при кристалізації речовини з рідкого стану в твердий, у тому числі при відпалі металів. Передбачається, що атоми вже вишикувалися в кристалічну решітку, але ще допустимі переходи окремих атомів з однієї комірки в іншу. Передбачається, що процес протікає при поступовому зниженні температури. Перехід атома з однієї комірки в іншу відбувається з деякою ймовірністю, причому вірогідність зменшується з пониженням температури. Стійка кристалічна решітка відповідає мінімуму енергії атомів, тому атом або переходить в стан з меншим рівнем енергії, або залишається на місці. (Цей алгоритм також називається *алгоритмом Метрополіса*, за ім'ям його автора Ніколаса Метрополіса) [8].

Повертаючись до імітації відпалу, *алгоритм Metropolis* може бути використаний для генерування послідовності рішень задачі комбінаторної оптимізації, припускаючи еквівалентності між фізичною системою багатьох частинок і завданням комбінаторної оптимізації [8]:

- розв'язання задачі комбінаторної оптимізації еквівалентні стану фізичної системи
- вартість рішення еквівалентно енергії стану системи.

Крім того, ми вводимо керуючий параметр, який відіграє роль температури. Таким чином Simulated annealing алгоритм, можна розглядати як

ітерації алгоритму Metropolis, виконані на зменшення значення керуючого параметра. Значення чотирьох функцій в процедурі SIMULATED_ANNEALING очевидні:

- INITIALIZE обчислює і задає початкові значення параметрам c та L;
- GENERATE вибирає сусідій розв'язок відносно поточного розв'язку;
- CALCULATE.LENGTH і CALCULATE_CONTROL обчислює нові значення параметрів L та c відповідно.

Порівнюючи simulated annealing з ітераційним поліпшенням, очевидно що simulated annealing можна розглядати як узагальнення. Алгоритм імітації відпалу стає ідентичним ітераційному покращенню в разі, коли значення керуючого параметра приймається як нуль. Що стосується порівняння продуктивність обох алгоритмів, для більшості завдань simulated annealing працює швидше, ніж ітераційне покращення [8].

Підсумковий час реалізації моделювання відпалу виходить шляхом створення послідовності однорідних ланцюжків Маркова обмеженої довжини при низхідному значенні керуючого параметра. Для цього набору параметрів повинно бути зазначено, що визначає збіжність алгоритму. Ці параметри об'єднані в так званий графік охолодження. Графік охолодження визначає скінченну послідовність значень параметра, і скінченне число переходів при кожному значенні керуючого параметра. Точніше, він визначає [8]:

• початкове значення параметра контролю c_0;

• зменшення функції для зниження вартості контрольного параметра;

• кінцеве значення контрольного параметра, вказаний критерій зупинки, також скінченне значення довжини кожного однорідного ланцюжка Маркова.

Статичний графік охолодження. Це простий графік, який також називається геометричним графіком. Він походить ще з початку роботи над графіком охолодження. Він і досі використовується в багатьох практичних ситуаціях. Остаточне значення контрольного параметра, тут остаточна величина фіксується і отримує невелике значення, яке може бути пов'язане з найменш можливою різницею у вартості між двома сусідніми розв'язками.

З моменту своєї появи в 1983 році моделювання відпалу було застосоване до досить великої кількості різних проблем у різних областях [8]. Більше 20 років досвіду привели такі загальні спостереження:

- Висока якість розв’язку може бути отриманий, але іноді за рахунок великих обсягів обчислень;
- У багатьох практичних ситуаціях, де наявні алгоритми не працюють, моделювання відпалу має великі плюси: загальність застосування і простота реалізації.

Таким чином, моделювання відпалу є алгоритмом, який всі практики математики та науковці комп'ютерних наук повинні мати у своєму інструментарії. [8]

Табу пошук (Tabu Search)

Табу-пошук (ТП) є мета-евристичним алгоритмом, який веде локальний пошук, щоб запобігти його від попадання в пастку в передчасних локальних оптимумах, забороняючи ті переміщення, які змушують повертатися до попередніх рішень і циклічної роботи. ТП починається з вихідного рішення. На кожній ітерації генерується окіл рішень, і найкраще з цього околу вибирається як нове рішення. Певні атрибути попередніх рішень зберігаються в табу-списку, який оновлюється в кінці кожної ітерації. Вибір найкращого рішення в околі відбувається таким чином, що він не приймає жодного з заборонених атрибутів. Найкраще допустиме рішення в даний час, оновлюється, якщо нове поточне рішення краще і допустиме. Процедура триває, поки не виконається будь-який з двох критеріїв зупину, якими є максимальне число виконуваних ітерацій і максимальне число ітерацій, під час яких чинне рішення не поліпшується [9].

Алгоритм ТП:

1. Обираємо початкове рішення x, вважаємо $F_{PZ} = F(x)$. Формуємо порожній список заборон.
2. Знаходимо нове рішення z таке, що:

А) $z \in N'(x)$

Б) $F(z) = \min\{F(y) \mid y \in N'(x)\}$

В) перехід $x \rightarrow z$ не є забороненим або $F(z) < Z_{PZ}$

3. Переходимо в нову точку $x = z$

   Змінюємо список заборон.

   Якщо $F(x) < F_{PZ}$, то змінюємо рекорд $F_{PZ} = F(x)$.

4. Якщо виконаний критерій зупинки, то алгоритм припиняє роботу, інакше повернутися до п. 2.

Як критерій зупинки використовується або зупинка по числу ітерацій, або необхідна точність по відношенню до заданої нижньої межі. Початкове рішення вибирається за допомогою якогось простого алгоритму. Змінюючи довжину списку заборон, можна керувати процесом пошуку. При зменшенні довжини, інтенсифікується пошук в поточній області, збільшення довжини сприяє переходу до іншої області [9].

Як околиці можна розглядати безліч всіх рішень, які виходять з поточного рішення зміною однією з координат. У 1992 році Файгл і Керн запропонували імовірнісну версію методу TS, який при кількості ітерацій прагнучих до нескінченності, сходиться до глобального оптимума, що, втім, властива більшості імовірнісних алгоритмів. Так випадково згенерована околиця скорочує трудомісткість алгоритму на кроці 2 і вносить елемент випадковості при виборі напрямку спуску. Якщо випадково згенерована околиця становить від 1 до 10 відсотків від початкової, то алгоритм не зациклюється навіть без переліку заборон.

Алгоритм TS застосовувався до широкого кола завдань дискретної оптимізації, показав високу працездатність і на сьогодні є однією з найпопулярніших імовірнісних евристик. Одне з можливих застосувань методу «Пошук з заборонами» для вирішення задач нерегулярного розкрою – упаковки (Р-У) запропоновано в роботі польськими авторами Блазевічем, Хаврулюком і Валковяком [9].

Задача комівояжера часто використовується, щоб показати функціональність табу-пошуку [9].

Задача комівояжера полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед Гамільтонових циклів [9].

Наприклад, якщо місто **A** і місто **B** розташовані поруч одне з одним, в той час як місто **C** розташоване далі, загальна пройдена відстань, буде коротшою, якщо міста А і Б відвідали одне за одним перед приїздом до міста С. Оскільки знаходження оптимального рішення до задачі комівояжера є NP-повним завданням, евристичні наближені методи (наприклад, локальний пошук) корисні для розробки близьких до оптимальних рішень.

Табу-пошук може бути використаний для пошуку **задовільного** рішення для задачі комівояжера (тобто, рішення, яке задовольняє критерію адекватності, а не абсолютно оптимальне рішення).

По-перше, пошук із заборонами починається з початкового рішення, яке може бути випадково згенерованим, або за допомогою якогось алгоритму найближчого сусіда. Для створення нового рішення, два міста, які відвідують в потенційному рішення, змінюються місцями. Загальна відстань подорожі між усіма містами використовується для оцінки ідеального рішення, в порівнянні з іншим. Для запобігання циклів і, щоб не застрягти в локальних оптимумах, рішення буде додано до переліку табу, якщо воно буде задовольняти рішення в околиці.

Нові рішення продовжують створюватися до появи деяких стоп-критеріїв, таких як довільне число ітерацій. Коли табу-пошук припиняється, повертається краще рішенням — рішення з найкоротшою відстанню, під час відвідування всіх міст. [9]

2.4. Евристичні алгоритми рішення ЗМТ

Евристичні алгоритми — це алгоритми побудовані на деякому правилі (евристиці), яке не завжди має строгу математичну основу, а також забезпечує в більшості випадків рішення наближене до точного.

В інформатиці евристичний алгоритм, або просто евристика – це алгоритм, спроможний видати прийнятне рішення проблеми серед багатьох рішень, але неспроможний гарантувати, що це рішення буде найкращим [10-11]. Отже, такі алгоритми є приблизними і неточними. Зазвичай такі алгоритми знаходять рішення, близьке до найкращого і роблять це швидко. Іноді такі алгоритми можуть бути точними, тобто вони знаходять дійсно найкраще рішення, але вони все одно будуть називатися евристичними до тих пір доки не буде доведено, що рішення дійсно найкраще. Один з найвідоміших – жадібний алгоритм, для того, щоб бути простим і швидким, цей алгоритм ігнорує деякі вимоги задачі [10-11].

Дві фундаментальні цілі в інформатиці – знаходження алгоритмів з імовірно найкращим часом виконання та з хорошою або оптимальною якістю.

Евристичний алгоритм відмовляється від однієї або обох цих цілей. Наприклад, він зазвичай знаходить дуже гарне рішення, але немає доказів, що рішення насправді не є поганим. Або він працює досить швидко, але не має гарантії, що він завжди знайде оптимальне рішення.

Декілька евристичних методів використовуються антивірусним ПЗ для виявлення вірусів та іншого шкідливого ПЗ.

Часто можна знайти таку задачу, в якій евристичний алгоритм буде працювати або дуже довго, або видавати невірні результати, однак, такі парадоксальні приклади можуть ніколи не зустрітись на практиці через свою специфічну структуру. Таким чином, використання евристики дуже поширене в реальному світі. Для багатьох практичних проблем евристичні алгоритми, можливо, єдиний шлях для отримання задовільного рішення в прийнятний проміжок часу [10].

Це алгоритми, що мають такі властивості [10]:

- Вони дозволяють знайти гарні, хоча і не завжди найкращі розв'язки з усіх, що існують.
- Метод пошуку або побудови розв'язку звичайно значно простіший, ніж той що гарантує оптимальність розв'язку.

Поняття "добрий розв'язок" змінюється від задачі до задачі, тому його важко визначити точно. Припустимість використання евристики залежить від співвідношення часу та складності пошуку розв'язку обома способами та співвідношення якості обох результатів розв’язку [10-11].

У цьому розумінні усі умови, що висуваються до розв'язку, зазвичай ділять на дві групи щодо витрат праці:

1. ті, які легко задовольнити;
2. ті, що вимагають великої роботи.

З іншої сторони, вони поділяються на такі групи щодо їхньої важливості для кінцевої якості:

a. ті, які обов'язково слід задовільнити;
b. ті, що можуть бути послаблені або змінені.

Цю ситуацію образно показано на рисунку 18 [11].

	1	2
a	Слід задовільнити	
b		Варто відмовитися

Рис. 18. Важливість рішень

Евристичні алгоритми діляться на:

1) двофазні алгоритми;
2) конструктивні алгоритми;
3) покращуючи алгоритми;

Конструктивні алгоритми

Це алгоритми, які крок за кроком вибудовують рішення, враховуючи вартість, яка отримується в ході рішення [12].

Конструктивні алгоритми це різновид евристичних алгоритмів, які розпочинаються з пустого рішення і багатокрокова продовжують побудову рішення, доки не буде отримано повне рішення [12].

Вони відрізняються від евристичних алгоритмів локального пошуку, які розпочинаються з повного рішення, а потім удосконалюють його за допомогою локальних ходів [12].

Прикладами деяких відомих проблем, які вирішуються за допомогою конструктивної евристики є: планування потоку магазинів, *проблема маршрутизації транспорту,* і проблема відкритого магазину [12].

Алгоритм Кларка-Райта

Алгоритм Кларка-Райта є одним з представників конструктивних евристичних алгоритмів [13]. Він є одним з простих евристик. Алгоритм має наступний алгоритм [13]:

1) зробити n маршрутів: $v_0 \rightarrow v_i \rightarrow v_0$, для кожного i >= 1;
2) вирахувати економію злиття місць доставки i & j, яка задається $s_{ij} = d_{i0} + d_{oj} - d_{ij}$, для всіх i,j >= 1 & i ≠ j;
3) відсортувати в порядку спадання збереження;
4) починаючи з верхнього списку збереження, об’єднайте два пов’язаних маршрутів з найбільшою економією при умові, що:
    - два місця доставки не знаходяться на тому ж маршруті;
    - ні одне місце доставки не являється внутрішнім по відношенню до його маршруту, що означає, що обидва запити напряму пов’язані з депо на відповідних маршрутах;
5. Повторювати крок 3, доки не буде досягнута додаткова економія.

На рисунках 19-20 продемонстрована робота алгоритму.

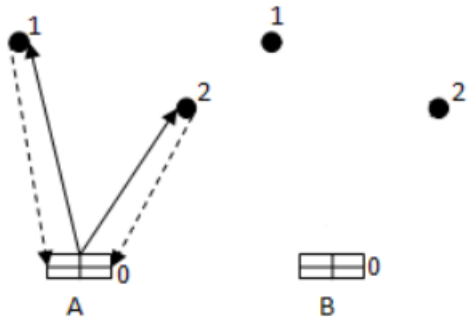

Рис. 19. Побудова маршруту для кожного клієнта

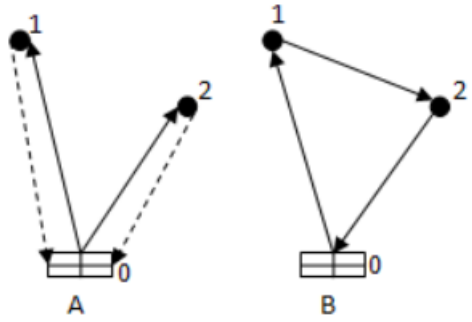

Рис. 20. Об’єднання маршрутів

Двофазні алгоритми

Задача в двофазному алгоритмі ділиться на 2 частини.

1) Збір вершин в групи;
2) Побудова маршруту для кожної групи.

Алгоритм пелюсток

Природнім розширенням алгоритму розгортання є створення декількох маршрутів, які називаються пелюстками, і кінцевий вибір шляхом рішення задачі розбиття форми [14].

Мінімізуємо $_{k \in S} d_k x_k$,

при умові

$_{k \in S} d_k x_k = 1$,

де i = 1, …, nm $x_k = 0$ or 1,

$k \in S$, S – множина маршрутів,

$x_k = 1$ тоді і тільки тоді, коли маршрут k належить рішенню,

a_{ik} – двійковий параметр, рівний 1, тільки якщо вершина і належить маршруту k, а d_k –вартість пелюстка k.

Якщо маршрути відповідають суміжним секторам вершин, то ця проблема має властивість заокруглення і може бути вирішена в поліноміальний час.

Алгоритм замітання

Так як в традиційній задачі маршрутизації транспорту з урахуванням вантажомісткості (ЗМТУВ) вузли оточують центральне депо, алгоритм замітання застосовується таким чином як показано на рисунку 21 [15].

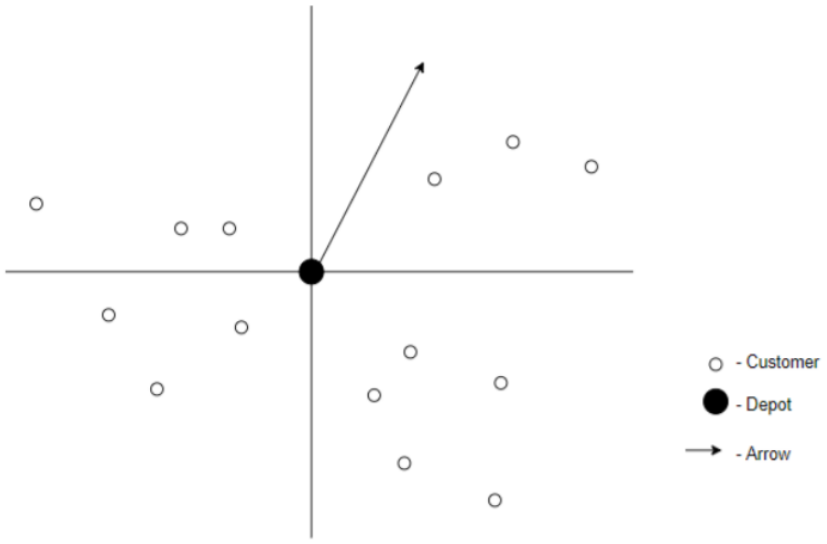

Рис. 21. Алгоритм замітання

1) радіальна лінія з центральним депо в якості точки розпочинається з 0^0 і проходить по вузлам, оточуючим депо, по часовій або проти часової стрілки;
2) коли перший вузол зустрічається, він присвоюється першому транспортному засобу і перевіряється на обмеження пропускної здібності. Якщо він проходить, клієнт вважається відвіданим;

3) далі лінія знову прокручується для отримання другого клієнта;
4) коли другий клієнт назначений, спочатку застосовується обмеження пропускної здібності, і якщо воно виконується, клієнт назначається транспортному засобу, і клієнт також вважається “відвіданим”;
5) “підмітання” продовжується до тих пір, поки місткість ТЗ не буде заповнена, або не будуть перевищені інші обмеження;
6) потім процедура починається з ТЗ номер 2 для вузлів що залишились;
7) повторення процедури триває до тих пір, доки всім вузлам не будуть назначені транспортні засоби.

Алгоритм Фішера Джекумера

Використовує для формування кластерів узагальнену задачу про призначення. Вважається, що кількість транспортних засобів k в цьому алгоритмі задано [16]. Робота алгоритму наступна [16]:

1) відбувається вибір по одній вершині j_k з множини всіх вершин V для початкового заповнення кожного кластера k;
2) далі обраховується вартість d_{ik} призначення кожної вершини і кожному кластеру k як $d_{ijk} = \min c_{0i} + c_{ijk} + c_{ik}^{0}, c_{0jk} + c_{jk}^{i} + c_{i0} – (c_{ijk} + c_{jk0}) = \min \{c0i + cijk + cjk0, c0jk + cjki + ci0\} – (c0jk + cjk0)$;
3) вирішується узагальнена задача про призначення з вартостями d_{ij}, з урахуванням ваги клієнтів q_i і вантажомісткістю ТЗ Q;
4) в кінці вирішується задача комівояжера для кожного кластера на основі результатів, знайдених за допомогою узагальненої задачі про призначення.

Покращуючи алгоритми

Спочатку відбувається формування допустимого рішення(будь-якого), а далі покращення його шляхом застосування послідовних невеликих змін.

Основні серед таких алгоритмів є:

1. Покращення всередині маршруту (intra-route):

- 2-опт (2-opt);
- 3-опт (3-opt);
- Or-опт (Or-Opt).

2. Покращення між маршрутами (inter-route):

- пересічення (Cross);
- обмін (Exchange);
- переміщення (Relocation).

Покращуючи всередині маршруту алгоритми для ЗМТ описані в термінах λ-опт операцій, які в свою чергу були викладені Ліном [17]. В цих алгоритмах видаляється λ ребер з маршруту, і відбувається з'єднання λ сегментів, що залишилися в усіх комбінаціях. І якщо перше досягнуте покращуючи з'єднання (найбільш успішне) знайдено, то нові зміни вносяться до маршруту. Якщо більше неможливо знайти підходящий варіант замін, то робота закінчується. Для перевірки λ-оптимальності рішення необхідно $O(n^{\lambda})$ [17].

Алгоритм 2-опт

2-опт алгоритм був запропонований Croes в 1958 році для вирішення задачі комівояжера [18]. Основна ідея полягає в тому, щоб пройти маршрут, як показано на рисунку 22, який пересікає себе і впорядковує його так, щоб він цього не робив [18].

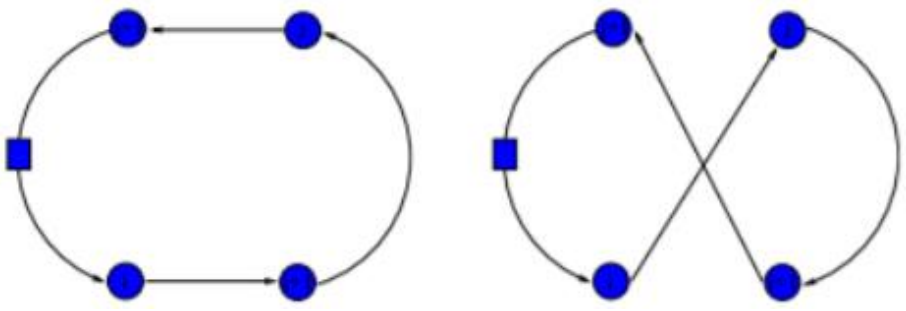

Рис. 22. Схема 2-опт алгоритм

Алгоритм 3-опт

3-опт алгоритм, представлений на рисунку 23, включає видалення 3-ох з'єднань в середині маршруту для створення 3-х частинах маршрутів.

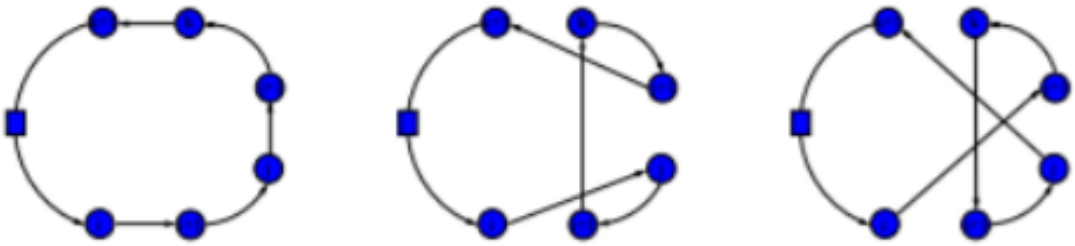

Рис. 23. Схема 3-опт алгоритм

Далі аналізується 7 різних способів повторити варіантів маршруту, щоб знайти оптимальний. Після цього, цей процес повторюється для іншого набору з 3-х підключень, поки всі можливі комбінації не будуть перевірені в маршруті. Він має часову складність O n^3.

Алгоритм Or-опт

Названий в честь Ilhan Or (1976) [18]. Or-otp алгоритм зображений на рисунку 24.

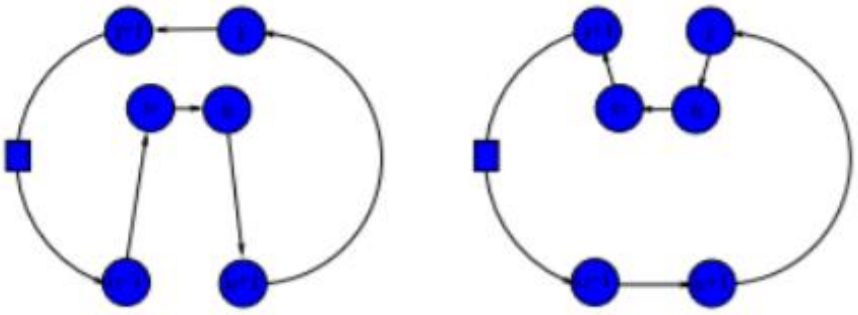

Рис. 24. Схема Or-опт алгоритм

Кроки виконання алгоритму:

1) переміщувати ланцюги з трьох послідовних вершин у всіх можливих варіаціях;

2) застосувати будь-який вигідний крок;
3) повторювати до тих пір, доки це вигідно;
4) почати переміщувати ланцюги з 2-ма вершинами і, нарешті, з єдиною вершиною.

Покращуючи маршрутні алгоритми коротко описуються в роботі [19].

Оператор переміщення, представлений на рисунку 25, переміщує одного клієнта в інший маршрут.

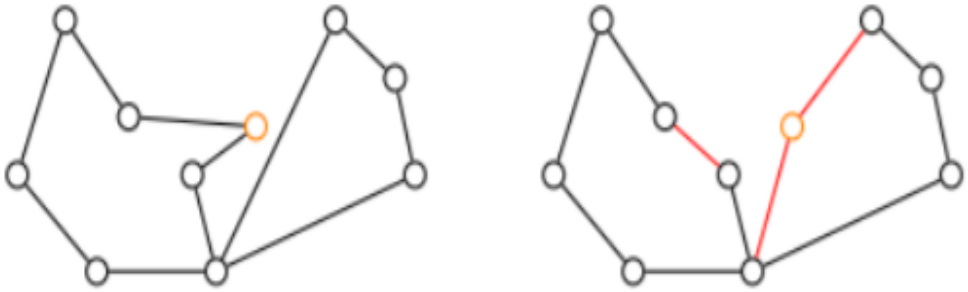

Рис. 25. Зразок алгоритму переміщення

Оператор обміну, зображений на рисунку 26, міняє двох клієнтів в двох різних маршрутах, зберігаючи їх позиції в маршрутах.

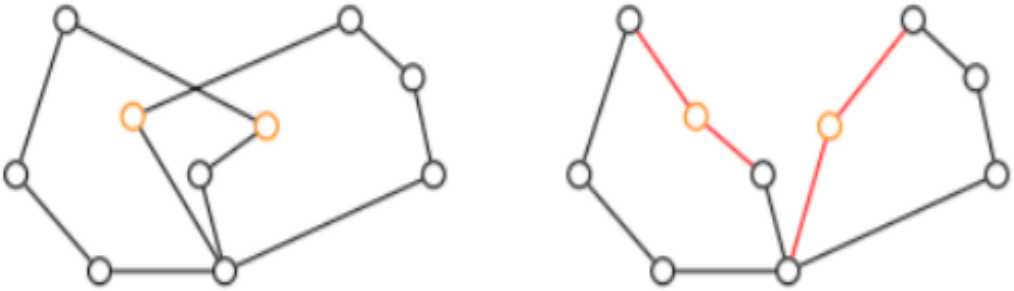

Рис. 26. Алгоритм обміну

2.5. Обрання алгоритму для реалізації

Необхідно обрати алгоритм для реалізації. В цьому розділі були розглянуті всі види алгоритмів для вирішення задачі маршрутизації транспорту.

Для обрання ефективного алгоритму висуваються наступні критерії:

1) ефективність;
2) простота реалізації;
3) допустимий час роботи.

На основі даних, отриманих з цього розділу сформована таблиця, яка дозволить обрати вид алгоритму.

Таблиця 1. Порівняння класів алгоритмів

	Ефективність	Простота реалізації	Допустимий час роботи
Точні	-	+	-
Евристичні	+	+	+
Метаевристичні	+	-	+

Точні алгоритми прості в реалізації, але їх можна використовувати тільки в задачах, коли вхідні дані мають невеликі розміри, інакше час роботи алгоритму занадто великий. Метаевристичні алгоритми чудово підходять для вирішення ЗМТ, вирішують її ефективно і за допустимий час роботи, але складні в реалізації, і потребують багато ресурсів для адаптації алгоритмів до конкретної задачі [9-11].

Виходячи з цього, потрібно обрати один з евристичних алгоритмів, яких є досить багато, і які досить добре підходять для вирішення ЗМТ, при цьому не є такими складними в реалізації як метаевристичні алгоритми.

На основі аналізу евристичних алгоритмам, що базується на різних дослідженнях і статтях, які використовувалися в цій роботі, був зроблений висновок, що для рішення задачі маршрутизації транспорту, частіше всього використовують двофазні і конструктивні алгоритми.

В таблиці 2 наведено порівняння аналізу за критерієм асимптоматичної обчислювальної складності покращуючих евристичних алгоритмів. Перед тим як зробити висновок по цій таблиці, дамо визначення асимптоматичної обчислювальній складності.

Таблиця 2. Асимптоматична обчислювальна складність алгоритмів

Евристичні алгоритми покращення	Асимптоматична обчислювальна складність
Relocation	$O(n^2)$
Exchange	$O(n^2)$
Cross	$O(n^2)$
2-opt	$O(n^2)$
3-opt	$O(n^2)$
Or-opt	$O(n^3)$

В теорії складності обчислень, асимптоматична обчислювальна складність використовує асимптоматичний аналіз для оцінки обчислювальної складності алгоритмів і задач обчислення, зазвичай пов’язаних з використанням у великій нотації О.

Що стосується обчислювальних ресурсів, асимптоматична складність часу і асимптоматичний простір складності зазвичай оцінюються. Друга асимптоматична оцінювана поведінка включає складність схеми і різні виміри паралельних обчислень, такі як кількість (паралельних) процесорів.

Так як в оригінальній роботі 1965 року Юріса Хартманіса і Річадра Е. Стернса [20], а також книги Майкла Гарей і Девіда С. Джонсона від 1979 року про NP-складність [1], термін “обчислювальна складність” алгоритмів став симптоматичною обчислювальною складністю.

Окрім того, якщо не вказано інше, термін “обчислювальна складність” зазвичай відноситься до верхньої оцінки асимптоматичної обчислювальної

складності алгоритму або задачі, яка зазвичай записується в термінах великої нотації O, наприклад $O(n^3)$. Інші типи оцінок обчислювальної складності – це нижні межі (позначається “Велика Омега”, наприклад, $\Omega(n)$) і асимптоматичні оцінки, коли асимптоматична верхня і нижня межі співпадають (записані з використанням “великої тети”, наприклад, $\Theta(n \log n)$).

У зв’язку з тим, що в рамках цієї кваліфікаційної роботи кількість вершин у вхідних даних буде не дуже великим, для виконання необхідних обчислень може бути використаним будь-який алгоритм з асимптоматичною обчислювальною складністю $O(n^2)$.

Таким чином, буде реалізований алгоритм переміщення, в силу його простоти і ефективності.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЕВРИСТИЧНОГО АЛГОРИТМУ ДЛЯ РІШЕННЯ ЗАДАЧІ МАРШРУТИЗАЦІЇ ТРАНСПОРТУ ТА ЙОГО АПРОБАЦІЯ

3.1. Підготовка до реалізації алгоритму

Використання евристичного алгоритму в умовах урахування вантажомісткості можна розділити на декілька етапів:

1) обґрунтування вибору технічних засобів;
2) підготовна вхідних даних;
3) реалізація алгоритму за допомогою обраних технічних засобів;
4) проведення обчислювального експерименту.

Готовий евристичний алгоритм переміщення, схематично зображений на рисунку 27 [21], повинен буде вирішувати задачу маршрутизації транспорту з урахуванням вантажомісткості.

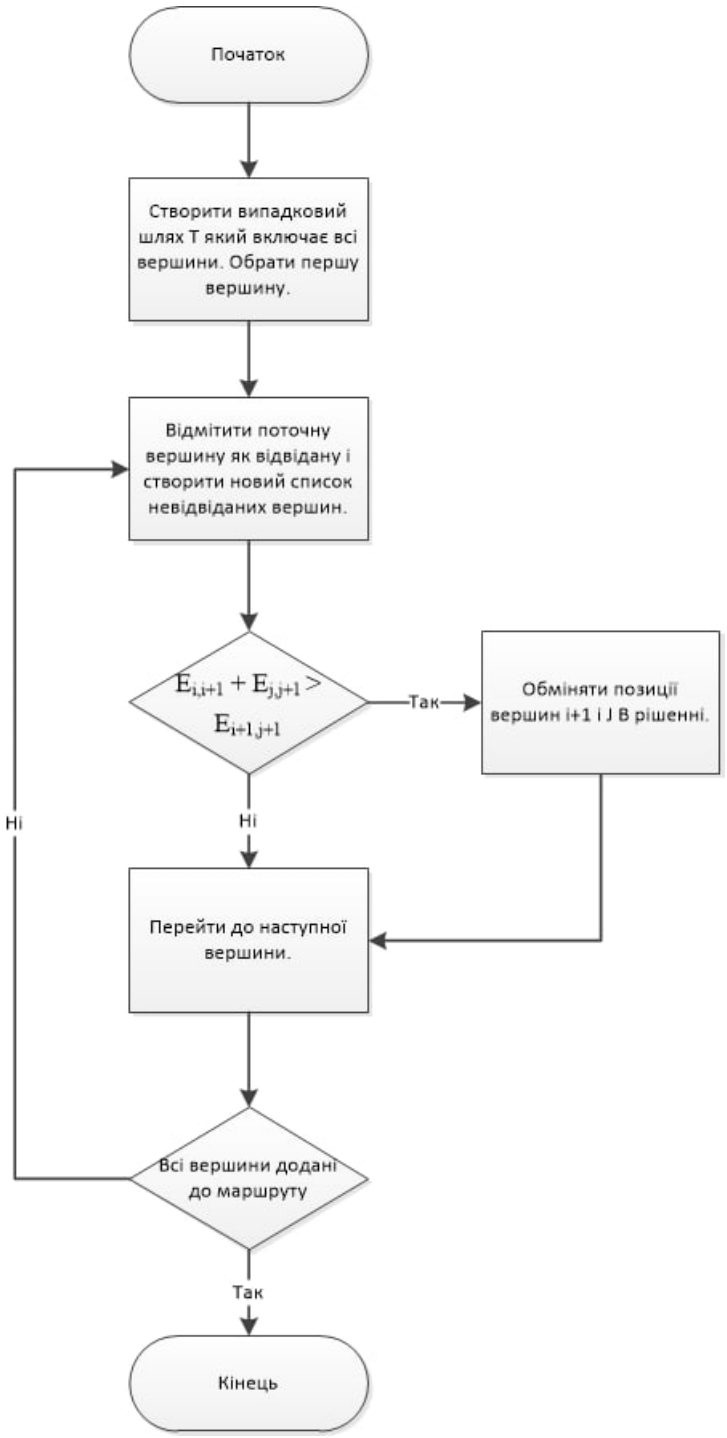

Рис. 27. Блок-схема алгоритму переміщення

3.2. Обґрунтування вибору технічних засобів

Мова програмування – це штучна мова, створена для передачі команд машинам, зокрема комп'ютерам. Мови програмування використовуються для створення програм, які контролюють поведінку машин, та для запису

алгоритмів. З часу створення перших програмованих машин було створено понад дві з половиною тисячі мов програмування. Щороку до них додаються нові. Деякими мовами вміє користуватись тільки невелике число їхніх розробників, інші стають відомі мільйонам людей. Професійні програмісти зазвичай застосовують у своїй роботі декілька мов програмування [22].

Мови програмування зазвичай складаються з інструкцій для комп'ютера. Мови програмування можуть використовуватися для створення програм, реалізуючи певні алгоритми [22].

Для розробки алгоритмів для рішення задач маршрутизації транспорту використовуються різні мови програмування. Розглянемо деякі з них:

C ++ – мова програмування високого рівня.

До переваг відносять:

1) підтримує різні стилі і технології;
2) можливість використання шаблонів для створення загальних контейнерів;
3) можливість імітації розширення мови для підтримки парадигм, які не підтримуються напряму;
4) є можливість роботи на низькому рівні з пам’яттю, адресами.

До недоліків відносять:

1) складність і перевантаженість;
2) відсутність GUI (Graphical user interface) за замовчуванням.

Matlab – мова програмування високого рівня, призначена для технічних розрахунків.

До переваг відносять:

1) простий для вивчення;
2) спрощена розробка нових алгоритмів;
3) щорічні оновлення модулів;
4) наявність готових модулів для роботи з нейронними мережами різних видів.

До недоліків відносять:

1) низька обчислювальна потужність;

2) перевантаженість операторами і функціями.

Java – об’єктно-орієнтована мова програмування. (ООП)

До переваг відносять:

1) використання парадигм ООП;

2) велика різноманітність бібліотек;

3) безпечність;

4) мультиплатформеність;

5) має багато зручних інструментів для автоматизації зборок.

До недоліків відносять:

1) необхідні великі ресурси;

2) складність синтаксису.

Python — інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією.

До переваг відносять:

1) використання парадигм ООП;

2) велика різноманітність бібліотек;

3) легкий синтаксис (динамічна типізація);

До недоліків відносять:

1) складність відлагодження (динамічна типізація);

2) складність синтаксису.

Таблиця 3. Порівняння мов програмування

	Java	C++	Matlab	Python
Продуктивність	+	+	-	-
Використання ООП	+	+-	-	+

Автоматизація зборки	+	-	-	+

Обираючи серед двох мов програмування, які є “чистокровними” об’єкто-орієнтованими мовами програмування, а саме Java і Python, обидві мови мають свої переваги і недоліки.

Швидкість виконання Python нижче ніж в Java. Оскільки Python динамічна та інтерпретується, виконання відбувається повільніше.

Java використовує більше пам’яті ніж Python, хоча і Python не можна назвати мовою, яка є економною по відношенню до пам’яті.

Складно підтримувати програму з великою кількістю рядків коду. Програмний код Java буде набагато більшим, ніж код Python. Сам синтаксис мови Python є дуже лаконічним і зрозумілим, а динамічна типізація спрощує оголошення змінних, але з іншої сторони, динамічна типізація вносить певну неясність розуміння того, якого типу даних змінна перед нами знаходиться, що може призвести до проблем з відлагодженням, коли програми почнуть збільшуватися в розмірах [23].

В сучасному світі, коли екологічність є дуже важливою складовою всіх напрямків людського існування, можна згадати цікаві дослідження і висловлювання вчених. Згідно з якими запуск програмного коду Python проходить набагато довше, ніж інших мов програмування і продукує більше викидів CO_2 в атмосферу, а також Python використовує найбільше електричної енергії під час своєї роботи (рис. 28) [24].

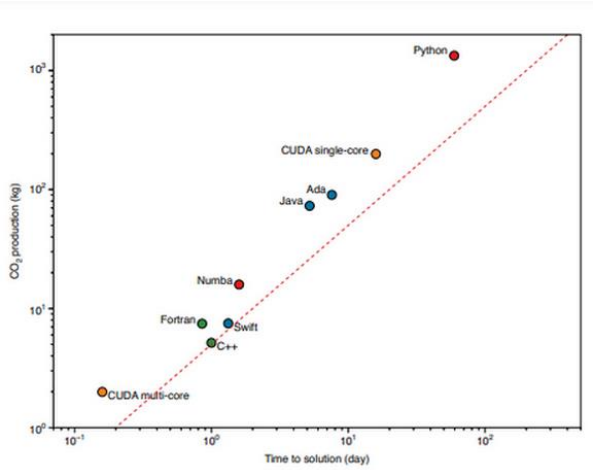

Рис. 28. Викид CO2 при використанні різних мов програмування

Серед цих мов програмування була обрана одна з об’єктно-орієнтованих мов, адже ООП дозволяє легко і лаконічно зберігати екземпляри певної сутності, яка має декілька властивостей. Якщо ж обирати серед об’єктно-орієнтованих мов, то можна знайти безліч доказів і в сторону Python і в сторону Java. Обидві мови на сьогоднішній день є дуже популярними і використовуються в широкому спектрі задач.

Для реалізації проєкту автором було обрано мову Java для реалізації евристичного алгоритму рішення ЗМТ.

3.3. Обрання тестових вхідних даних

Під вхідними даними в ЗМТ, розуміється набір вхідних параметрів, таких як кількість клієнтів і їх координати, умови обмежень і так далі. Замість реальних завдань, програмі може бути переданий заранння підготовлений набір даних для перевірки її ефективності, вони називаються тестами продуктивності.

Використовуються вони не тільки для перевірки працездатності програми, але і для того, щоб порівняти отримані результати (маршрути, час роботи програми) з результати роботи інших алгоритмів на цьому ж наборі вхідних даних.

Наведемо список вхідних даних для задачі маршрутизації транспорту з урахуванням вантажомісткості, що були взяті з роботи [25].

Набір А складається з 27 екземплярів, кількість ТЗ від 5-ти до 10-ти, максимальна вантажомісткість ТЗ 100 умовних одиниць (рис. 29).

Benchmark	Instance	*n*	*K*	*Q*	UB
	A-n32-k5	31	5	100	784
	A-n33-k5	32	5	100	661
	A-n33-k6	32	6	100	742
	A-n34-k5	33	5	100	778
	A-n36-k5	35	5	100	799
	A-n37-k5	36	5	100	669
	A-n37-k6	36	6	100	949
	A-n38-k5	37	5	100	730
	A-n39-k5	38	5	100	822
	A-n39-k6	38	6	100	831
	A-n44-k6	43	6	100	937
	A-n45-k6	44	6	100	944
	A-n45-k7	44	7	100	1146
	A-n46-k7	45	7	100	914
	A-n48-k7	47	7	100	1073
	A-n53-k7	52	7	100	1010
	A-n54-k7	53	7	100	1167
	A-n55-k9	54	9	100	1073
	A-n60-k9	59	9	100	1354
	A-n61-k9	60	9	100	1034
	A-n62-k8	61	8	100	1288
	A-n63-k9	62	9	100	1616
	A-n63-k10	62	10	100	1314
	A-n64-k9	63	9	100	1401
	A-n65-k9	64	9	100	1174
	A-n69-k9	68	9	100	1159
	A-n80-k10	79	10	100	1763

Рис. 29. Набір А

Набір B складається з 23 екземплярів, кількість транспортних засобів може бути від 5-ти до 10-ти, максимальна вантажомісткість ТЗ 100 умовних одиниць, кількість клієнтів від 30-ти до 77.

Benchmark	Instance	n	K	Q	UB
	B-n31-k5	30	5	100	672
	B-n34-k5	33	5	100	788
	B-n35-k5	34	5	100	955
	B-n38-k6	37	6	100	805
	B-n39-k5	38	5	100	549
	B-n41-k6	40	6	100	829
	B-n43-k6	42	6	100	742
	B-n44-k7	43	7	100	909
	B-n45-k5	44	5	100	751
	B-n45-k6	44	6	100	678
	B-n50-k7	49	7	100	741
	B-n50-k8	49	8	100	1312
	B-n51-k7	50	7	100	1032
	B-n52-k7	51	7	100	747
	B-n56-k7	55	7	100	707
	B-n57-k7	56	7	100	1153
	B-n57-k9	56	9	100	1598
	B-n63-k10	62	10	100	1496
	B-n64-k9	63	9	100	861
	B-n66-k9	65	9	100	1316
	B-n67-k10	66	10	100	1032
	B-n68-k9	67	9	100	1272
	B-n78-k10	77	10	100	1221

Рис. 30. Набір B

Набір E складається з 13 екземплярів, кількість ТЗ різниться від 4-ох до 14-ти, максимальна вантажомісткість ТЗ 112-6000 умовних одиниць в залежності від екземпляру, кількість вершин від 12-ти до 100.

Benchmark	Instance	*n*	*K*	*Q*	UB
	E-n13-k4	12	4	6000	247
	E-n22-k4	21	4	6000	375
	E-n23-k3	22	3	4500	569
	E-n30-k3	29	3	4500	534
	E-n31-k7	30	7	140	379
	E-n33-k4	32	4	8000	835
	E-n51-k5	50	5	160	521
	E-n76-k7	75	7	220	682
	E-n76-k8	75	8	180	735
	E-n76-k10	75	10	140	830
	E-n76-k14	75	14	100	1021
	E-n101-k8	100	8	200	815
	E-n101-k14	100	14	112	1067

Рис. 31. Набір E

Набір F складається з 3-ох екземплярів, кількість ТЗ від 5-ти до 7-ми, максимальна вантажомісткість 2010-30000 умовних одиниць, к-сть вершин від 30-ти до 77.

Benchmark	Instance	*n*	*K*	*Q*	UB
	F-n45-k4	44	4	2010	724
	F-n72-k4	71	4	30000	237
	F-n135-k7	134	7	2210	1162

Рис. 32. Набір F

Набір M складається з 5-ти екземплярів, кількість ТЗ від 7-ми до 17-ти, максимальна вантажомісткість 200 умовних одиниць, к-сть вершин від 100 до 199.

Benchmark	Instance	*n*	*K*	*Q*	UB
	M-n101-k10	100	10	200	820
	M-n121-k7	120	7	200	1034
	M-n151-k12	150	12	200	1015
	M-n200-k16	199	16	200	1274
	M-n200-k17	199	17	200	1275

Рис. 33. Набір M

Набір P складається з 23-х екземплярів, кількість ТЗ від 2-ох до 15-ти, максимальна вантажомісткість 35-3000 умовних одиниць, к-сть вершин від 15 до 100.

Benchmark	Instance	*n*	*K*	*Q*	UB
	P-n16-k8	15	8	35	450
	P-n19-k2	18	2	160	212
	P-n20-k2	19	2	160	216
	P-n21-k2	20	2	160	211
	P-n22-k2	21	2	160	216
	P-n22-k8	21	8	3000	603
	P-n23-k8	22	8	40	529
	P-n40-k5	39	5	140	458
	P-n45-k5	44	5	150	510
	P-n50-k7	49	7	150	554
	P-n50-k8	49	8	120	631
	P-n50-k10	49	10	100	696
	P-n51-k10	50	10	80	741
	P-n55-k7	54	7	170	568
	P-n55-k10	54	10	115	694
	P-n55-k15	54	15	70	989
	P-n60-k10	59	10	120	744
	P-n60-k15	59	15	80	968
	P-n65-k10	64	10	130	792
	P-n70-k10	69	10	135	827

Рис. 34. Набір P

Набір Z складається з 14-ти екземплярів, кількість ТЗ від 5-ти до 18-ти, максимальна вантажомісткість 140-200 умовних одиниць, к-сть вершин від 50 до 100.

Benchmark	Instance	*n*	*K*	*Q*	UB
	CMT1	50	5	160	524.61
	CMT2	75	10	140	835.26
	CMT3	100	8	200	826.14
	CMT4	150	12	200	1028.42
	CMT5	199	17	200	1291.29
	CMT6	50	6	160	555.43
	CMT7	75	11	140	909.68
	CMT8	100	9	200	865.95
	CMT9	150	14	200	1162.55
	CMT10	199	18	200	1395.85
	CMT11	120	7	200	1042.12
	CMT12	100	10	200	819.56
	CMT13	120	11	200	1541.14
	CMT14	100	11	200	866.37

Рис. 35. Набір Z

Набір W Складається з 12-ти екземплярів, кількість ТЗ від 10-ти до 20-ти, максимальна вантажомісткість 900-2500 умовних одиниць, к-сть вершин від 560 до 1200.

Benchmark	Instance	*n*	*K*	*Q*	UB
	Li_21	560	10	1200	16212.83
	Li_22	600	15	900	14499.04
	Li_23	640	10	1400	18801.13
	Li_24	720	10	1500	21389.43
	Li_25	760	19	900	16668.51
	Li_26	800	10	1700	23977.73
	Li_27	840	20	900	17372.64
	Li_28	880	10	1800	26566.03
	Li_29	960	10	2000	29154.34
	Li_30	1040	10	2100	31742.64
	Li_31	1120	10	2300	34330.94
	Li_32	1200	11	2500	37159.41

Рис. 36. Набір W

Кожен з представлених наборів вхідних даних відрізняються набором вхідних параметрів, наприклад вантажомісткістю ТЗ, кількістю маршрутів і кількістю самих ТЗ.

В якості екземпляра тесту продуктивності був обраний набір Z.

3.4. Реалізація та створення програмного модуля

Розробка програми для щоденного використання в логістичних задачах і її тест являється фінальною частиною роботи. Вона дає можливість для подальшої інтеграції реалізованого алгоритму, у випадку його успішної роботи до СППР ФОП “Устимук В. С.”, з отриманням фінансового ефекту.

Основні етапи вирішення ЗМТ:

1) завантаження вхідних даних;
2) обробка даних і створення декількох замкнутих маршрутів, які проходять через встановлену кількість цільових вершин. Через кожну вершину повинен проходити один єдиний маршрут;
3) обчислення матриці вартості переміщень;
4) запуск алгоритмів рішення ЗМТ для побудови маршрутів;
5) виведення отриманих результатів.

Критерії програмної реалізації ЗМТ:

1) всі маршрути повинні слідувати через депо;
2) єдина ціна об’їзду маршрутів повинна бути мінімальною.

Були виділені і створені наступні Java-класи, які наведені у додатку А. Модуль складається з наступних складових:

1) клас Node – зберігає інформацію про клієнта: його координати, максимальна вантажомісткість і порядковий номер;
2) клас Route – допоміжний клас, який зберігає в собі всі точки-клієнти і додаткову інформацію для роботи алгоритму;
3) клас Loader – призначений для зберігання вхідних даних (тестів продуктивності);

4) клас Utils – зберігає в собі набір методів для демонстрації отриманих рішень алгоритму;
5) клас Solution – клас, призначений для збору отриманих даних і розподілення їх по транспортним засобам;
6) клас Problem – формує проблему, яку потрібно вирішити;
7) клас Main – головна функція.

Наступним кроком, після визначення і реалізації класів, настав час реалізації самого алгоритму переміщення. Для початку був сформований псевдокод, представлений нижче.

Input: Множина клієнтів C = $\{c_1,\ldots,c_n\}$, n x n матриця D відстаней.
Output: Перестановка T = $(c_{p1},\ldots,c_{pn})$ клієнтів.
begin
t := початковий тур t_0;
Нехай Q = {(i,j) | i,j ∈ {1,…..,n} & i ≠ j};
N := Q;
repeat
Нехай t = $(c_{i1},\ldots,c_{in})$;
Обрати пару індексів (p, q) ∈ N;
N := N – {(p, q)};
t’:= $(c_{i1},\ldots,c_{ip-1},c_{iq},c_{iq-1},\ldots,c_{ip+1},c_{ip},c_{iq+1},\ldots,c_{in})$;
if l(t’) < l(t) **then**
begin
t:= t’;
N := Q
end
until N = not empty;
return t
end.

На основі псевдокоду, створювався програмний код на мові програмування Java в класі RelocateOptimizer. Фрагмент коду представлено нижче.

```
private boolean relocate(List<Vehicle> aList) {
        double gain;
        double bestGain = 0;

        Vehicle bestSourceVehicle = null;
        Vehicle bestDestVehicle = null;
        Pair<Integer, Integer> bestRelocate = new Pair<>();

        for (Vehicle b : aList) {
                for(Vehicle a : aList){
                        System.out.println("expanding {" + a + "}, shrinking {" + b +
"}";

                        float remainingCapacity = a.getCapacity() -
a.getRoute().demand();

                        for(int i = 1; i < b.getRoute().size() - 1; i++){
                        if(b.getRoute().demand() < remainingCapacity){
                                System.out.println("node {" + i + "} fit into route {" + a +
" }");
                                /*
                                *
                                *
                                */
                                for(int j = 0; j < a.getRoute().size() - 1; j++){
                                gain = calculateGain(a.getRoute(), b.getRoute(), j,i);
```

```
                        System.out.println("node {" + i + "}, between {" + j + "}
{" + (j + 1) + "}, gain = {" + gain + "}");

                        if (gain > bestGain){
                        bestSourceVehicle = b;
                        bestDestVehicle = a;
                        bestRelocate.set(i, j);
                        bestGain = gain;
                        }
                        }
                    }
                    }
                    }
        }
}
```

Для забезпечення демонстрації роботи алгоритму, був реалізований консольне виведення інформації. На першому етапі виводяться вхідні дані в зручній для перегляду формі (Рис. 37).

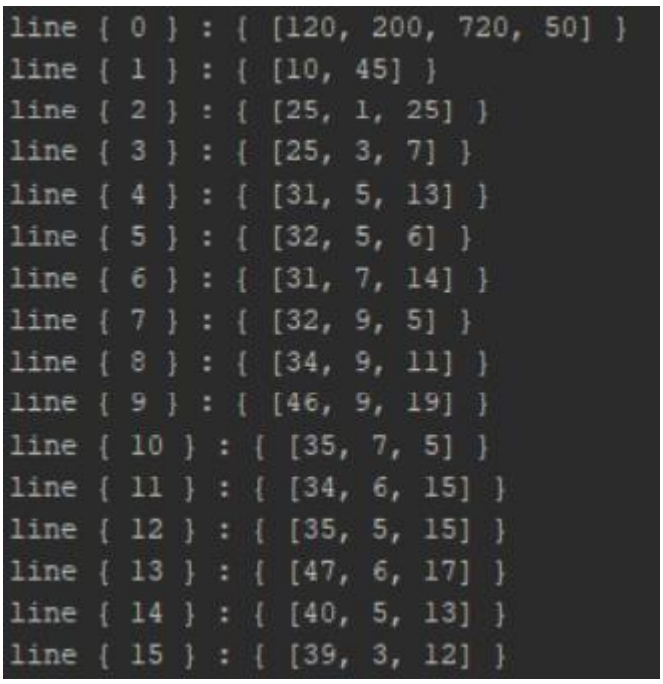

Рис. 37. Приклад виведення консольного виведення початкових значень

В результаті роботи алгоритму виводяться оптимізовані маршрути. Ці маршрути можна побачити на рисунку 38.

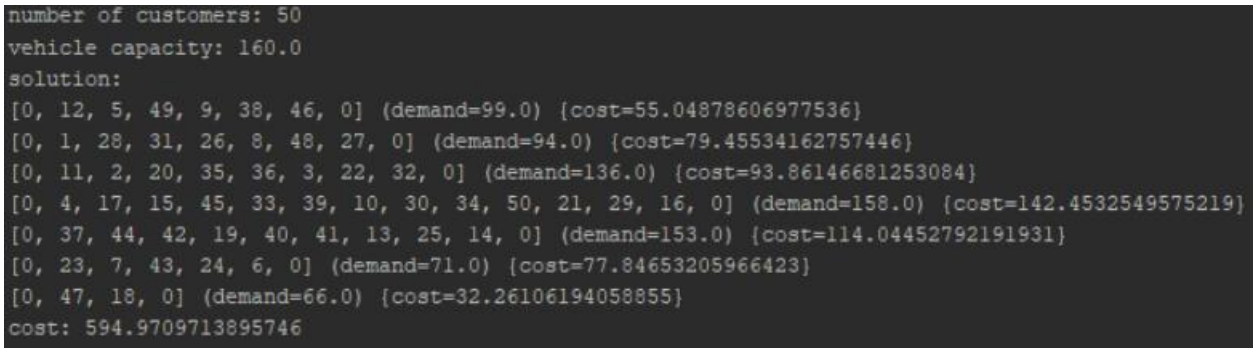

Рис. 38. Приклади маршрутів ЗМТУВ

3.5. Проведення апробації використання створеного модуля

Порівняємо результати тесту продуктивності на наборі тестових наборів представлених в роботі [25] по кожному екземпляру, з результатами обраного евристичного алгоритму.

Отримані результати представлено в таблиці 4.

Таблиця 4. Результати роботи алгоритму

	Кращі значення вартості маршрутів, отриманих на основі вхідних даних Z	Значення вартості маршрутів, отримані розробленим алгоритмом
1.	524,61	594,97
2.	835,26	903,90
3.	826,14	961,39
4.	1028,42	1228,09
5.	1291,29	1625,01
6.	555,43	594,97
7.	909,68	903,90
8.	865,95	961,39
9.	1162,55	1228
10.	1395,85	1625
11.	1042,12	1438,90
12.	819,56	967,81
13.	1541,14	1438
14.	866,37	967

В якості результативного параметра обирається загальна вартість проїду автомобіля з всіх оптимізованих маршрутів. В першій колонці представлені екземпляри тесту продуктивності, в кількості 14 штук. В другій колонці

знаходяться кращі результати тесту, в третій — результати отримані під час застосування реалізованого алгоритму переміщення.

Провівши аналіз таблиці 4, ми можемо сказати, що отримані нами результати, отримані під час використання евристичного алгоритму переміщення, близькі до кращих результатів даного тесту продуктивності.

3.6. Інтеграція алгоритма до існуючої СППР

Після апробації, отриманих під час тестів, результатів, алгоритм був інтегрований до існуючої системи підтримки прийняття рішень логіста. В цій СППР існує довідник угод, який формується на основі обраних вигідних для виконання замовлень, і тепер на формі цього довідника з’явиться додаткова кнопка “Розрахувати маршрути”, натиск на яку буде забезпечувати виконання обраного алгоритму.

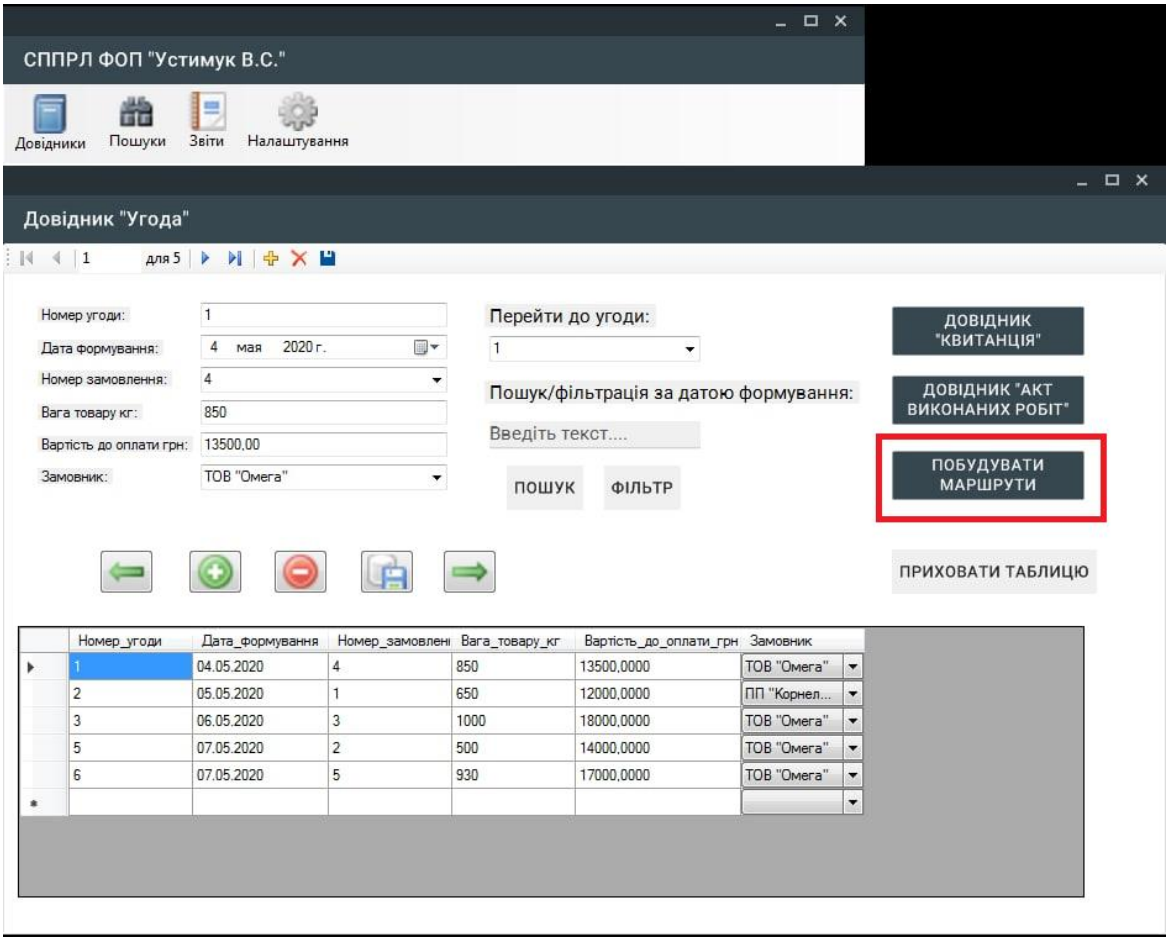

Рис. 39. Форма довідника з новою кнопкою

Натиск на цю кнопку активує алгоритм, який будує маршрути, охоплюючи всіх клієнтів, яких необхідно відвідати, при цьому мінімізуючи шлях, який необхідно пройти, кількість транспортних засобів, необхідних для цього, і, отже, вцілому зменшує витрати на виконання замовлень.

Після того як маршрути буде розраховано, програма виведе відповідне сповіщення про те, що вони сформовано, та назву файлу, в якому збережено ці маршрути. Файл має назву, яка відповідаю дню, коли було сформовані ці маршрути, як на рисунку 40.

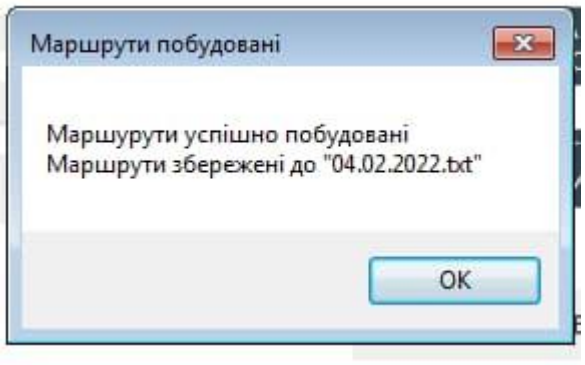

Рис. 40. Вікно сповіщення про побудову маршрутів

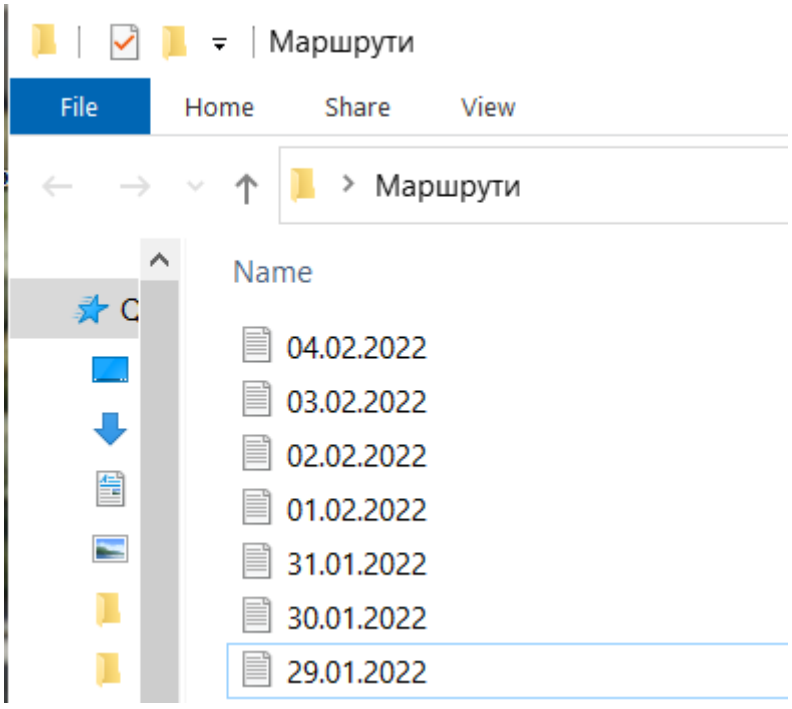

Рис. 41. Вікно з файлими маршрутами

На рисунку 42 ми бачимо приклад створених маршрутів. Можемо звернути увагу на те, що в маршрутах ми маємо більше клієнтів, ніж під час перегляду угод, це пов’язано з тим, що багато клієнтів має більше одного складу, а в рамках вирішення саме задачі маршрутизації транспорту, нас цікавить саме кількість складів клієнтів.

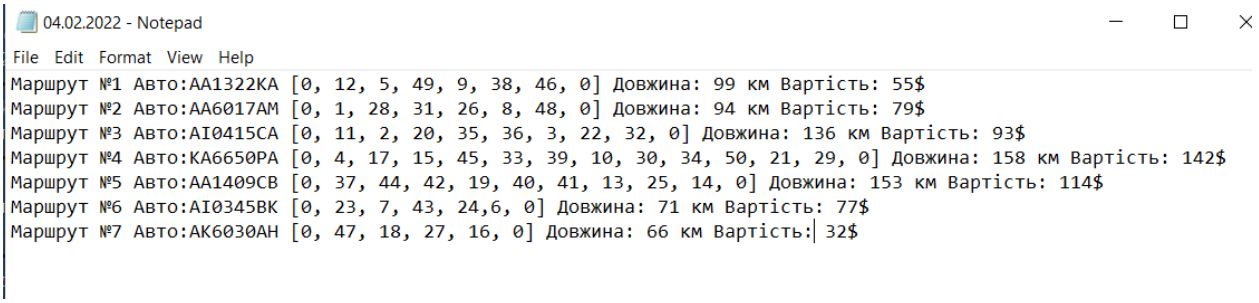

Рис. 42. Приклад створених маршрутів

Також можна звернути увагу на те, що клієнти-склади в маршруті позначені своїм порядковим номер вершини графа. Таке представлення є достатнім для побудови маршрутів. Але коли цей маршрут буде передано водію, він вже буде мати зовсім інший вигляд. Приклад того, як буде виглядати маршрут, який буде передано водію представлено на рисунку 43.

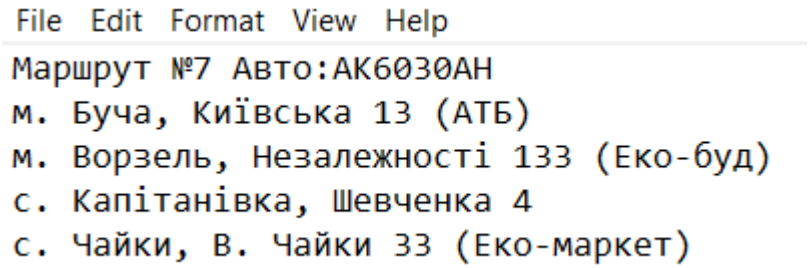

Рис. 43. Розшифрований маршрут для водія

ВИСНОВКИ

Кваліфікаційна робота на здобуття освітнього ступеню магістра присвячена розв’язанню науково-технічного завдання покращенню процесу прийняття рішень логістів ФОП “Устимук В. С.” при обранні ефективного маршруту для забезпечення мінімізації витрат на доставку замовлень.

В даній роботі були проаналізовані та досліджені види задач маршрутизації транспорту, досліджені методи рішення ЗМТ, досліджено існуючі алгоритми вирішення ЗМТ та обґрунтовано обрання найбільш ефективного алгоритму; удосконалено існуючу СППР логіста ФОП “Устимук В. С.” за рахунок використання обраних алгоритмів та провести апробацію для оцінки ефективності їх використання для побудови маршрутів в реальних логістичних завданнях. В роботі використані наступні методи дослідження: метод структурно-функціонального аналізу для дослідження предметної області; методи моделювання та побудови баз даних, для аналізу та удосконалення бази даних системи; евристичні та метаевристичні алгоритми для розв’язання задачі побудови маршрутів; об’єктно-орієнтоване програмування для створення елементів системи.

Практичне значення результатів магістерських досліджень полягає у обґрунтуванні алгоритму вирішення задачі маршрутизації транспорту, для подальшої інтеграції даного алгоритму в існуючі інформаційні системи для автоматизації процесу побудови маршрутів та вибору автомобілів.

Отриманий в результаті програмний продукт, удосконалить процес прийняття рішень логіста при виборі автомобілів для виконання замовлень та прокладання маршруту для виконання замовлень за рахунок використання інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Michael Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness [Text]/ Garey Michael, S. Johnson David// New York: W. H. Freeman & Co. – 1979.
2. ЗАДАЧА МАРШРУТИЗАЦИИ ТРАНСПОРТА. ДИСКРЕТНАЯ МАТЕМАТИКА: АЛГОРИТМЫ [Електронний ресурс] – Режим доступу до ресурсу: https://www.lobanov-logist.ru/library/352/55059/.
3. Hoskins M., Lobit H., Pillac V., Vidal T., Vigo D., Mendoza J.E., Guéret C., VRP-REP: the vehicle routing community repository. Third meeting of the EURO Working Group on Vehicle Routing and Logistics Optimization [Электронный ресурс] / M. Hoskins, H. Lobit, V. Pillac, T. Vidal, D. Vigo, J.E. Mendoza, C. Guéret // Oslo, Norway – 2014.— 89p.
4. John E. Mitchell, Branch-and-Cut Algorithms for Combinatorial Optimization Problems [Text] / E. Mitchell John // Mathematical Sciences Rensselaer Polytechnic Institute Troy, NY, USA. – 1999. – P.19.
5. Метаевристика [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Метаевристика.
6. Мурашиний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Мурашиний_алгоритм
7. Генетичний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Генетичний_алгоритм
8. Алгоритм імітації відпалу [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Алгоритм_імітації_відпалу
9. Табу-пошук [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Табу-пошук
10. Евристичний алгоритм [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Евристичний_алгоритм
11. Евристичні алгоритми. [Електронний ресурс] – Режим доступу до ресурсу: http://mmsa.kpi.ua/sancho/ASD_HTM/Arti04.html.
12. Конструктивна евристика - Constructive heuristic [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikisla.ru/wiki/Constructive_heuristic.
13. Clark G., Write J. W., Scheduling of vehicles from central depot to a number delivery points / G. Clark, J. W. Write// Oper. Res. Quarts, 1964. – 568-581c
14. Ryan D. M., Hjorring C., Glover F., Extensions of the Petal Method for Vehicle Routing [Text] / D. M. Ryan, C. Hjorring, F. Glover // Journal of the Operational Research Society. – 1993. – P.289-296.

15. GUlert B.E., Miller L.R., A heuristic algorithm for the vehicle dispatch Problem [Text] / B.E. GUlert, L.R. Miller // Operation Research. – 1974. – Vol.22 – P. 340-349.
16. Fisher M., Jaikumar R., A generalized assignment heuristic for vehicle routine / M. Fisher, R. Jaikumar // Net-works, 1981. – P. 109-124.
17. Lin S., Computer solutions of the traveling salesman problem [Электронный ресурс] / S. Lin // Bell System Technical Journal. — 1965. — № 44. — P. 2245–2269.
18. Croes G.A., A Method for Solving Traveling-Salesman Problems [Text] / G.A. Croes // Operations Research. – 1958. – Vol. 6, № 6. – P. 791-812.
19. Ferrucci, Francesco, Pro-active Dynamic Vehicle Routing [Книга] / Ferrucci, Francesco // Operations Research. — 2013г. — P. 280.
20. Hartmanis J., Stearns R. E., On the computational complexity of algorithms [Text] J. Hartmanis, R. E. Stearns // Transactions of the American Mathematical Society. – 1965. – P. 285-306.
21. Sathyan Anoop, Boone Nathan, Cohen Kelly, Comparison of Approximate Approaches to Solving the Travelling Salesman Problem and its 42 Application to UAV Swarming [Электронный ресурс]/ Anoop Sathyan, Nathan Boone, Kelly Cohen // University of Cincinnati, USA. – 2015. – P. 17.
22. Мова програмування [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Мова_програмування
23. Что лучше: Java, C++ или Python? [Електронний ресурс] – Режим доступу до ресурсу: https://upread.ru/blog/articles-it/java-c-ili-python.
24. Астрономам порекомендовали меньше использовать суперкомпьютеры и Python из-за вреда экологии [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/news/t/519414/.
25. Christofides N., The vehicle routing problem. In N. Christofides, A. Mingozzi, P. Toth, C. Sandi, editors [Text] / N. Christofides, A. Mingozzi, P. Toth // Combinatorial Optimization. — Wiley, Chichester, 1979. — P. 315–338.

ДОДАТКИ

ДОДАТОК А. Діаграма класів