Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи рішення задачі маршрутизації транспорта з урахуванням додаткових обмежень
(тема)

Виконав:

студент II курсу, групи СПм-23-2

Бондаренко К.В.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Іващенко Г.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.
кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бондаренко Катерині Володимирівні
(прізвище, ім'я, по батькові)

1. Тема роботи Методи рішення задачі маршрутизації транспорта з урахуванням додаткових обмежень

затверджена наказом по університету від " 22 " листопада 2024 р. № 1236 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи 1) граф; 2) теорія графів; 3) задача маршрутизації транспорту; 4) місткість транспорту; 5) часові вікна; 6) алгоритми побудови маршрутів; 7) інтегроване середовище розробки Visual Studio 2022; 8) мова програмування C#; 9) джерела інформації з теорії графів та задач пошуку оптимальних маршрутів.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) розгляд алгоритмів рішення задачі маршрутизації транспорту;
3) вибір технологій розробки та інструментальних засобів;
4) реалізація програмного застосунку;
5) аналіз результатів досліджень;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд алгоритмів рішення задачі маршрути-	26.11.24-30.11.24	
	зації транспорту		
2	Вибір та обґрунтування методики дослідження	02.12.24-05.12.24	
3	Вибір інструментальних засобів	06.12.24-10.12.24	
4	Розробка алгоритмічного забезпечення	11.12.24-21.12.24	
5	Проведення експериментів	23.12.24-03.01.25	
6	Оформлення матеріалів кваліфікаційної роботи	04.01.25-07.01.25	
7	Подання кваліфікаційної роботи керівникові та	08.01.25-11.01.25	
	її попередній захист		
8	Подання кваліфікаційної роботи на рецензу-	13.01.25-17.01.25	
	вання		

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Іващенко Г.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 99 с., 11 рис., 6 табл., 2 дод., 54 джерела.

ГРАФ, ОПТИМАЛЬНІ МАРШРУТИ, ЗАГАЛЬНА ДОВЖИНА МАРШРУТІВ, ЧАСОВІ ВІКНА, МІСТКІСТЬ ТРАНСПОРТУ, АЛГОРИТМ ТАБУ ПОШУКУ, ЖАДІБНИЙ АЛГОРИТМ, ГЕНЕТИЧНИЙ АЛГОРИТМ, МУРАШИНИЙ АЛГОРИТМ, АЛГОРИТМ ІМІТАЦІЇ ВІДПАЛУ.

Метою кваліфікаційної роботи є дослідження використання метаевристичних алгоритмів при вирішенні задач маршрутизації транспорту з обмеженою вантажопідйомністю та часовими вікнами.

У ході виконання роботи розроблено застосунок, який виконує пошук оптимальних маршрутів на заданих користувачем вхідних даних за допомогою наближених алгоритмів: жадібного, генетичного та мурашиного алгоритмів, табу пошуку та імітації відпалу. Алгоритми протестовані на різних наборах даних, включаючи варіанти з різними обмеженнями та розмірністю задачі. Також проведено порівняльний аналіз результатів роботи кожного алгоритму щодо класичного евристичного методу, такого як жадібний алгоритм, для визначення їх ефективності в контексті задачі CVRPTW.

ABSTRACT

Master’s thesis: 99 pages, 11 figures, 6 tables, 2 appendices, 54 sources.

GRAPH, OPTIMAL ROUTES, TOTAL ROUTES LENGTH, TIME WINDOWS, TRANSPORT CAPACITY, TABU SEARCH ALGORITHM, GREEDY ALGORITHM, GENETIC ALGORITHM, ANT COLONY OPTIMIZATION ALGORITHM, SIMULATED ANNEALING ALGORITHM.

The major goal of this thesis is to analyze the work of metaheuristic algorithms in solving transport routing problems with limited capacity and time windows.

In order to analyze these algorithms, an application has been developed that finds for optimal routes on user-specified input data using all of the researched algorithms. The algorithms were tested on various data sets, including options with different constraints and problem dimensions. A comparative analysis of the performance of each algorithm against a classical heuristic method, such as the greedy algorithm, was also conducted to determine their effectiveness in the context of the CVRPTW problem.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ТЗ – транспортний засіб

ACO – алгоритм оптимізації колоній мурах (англ., Ant Colony Optimization)

CVRP – завдання маршрутизації транспорту з обмеженою вантажопідйомністю (англ., Capacitated Vehicle Routing Problem)

CVRPTW – завдання маршрутизації транспорту з урахуванням обмеженої вантажопідйомності та часовими вікнами (англ., Capacitated Vehicle routing problem with Time Windows)

GA – генетичний алгоритм (англ., Genetic Algorithm)

MDVRP – завдання маршрутизації транспорту з кількома депо (англ., Multiple Depot Vehicle Routing Problem)

PVRP – періодичне завдання маршрутизації транспорту (англ., Periodic Vehicle Routing Problem)

SA – алгоритм імітованого відпалу (англ., Simulated Annealing Algorithm)

SVRP – стохастичне завдання маршрутизації транспорту (англ., Stochastic Vehicle Routing Problem)

TS – алгоритм табу-пошуку (англ., Tabu Search)

VRP – завдання маршрутизації транспорту (англ., Vehicle routing problem)

VRPPD – завдання маршрутизації транспорту з підбором та доставкою (англ., Vehicle Routing Problem with Pick-ups and Deliveries)

VRPSF – завдання маршрутизації транспорту з супутніми об'єктами (англ., Vehicle Routing Problem with Satellite Facilities)

VRPTW – завдання маршрутизації транспорту з часовими вікнами (англ., Vehicle Routing Problem with Time Windows)

ВСТУП

Завдання маршрутизації транспортних засобів (Vehicle Routing Problem – VRP) є практичною проблемою у системах транспортування, розподілу та логістики. Її можна визначити як задачу створення оптимальних маршрутів доставки (або збору) товарів з одного або декількох складів до набору географічно розташованих точок або клієнтів з урахуванням додаткових обмежень. Зазвичай об'єктивні витрати VRP визначаються на основі суми відстаней, часу в дорозі, кількості транспортних засобів або їх комбінації [1].

У логістичних завданнях, на практиці, VRP розглядають із різними обмеженнями, які необхідно враховувати під час побудови маршрутів. У роботі розглянуто задачу маршрутизації транспорту з урахуванням обмеженої вантажопідйомності та часовими вікнами (Capacitated VRP With Time Windows – CVRPTW).

Процес пошуку оптимальних маршрутів здійснюється за допомогою спеціалізованих алгоритмів. Ці алгоритми відрізняються між собою правилами прийняття рішень у процесі вирішення задачі, що дозволяє ефективно знаходити мінімальні маршрути для транспортних засобів. З їх допомогою можна не лише оптимізувати використання ресурсів, а й значно скоротити час на виконання завдань доставки, покращити планування маршрутів, а також мінімізувати операційні витрати [2].

Метою роботи є проведення аналізу існуючих методів рішення задачі маршрутизації транспорту з урахуванням обмеженої вантажопідйомності с часовими вікнами. Робота ґрунтується на досліджені евристичних та метаевристичних підходів (жадібний, генетичний, мурашиний алгоритми, табу-пошук та імітація відпалу) і завершується отриманням даних про результати цих рішень, що оцінюються двома основними метриками: часом пошуку рішення та остаточною вартістю маршруту, визначеною цільовою функцією.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис задачі

Завдання маршрутизації транспорту – це клас завдань комбінаторної оптимізації, спрямований на мінімізацію витрат на транспортні ресурси, вартість маршруту та часу доставки товару групі клієнтів.

Класично завдання транспортної маршрутизації описується так: є парк транспортних засобів (ТЗ), які спочатку знаходяться в депо, що є початком і кінцем кожного маршруту, та певна кількість точок, яким необхідно доставити товар. Задано матрицю невід'ємних вартостей або відстаней між клієнтами.

Метою завдання VRP є знаходження маршруту для кожного транспорту, такого, щоб усі точки були відвідані лише один раз рівно одним транспортним засобом, і щоб загальна вартість маршрутів була мінімальною (рисунок 1.1). Також повинні виконуватися додаткові обмеження задачі [3].

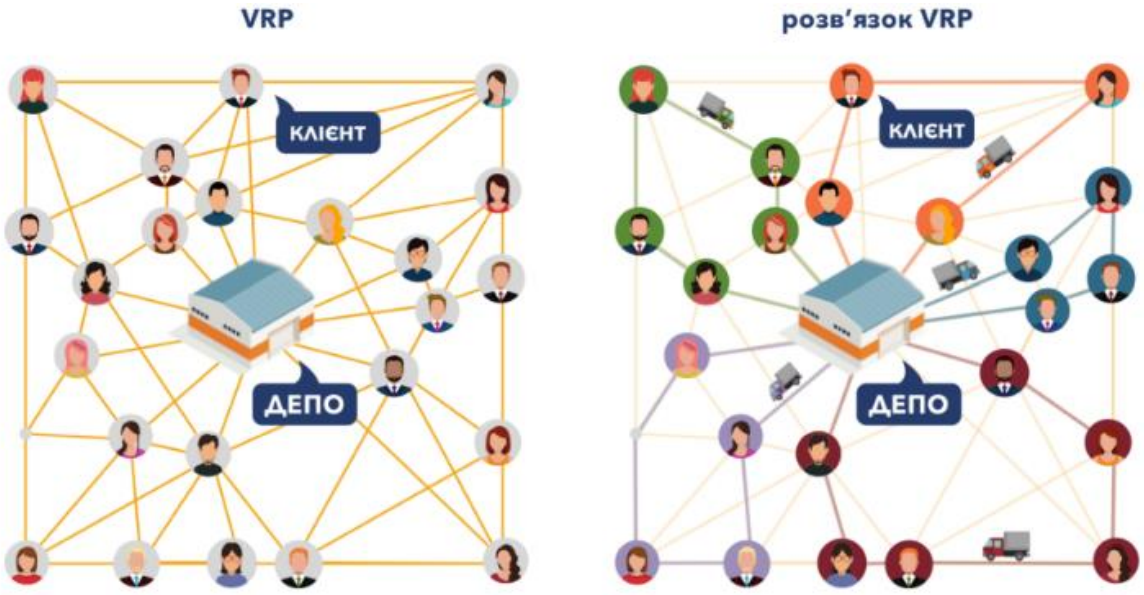

Рисунок 1.1 – Приклад VRP та один з варіантів її розв’язання

Через різні умови та потреби при плануванні маршрутів VRP набула багатьох варіацій, що формулюють різні завдання маршрутизації транспорту, відмінні від класичного. Найбільш дослідженими варіантами задач маршрутизації є:

- Capacitated VRP (CVRP): кожний транспортний засіб має обмежену вантажопідйомність;
- VRP With Time Windows (VRPTW): кожен замовник має бути обслугований у певне «часове вікно»;
- Multiple Depot VRP (MDVRP): використовується декілька депо для обслуговування клієнтів;
- VRP With Pick-Ups And Delivering (VRPPD): клієнти можуть повертати товари в депо;
- VRP With Backhauls (VRPB): аналогічно до попередньої, але повернення починається тільки після доставки всіх товарів з депо;
- Split Delivery VRP (SDVRP): кожен клієнт може обслуговуватись одночасно декількома машинами;
- Periodic VRP (PVRP): доставка здійснюється протягом кількох днів;
- Stochastic VRP (SVRP): деякі компоненти завдання (кількість та запити клієнтів, довжина шляху) можуть мати випадкову поведінку;
- VRP With Satellite Facilities (VRPSF): існує можливість дозавантаження автомобіля на маршруті [4].

Для задачі маршрутизації містких транспортних засобів (CVRP) у кожної точки є попит, який відповідає кількості (наприклад, вазі або обсягу) товару, який необхідно забрати або доставити транспортом. Кожен транспортний засіб може перевозити обмежену вагу і також може бути обмежений за загальною відстанню, яку він може проїхати. Ціль: мінімізувати парк машин, необхідних для виконання завдання, а також загальний час виконання завдання [5, 7].

У задачі маршрутизації транспортних засобів з часовими вікнами

(VRPTW) для виконання запиту кожного клієнта існує відомий проміжок часу, визначений як інтервал – часове вікно, у яке кожен клієнт має обслуговуватись лише одним транспортним засобом. Ціль: мінімізувати кількість машин, загальні часи шляху та очікування, необхідні для обробки запитів клієнтів у призначені інтервали часу. Часові вікна можна розділити на дві категорії: жорсткі часові вікна, які забороняють доставку за межами заздалегідь визначеного вікна, та м'які часові вікна, які допускають деяку гнучкість, беручи штрафи за ранню чи пізню доставку. У роботі використовуються жорсткі часові вікна, оскільки таке обмеження найчастіше застосовуються на практиці [6, 7].

Таким чином, головна мета завдання CVRPTW може бути сформульована наступним чином: транспортні засоби, що мають обмежену вантажопідйомність, повинні доставити товари споживачам у певні часові проміжки з відомим попитом, мінімізуючи при цьому витрати на перевезення (загальну вартість та час виконання доставки), забезпечуючи використання меншої кількості транспорту (рисунок 1.2).

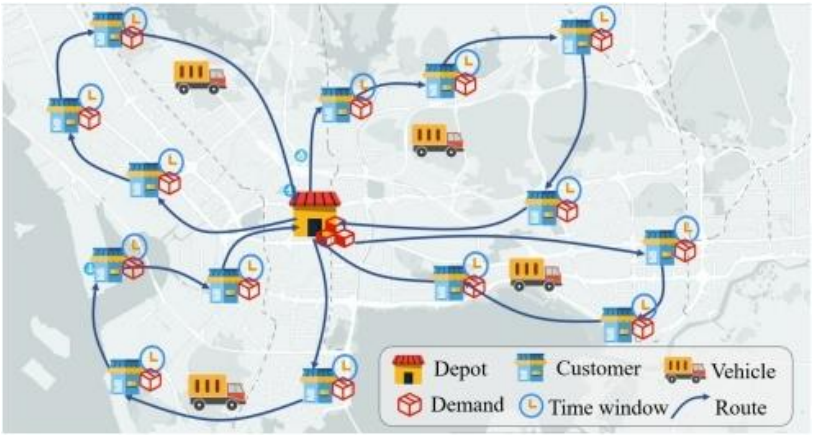

Рисунок 1.2 – Приклад задачі CVRPTW

У задачі використовується вартість об'їзду послідовності вершин транспортним засобом – параметр, що виражає будь-які витрати на відвідування клієнтів і може бути представлений як вартість палива, що

споживається, так і час, необхідний для виконання роботи. Для дослідження алгоритмів не важливо, які витрати характеризує даний параметр. На практиці він приймається як еквівалент мінімізації загальної пройденої відстані мінімальної кількості транспортних засобів. Також, треба враховувати, що вартість проїзду не може від’ємною.

1.2 Галузі застосування задачі маршрутизації транспорту

При вирішенні практичних транспортних завдань планувальникам потрібна система, яка дозволяє точно обчислювати обсяги вантажоперевезень у необхідний час, розраховувати кількість одиниць транспорту, необхідних для забезпечення вантажопотоків, визначати раціональні маршрути руху, а також скоротити сумарні витрати на транспортування. Прикладами застосування можуть бути доставка пошти, збирання твердих відходів, прибирання вулиць, розподіл товарів, проектування телекомунікацій, транспортні мережі, маршрутизація шкільних автобусів, системи dial-a-ride, перевезення інвалідів та маршрутизація продавців та підрозділів з технічного обслуговування [8].

Розглядаються найбільш широко розповсюджені сфери, що використовують алгоритми рішення задачі маршрутизації транспорту. В [9] проведено дослідження застосунків маршрутизації транспортних засобів, їх використання та значення у різних секторах та галузях по всьому світу. У результаті виявлено, у яких галузях застосування рішень маршрутизації транспортних засобів може допомогти зробити роботу менш складною.

1.2.1 Електронна комерція

В умовах швидкого зростання електронної комерції (e-commerce), алгоритми VRP знаходять широке застосування для вирішення завдань доставки замовлень «до дверей» клієнтів. Онлайн-рітейлери, такі як Amazon,

використовують VRP для управління своїм великим автопарком транспортних засобів. Задача полягає в тому, щоб доставити тисячі замовлень із різних складів до кінцевих споживачів за мінімальний час і з мінімальними витратами. При цьому враховуються географічне розташування клієнтів, часові вікна доставки та обмеження на вагу і об'єм товару в транспорті. Це дозволяє великим компаніям знизити логістичні витрати, одночасно підвищуючи рівень обслуговування, зокрема, забезпечуючи доставку в той же день чи навіть протягом кількох годин [10].

1.2.2 Постачальники та розподільчі компанії

Часто у сфері продажів розподільчі компанії оцінюють рівень запасів клієнтів в такий спосіб, щоб поповнювати їх до виснаження запасів. Отже, компанії заздалегідь знають перелік клієнтів, яких потрібно обслуговувати, і відповідно планують маршрути для доставки товарів (рисунок 1.3). Однак фактичний обсяг попиту залишається невизначеним, деякі клієнти можуть вичерпати свій запас і їх необхідно терміново обслужити. Тому, вони повинні враховувати обмеження на час доставки, оскільки клієнти мають фіксовані часові вікна, коли вони можуть прийняти товар.

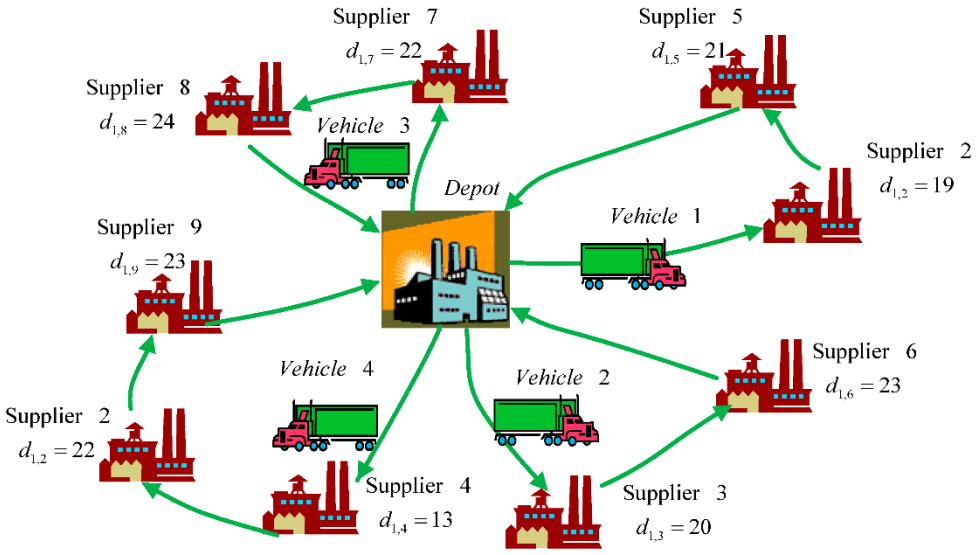

Рисунок 1.3 – Приклад VRP для постачальників та розподільчих компаній

Рішення задачі дозволяє компаніям ефективно планувати маршрути для різних транспортних засобів, мінімізувати витрати на доставку та уникати штрафів за запізнення. Зокрема, це актуально для бізнесів, які постачають товари до магазинів або інших об'єктів, де час доставки може бути критичним для підтримання запасів на належному рівні [11].

1.2.3 Використання дронів для доставки

Дрони використовується у сферах, пов'язаних з комерційним та некомерційним використанням, водночас вони використовуються для роботи з рішеннями VRP, відомими як VRPD. Дрони використовуються декількома компаніями, такими як Amazon, DHL, Federal Express, які можуть доставляти товари за допомогою дронів прямим або непрямим способом. Рисунок 1.4 описує безпілотник, створений для цілей доставки.

Рисунок 1.4 – Приклад дрону

У роботі [12] запропоновано алгоритм з використанням VRPD, де транспортні засоби доставки з набором дронів, що перевозять товари, прямують у певні точки, де дрони беруть на себе керування, доставляючи товари звідти до найближчих будинків і повертаючись до транспортного засобу. Потім транспортний засіб повертається на склад, коли всі призначені товари будуть доставлені.

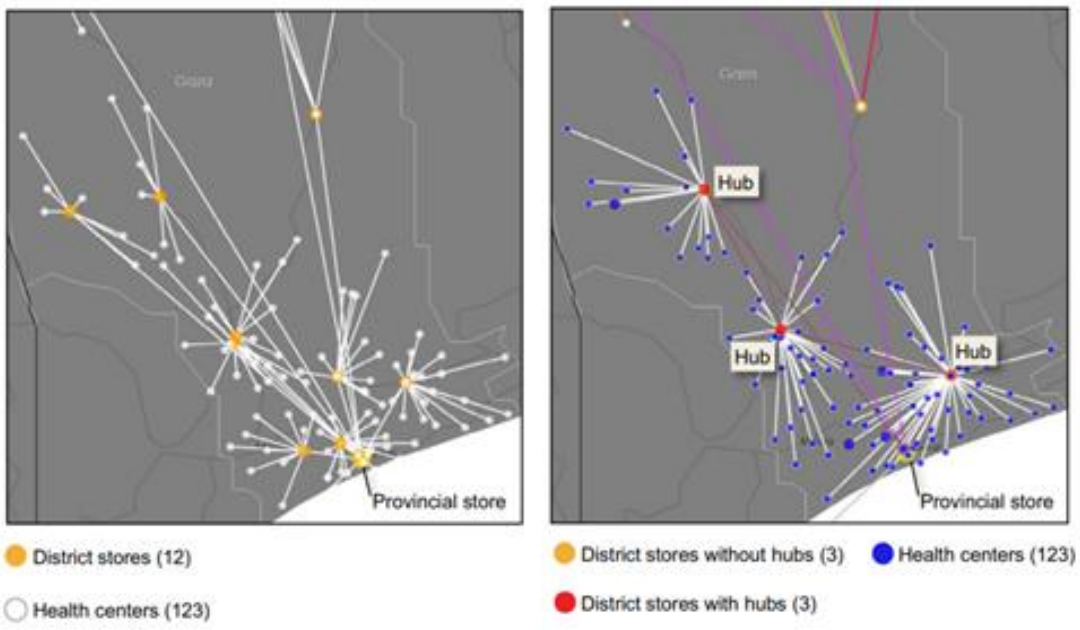

Рисунок 1.5 – Маршрут VRP до та після впровадження дронів

На рисунку 1.5 показано порівняльне дослідження маршрутизації транспортних засобів в області між використанням безпілотника і без використання безпілотника. У результаті кількість кластерів чи підстанцій скорочується через використання безпілотників.

Дослідження застосування безпілотників у логістиці стає важливим напрямом, що привертає увагу клієнтів для застосування дронів у багатьох галузях [13].

1.2.4 Збирання відходів

На рисунку 1.6 показані маршрути транспортних засобів, які мають збирати відходи у місті, із зазначенням кількості зупинок та загальної відстані, яка буде пройдена.

Реальні програми зазвичай включають декілька аспектів і обмежень, які не є достатніми в базовому VRP, що призводить до додавання обмежень до VRP, таких як пропускна здатність та часові вікна [14, 15].

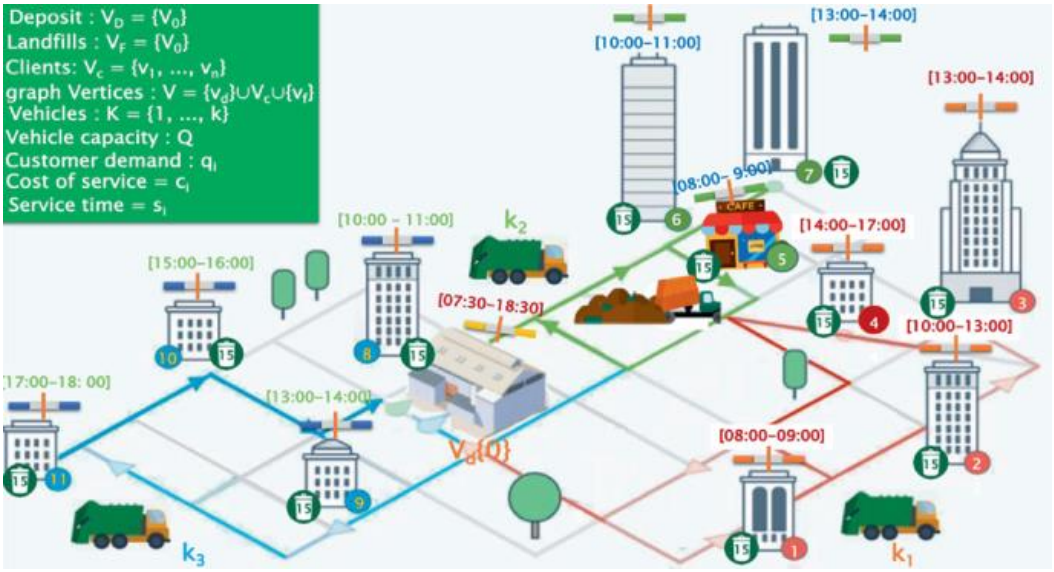

Рисунок 1.6 – Проблема маршруту транспортних засобів (збирання відходів)

1.2.5 Автоматизовані склади

У сучасних автоматизованих складах, де значна частина роботи виконується роботизованими системами, алгоритми VRP використовуються для оптимізації переміщення товарів по складу. Такі алгоритми дозволяють визначити найкоротші маршрути для перевезення товарів між різними точками, такими як зонами зберігання, пакувальними станціями та відвантажувальними пунктами.

Рішення VRP допомагають уникати заторів у складських коридорах, забезпечують безперервний потік товарів і мінімізують час, необхідний для виконання складських операцій. Це особливо актуально для складів з великою кількістю рухомих елементів, де ефективна маршрутизація може значно підвищити швидкість обробки замовлень [16].

1.3 Загальна інформація про алгоритми вирішення VRP

Ефективне планування маршрутів дозволяє покращити обслуговування клієнтів та зменшити витрати на перевезення товарів. Вивчення задачі

маршрутизації транспорту та методів її вирішення необхідні для розробки програм, здатних враховувати різні обмеження та забезпечувати оптимальне використання ресурсів при транспортуванні товарів [17].

Існує багато методів для вирішення задачі маршрутизації транспорту: точні алгоритми, які забезпечують оптимальні рішення, та методи наближеного рішення (евристичні та метаевристичні алгоритми), результатом яких є близькі до оптимальних рішення [18].

Точні алгоритми спрямовані на знаходження оптимального рішення, перевіряючи всі можливі комбінації маршрутів та використовуючи техніки розгалуження та обмеження для скорочення простору пошуку [19]. Мають велику обчислювальну складність для задач великої розмірності, адже при збільшенні кількості клієнтів обчислювальні витрати експоненційно зростають. Точні алгоритми призначені для знаходження оптимального розв'язку в конкретних умовах, тому вони менш адаптивні до змінних умов або до різних обмежень у порівнянні з наближеними методами [19, 20].

Евристичні методи проводять обмежений пошук у просторі рішень, і знаходять наближене рішення за невеликий обчислювальний час, використовуючи інтуїтивні підходи або правила для побудови розв'язку. Класичним прикладом є жадібний алгоритм. Цей клас можна розділити на дві категорії: конструктивна евристика та евристика покращення [20].

Конструктивна евристика (евристика побудови) починається з порожнього рішення. Повне рішення розраховується поетапно, коли кожен компонент додається по одному до часткового рішення на основі певних правил. Евристики побудови рішення використовуються для ініціалізації маршрутів, які потім можуть бути покращені іншими методами для досягнення оптимальних результатів [21].

Алгоритми евристики покращення вирішують задачу маршрутизації транспортних засобів шляхом оптимізації вже побудованих початкових маршрутів. Евристика покращення ітеративно покращує існуюче рішення маршрутизації (знайдене за допомогою конструктивної евристики),

виконуючи локальний пошук у просторі для зменшення загальної вартості рішення. Поліпшення рішення може включати зміну порядку клієнтів у маршрутах або об'єднання маршрутів з метою зменшення загальної довжини шляхів або вартості [21].

Евристика швидко створює рішення незалежно від розміру задачі, і її легко реалізувати в різних варіантах VRP. Вона ефективна у визначенні локального оптимуму. Такі алгоритми в процесі роботи відсікають велику кількість непотрібних (заздалегідь неоптимальних) рішень, що суттєво впливає на їхню швидкодію при розв'язання NP-складних завдань. Основне обмеження полягає у тому, що алгоритм може застрягати у локальному оптимумі. Також, остаточне рішення залежить від початкової точки локального пошуку та має розрив з оптимальним рішенням. Евристичні методи не завжди здатні виявити глобально оптимальний маршрут, а відмінність у виборі початкових умов може привести до різних варіантів розв'язку [20, 21].

Для подолання обмежень евристичних алгоритмів у процесі пошуку оптимізованого рішення у локальній області, запроваджено метаевристичні алгоритми. Метаевристика – це евристичний алгоритм високого рівня, який об'єднує різні алгоритми евристики в одну загальну структуру, та використовує можливості цих алгоритмів для обчислення ефективних рішень. Метаевристичні методи спрямовані на глобальний пошук розв'язків за допомогою рандомізації, що дозволяє їм уникати застрягання в локальному оптимумі, а також можуть спрямовувати евристичний алгоритм у просторі пошуку.

Деякі метаевристичні алгоритми можна розбити на незалежні частини, які можуть виконуватися одночасно (паралельно), що дозволяє їм працювати швидше. Ці методи є адаптивними до змінних умов і можуть ефективно пристосовуватися до різноманітних обмежень та умов задачі. Але, налаштування параметрів методів потребують спеціальних знань, і вибір неоптимальних параметрів впливає на якість розв'язку.

Серед найбільш застосовуваних метаевристик можна виділити: генетичний алгоритм (Genetic Algorithm), алгоритми мурашиної колонії (Ant Colony Algorithm), імітації відпалу (Simulated Annealing) та пошук із заборонами (Tabu Search) [20-22].

Оскільки завдання маршрутизації транспорту належить до класу NP-повних завдань теорії обчислень, є необхідність у застосуванні наближених алгоритмів для вирішення завдання [23]. При розв'язанні задач з використанням евристики та метаевристики не існує обмежень на кількість клієнтів. Евристичні та метаевристичні підходи ефективно обробляють велику кількість обмежень та отримують майже оптимальні рішення. Конструктивна евристика застосовується завдяки її швидкості та ефективності отримання швидких якісних рішень. Хоча евристичні алгоритми можуть бути швидкими та простими у реалізації, для задач з великою кількістю клієнтів та складними обмеженнями використання метаевристичних методів більш доречно, оскільки вони включають здатність ефективно обробляти великі простори пошуку та адаптивність до змін у задачі. Метаевристики гнучкіші, часто використовуються в широкому спектрі завдань, але можуть вимагати більше часу на обчислення.

1.4 Аналіз існуючих методів розв’язку CVRPTW

Основна складність задач лінійного програмування, до яких відноситься VRP, полягає в їх розмірності, що характеризується великою кількістю змінних та обмежень. Обсяг пам'яті та час вирішення збільшуються експоненційно у міру додавання змінних. Тому для того, щоб знаходити наближене до оптимального рішення за менший час, розроблені евристики та метаевристики. Для складних проблем наближені підходи пропонують найкращий компроміс між якістю рішення та обчислювальним часом [24].

Розглянуто найбільш відомі алгоритми побудови оптимальних маршрутів для транспорту, а саме: класичний евристичний алгоритм –

жадібний, та метаевристичні алгоритми – генетичний та мурашиний алгоритми, симуляція відпалу та табу-пошук.

Виявлено, що жадібний алгоритм гарантує швидке знаходження рішення за рахунок вибору локально найкращого варіанту на кожному кроці, проте це не завжди веде до отримання глобально оптимального результату. Основною перевагою цього підходу є простота та швидкість виконання, але недоліком є те, що знайдене рішення зазвичай є субоптимальним, оскільки алгоритм не аналізує всі можливі варіанти [25, 53].

Генетичний алгоритм, як представник метаевристичних підходів, базується на принципах природної еволюції, використовуючи мутації та відбір найкращих рішень з покоління в покоління. Даний алгоритм є більш гнучким і здатен знайти якісні рішення при складних умовах та у великих просторах, однак його недоліком є тривалий час виконання, оскільки потрібно багато ітерацій для досягнення оптимального результату [26, 53].

Мурашиний алгоритм використовує модель поведінки колонії мурах для пошуку оптимальних шляхів і знаходить ефективні шляхи для проблеми комівояжера та інших задач оптимізації. Перевагою алгоритму є можливість ефективного використання паралельних процесів, але значна кількість параметрів налаштування та потреба в довготривалій адаптації роблять його складним у застосуванні на практиці [26, 53].

Алгоритм симуляції відпалу імітує процес охолодження металу і знаходить глобальний мінімум системи шляхом випадкових змін станів. Його налаштування простіше порівняно з іншими метаевристичними методами, однак він залежить від вибору кількості ітерацій та функції зниження температури, що впливає на якість рішення і час виконання [26, 53].

Табу-пошук є ефективним методом локального пошуку з додатковою пам’яттю для уникнення повторних перевірок однакових рішень. Він працює в складних середовищах та забезпечує стабільність розв'язків у динамічних та складних умовах, але його основний недолік – складність в налаштуванні та висока обчислювальна витратність для великих задач [27, 54].

1.5 Постановка задачі

У рамках проекту досліджується ефективність наближених алгоритмів для вирішення задачі маршрутизації транспортних засобів з додатковими обмеженнями. В основі дослідження лежить завдання, у якому вводяться обмеження такі, що транспортний засіб має конкретну місткість, і кожен клієнт може бути обслужений лише у індивідуальний проміжок часу (CVRPTW). До класичного опису завдання додаються нові правила та елементи даних двох видів задачі VRP – CVRP та VRPTW. Мета завдання полягає у тому, щоб у результаті визначити набір маршрутів для парку ідентичних транспортних засобів, які повинні обслужити набір клієнтів, розташованих на певних точках, з відомим попитом та часовими вікнами для обслуговування. Усі транспортні засоби виходять із центрального складу та повинні повертатися назад після виконання свого маршруту.

Рішення CVRPTW зводиться до інших відомих комбінаторних задач оптимізації, включаючи задачу комівояжера (Traveling Salesman Problem, TSP) і завдання упаковки в контейнери (Bin Packing Problem, BPP) [28].

Для математичного опису завдання CVRPTW можна представити у вигляді орієнтованого графа $G = (N, E)$, де $N = \{0, \dots, n\}$ – множина вершин, E – множина дуг, що з'єднують вершини графа. V – множина транспортних засобів з однаковою вантажопідйомністю Q. Вирази $C = \{1, \dots, n\}$ та $D = \{1, \dots, d_n\}$ характеризують множину клієнтів та множину запитів клієнтів відповідно. Умова належності $(i, j) \in E$ показує дугу, що з'єднує вершину i з вершиною j. Елементи $t_{i,j}$ та $c_{i,j}$ означають час перевезення дугою (i, j), з урахуванням часу обслуговування клієнта i, та вартість перевезення вантажу від клієнта i до клієнта j відповідно. Інтервал $[a_i, b_i]$ позначає часовий проміжок, у який повинен бути обслужений i-ий клієнт. Елемент s_i^k описує час прибуття транспортного засобу до клієнту (час відправлення будь-якого транспортного засобу з депо завжди дорівнює 0). Рівняння $x_{i,j}^k = \{0,1\}$ описує змінну, що характеризує напрямок руху, яка приймає значення 1, якщо

транспортний засіб рухається з i до j, та значення 0 транспортний засіб рухається з j до i. Зазвичай прийнято припускати постійну швидкість, щоб відстань, час проїзду та витрати на проїзд вважалися однаковими. Рішенням задачі є розбиття множини N на підмножини (маршрути) та указання порядку обходу на кожній підмножині (перестановка вершин маршруту). Рішення є прийнятним, якщо всі маршрути задовольняють додаткові обмеження завдання. Таким чином, з урахуванням прийнятих позначень, CVRPTW зводиться до задачі лінійного програмування. При $a_0 = 0, s_{0,k} = 0\ \forall k$:

$$\min \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{i,j} x_{i,j}^k, \tag{1.1}$$

при обмеженнях

$$\sum_{k \in V} \sum_{j \in N} x_{i,j}^k = 1,\ i \in \{1, \dots, n\}, \tag{1.2}$$

$$\sum_{i=1}^{n} d_i \sum_{j \in N} x_{i,j}^k \leq Q,\ \ i \in \{1, \dots, n\}, \tag{1.3}$$

$$\sum_{j \in N} x_{0,j}^k = 1, \forall k \in V, \tag{1.4}$$

$$\sum_{i \in N} x_{i,h}^k - \sum_{j \in N} x_{h,j}^k = 0, \forall h \in \{1, \dots, n\}, \forall k \in V, \tag{1.5}$$

$$\sum_{i \in N} x_{i,n}^k = 1, \forall k \in V, \tag{1.6}$$

$$x_{i,j}^k \left(s_i^k + t_{i,j} - s_j^k\right) \leq 0, \forall i, j \in N, \forall k \in V, \tag{1.7}$$

$$a_i \leq s_i^k \leq b_i, \forall i \in N, \forall k \in V\ , \tag{1.8}$$

$$x_{i,j}^{k} \in \{0,1\}, \forall i, j \in N, \forall k \in V, \quad (1.9)$$

$$\sum_{k \in V} \sum_{j \in N} x_{0,j}^{k} \leq |V|, \forall j \in N, \forall k \in V. \quad (1.10)$$

Цільова функція (1.1) мінімізує загальні витрати на перевезення товару. Рівняння (1.2) показує відвідування кожного клієнта один раз. Нерівність (1.3) свідчить про обмеження вантажопідйомності всіх транспортних засобів. Рівняння (1.4), (1.5) та (1.6) позначають відправлення з депо, переміщення між клієнтами та повернення в депо відповідно. Нерівність (1.7) пов'язує між собою час відправлення з пункту та час прибуття до іншого пункту. Подвійна нерівність (1.8) встановлює часові рамки. Вираз (1.9) – це обмеження цілісності, а нерівність (1.10) означає обмеження використання певного числа транспортних засобів [28].

Головним завданням є мінімізація загальної вартості маршрутів, яка включає: загальну довжину маршруту транспортних засобів, час у дорозі, включаючи можливе очікування для дотримання часових вікон, та число задіяних транспортних засобів. Відповідно до 1.2-1.10 при вирішенні задачі необхідно дотримуватись таких умов та обмежень:

- обмежена місткість транспортних засобів – загальний попит клієнтів, які обслуговує один транспортний засіб, не повинен перевищувати його вантажопідйомність;

- часові вікна для обслуговування – кожен клієнт повинен бути обслугований у певний проміжок часу. Рішення є неприйнятним, якщо клієнт обслуговується після верхньої часової межі. Машина, що прибула раніше нижнього межі, чекає на її настання;

- маршрути транспортних засобів – кожен автомобіль повинен почати і завершити свій маршрут у центральному депо, при цьому кожен клієнт повинен обслуговуватись один раз.

Для вирішення завдання використовуються наближені методи оптимізації маршрутизації транспорту, а саме: жадібний, генетичний, мурашиний алгоритми, алгоритми імітації відпалу та табу-пошуку. Для

розробки програми необхідно виконати наступні завдання:

- ознайомитися з існуючими рішеннями в області маршрутизації транспорту, вивчити принцип роботи подібних рішень;
- проаналізувати алгоритми, які використовуються для вирішення задач пошуку оптимальних маршрутів на графі, включаючи обмеження по часу та вантажопідйомності;
- визначити структуру файлу для зберігання вхідних даних, включаючи інформацію про клієнтів, депо та інших елементів обмежень задачі;
- оптимізувати обрані алгоритми для роботи з великими графами з метою підвищення швидкості обчислень;
- обрати технології для створення користувацького інтерфейсу;
- вибрати відповідні елементи керування для розробки інтерфейсу застосунків, що відповідає поставленій задачі;
- виконати розробку і відлагодження програми у середовищі Microsoft Visual Studio 2022;
- реалізувати функції, необхідні для коректної роботи програми, включаючи побудову маршрутів з урахуванням обмежень;
- створити зручний інтерфейс для користувача з можливістю зберігання і завантаження результатів та налаштувань вибору алгоритмів для тестування;
- провести аналіз отриманих результатів;
- тестування програми на різних наборах вхідних даних, перевіряючи роботу алгоритмів, щоб переконатися в його коректності та ефективності.

У рамках роботи необхідно розробити тестове програмне середовище, яке дозволяє реалізувати задані алгоритми пошуку набору маршрутів на спеціально підготовлених вихідних даних та візуалізувати розв'язання задачі у вигляді графу. Візуалізація повинна включати відображення маршрутів транспортних засобів, розташування клієнтів та дотримання часових вікон.

2 АНАЛІЗ АЛГОРИТМІВ ТА ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

Різноманітність умов для вирішення задачі CVRPTW вимагає детального аналізу поширених алгоритмів пошуку оптимальних маршрутів у графі, для аналізу їхньої ефективності та можливості обрати алгоритми для врахування конкретних обмежень і вимог задачі, забезпечуючи мінімальні витрати часу і ресурсів на доставку.

Для вирішення задачі CVRPTW використовуються жадібні евристики, спрямовані на швидке знаходження оптимальних маршрутів у графах, та більш гнучкі алгоритми класу стохастичної оптимізації (метаевристики) як альтернатива евристиці для поліпшення ефективності рішень у задачах CVRPTW.

2.1 Жадібна евристика

Жадібний алгоритм – це простий евристичний підхід, який на кожному кроці обирає найкраще з доступних рішень, не враховуючи можливих наслідків для подальших етапів пошуку. Наприклад, для задачі комівояжера жадібна стратегія може полягати у виборі найближчого з невідвіданих міст на кожному кроці [29].

Задачі, до яких застосовуються жадібні алгоритми, зазвичай характеризуються двома властивостями: можливістю жадібного вибору та оптимальною підструктурою [30]. Якщо локально оптимальні рішення приводять до глобального оптимального розв'язку, до такої оптимізаційної задачі можна застосувати принцип жадібного вибору. Оптимальність жадібного алгоритму доводиться так: спочатку перевіряється, що початковий жадібний вибір сприяє пошуку оптимального рішення; для кожного рішення існує альтернативне йому рішення, що узгоджується з жадібним вибором і не є гіршим. Потім показується, що підзадача, що утворилася після першого

вибору, аналогічна початковій, і так продовжується доведення методом індукції. На відміну від динамічного програмування, жадібний алгоритм не повертається до вже прийнятих рішень для їх перегляду [29].

Класична жадібна стратегія починається з порожнього набору і поступово додає до нього вершини, доки всі вершини графа не виявляться пов'язаними з цим набором. На кожному кроці алгоритм вибирає вершину з найбільшою кількістю непокритих сусідніх вершин і включає її у набір. Повторення процесу до покриття графа дає бажану множину, пов’язану з набором. Ця жадібна стратегія може бути узагальнена для пошуку k-охоплюючих множин у простих графах шляхом ітеративної перевірки вершин, які ще не k-покриті в графі [30]. Жадібний алгоритм для вирішення задачі маршрутизації транспорту будує маршрути, додаючи на кожному кроці найближчого клієнта, якого можна обслугувати без порушення обмежень залежно від умови задачі. Кожен транспортний засіб починає маршрут із початкової точки (складу), а як тільки неможливо додати нових клієнтів через обмеження, маршрут завершується, і алгоритм переходить до нового транспортного засобу або маршруту. Такий підхід швидко створює маршрути, але не гарантує оптимальний розв'язок через локально-оптимальні вибори на кожному етапі [30].

2.2 Метаевристика

Метаевристикою називається «інтелектуальна стратегією, що поєднує підлеглі евристики для диверсифікації та інтенсифікації» [31]. Метаевристики поділяються на дві основні групи: засновані на одному рішенні та методи засновані на популяції. Обидві категорії відрізняються способами отримання найкращого рішення.

Методи, орієнтовані на одне рішення (також відомі як методи, засновані на сусідстві чи локальному пошуку) виконують ітеративні низькорівневі пошуки, щоразу оптимізуючи одне поточне рішення,

неодноразово переходячи до сусідніх рішень. Тобто низькорівневий пошук є евристикою покращення. При вирішенні задачі маршрутизації важливим є те, як спрямовувати пошук для уникнення локальних оптимумів та підтримувати дослідження всього простору рішень. Для цього застосовуються адаптивні високорівневі правила, які проектуються трьома способами: множинні запуски, зміна ландшафту та розробка правил прийняття [31, 32].

У першому підході локальний пошук запускається з кількох початкових точок, використовуючи різні стратегії, такі як багаторазові запуски, зміна початкових точок у процесі оптимізації та їх комбінування. У другому підході пошук налаштовується модифікацією ландшафту придатності, змінюючи його під час оптимізації або використовуючи альтернативні ландшафти. У контексті метаевристичних алгоритмів зміна ландшафту означає зміну оцінної функції (або функції придатності), що визначає якість рішень.

Ландшафт пошуку – це метафора простору, де рішення представлені як точки, а висоти чи глибини (значення функції) показують їхню «придатність» чи якість. У третьому підході напрям пошуку регулюється розробленими критеріями приймання, виключаючи непродуктивні рішення або допускаючи найгірші рішення з певною ймовірністю [32]. Алгоритми локального пошуку фокусуються на оцінці та зміні одного або кількох «поточних станів» замість систематичного і послідовного дослідження всіх можливих шляхів, що виходять з «початкового стану». У цих методах мета полягає в тому, щоб знайти найкращий стан на основі цільової функції.

Методи, засновані на сусідстві, можуть потрапити в пастку локальних оптимумів, що призводить до неоптимальних рішень [33]. Термін «пастка локальної оптимальності» вказує на ситуацію в оптимізаційних завданнях, коли алгоритм оптимізації впадає в локально оптимальне розв'язання, яке може бути найкращим у межах обмежень певної області, але не є глобально найкращим розв'язком для всієї задачі оптимізації. Метаевристика на основі популяції відноситься до групи алгоритмів оптимізації, які використовують

популяцію потенційних рішень для дослідження простору пошуку та знаходження оптимального рішення. Еволюційні обчислення та ройовий інтелект – два основні підходи до виконання пошуку на основі популяції.

Еволюційні обчислення використовують концепцію природної еволюції і застосовують генетичні оператори, такі як відбір, мутація і кросинговер, щоб дозволити популяції розвиватися з часом. Індивідууми, які сильніші (пристосовані до навколишнього середовища), ніж конкуренти, з більшою ймовірністю зроблять потомство, яке зможе краще вижити [32, 34].

Ройовий інтелект – це область досліджень, яка вивчає природні та штучні системи, що складаються з множини індивідуумів, які співпрацюють один з одним. Ройовий інтелект імітує колективну поведінку соціальних організмів, таких як бджоли, птахи, мурахи, риби та включає взаємодії між популяцією та навколишнім середовищем для досягнення оптимального рішення. Алгоритми ройового інтелекту швидше і ефективніше працюють при дослідженні простору пошуку, але вони можуть не «сходитись» до глобального оптимуму або застрягати в локальному оптимумі. Алгоритмам еволюційних обчислень може знадобитися більше часу для збіжності, але вони мають більшу надійність і можуть вирішувати складні завдання оптимізації, які включають велику кількість змінних [32, 34].

Для дослідження обох категорій метаевристичних алгоритмів у роботі розглянуто чотири методи, а саме генетичний та мурашиний алгоритми для представлення метаевристик на основі популяції та імітований відпал з табу-пошуком для представлення метаевристик на основі одного рішення.

2.2.1 Генетичний алгоритм

Генетичний алгоритм (Genetic algorithm, GA) – це алгоритм популяційного пошуку, який використовує механіку природного відбору та природної генетики для ітеративного розвитку популяції можливих рішень. Алгоритм заснований на припущенні, що природна еволюція підтримує

баланс між різноманітністю популяції та адаптивністю [35, 41, 44].

Рішення GA виражені як генетичний код (хромосоми) агенту, що є унікальним для кожного агенту. GA включає набір агентів, які здатні породжувати кращі рішення, використовуючи різні стратегії пошуку оптимуму: кросинговер, мутацію, селекцію. Існує велика кількість різних модифікацій GA, відмінності яких полягають у поданні хромосом та реалізації генетичних операторів. Генетичні алгоритми працюють із популяцією потенційних рішень, що дозволяє їм проводити багатоваріантний пошук одночасно, збільшуючи ймовірність знаходження оптимального рішення [42].

Генетичний алгоритм починає з ініціалізації популяції, створюючи випадковим чином множину індивідуумів, кожен з яких представляє можливе рішення. Після цього виконується декодування цих рішень і на підставі цього використовується налаштована функція пристосованості для оцінки пристосовуваності кожної особини [35, 36].

Функція придатності GA вимірює якість рішення на основі цілей мінімізації загальної пройденої відстані, балансування робочого навантаження між транспортними засобами та врахування обмежень пропускної спроможності. Оцінка придатності включає розрахунок загальної відстані, пройденої кожним транспортним засобом, з урахуванням послідовності відвідуваних клієнтів [35, 36].

На наступному етапі застосовується операція селекції (відбір), яка вибирає особини з найкращим пристосуванням для подальшого розмноження. У цьому контексті пристосованість означає ступінь корисності чи якості, які необхідно максимізувати у процесі пошуку рішень. Тому більш пристосовані особини мають більший шанс бути відібраними для розмноження, що у свою чергу покращує загальну якість популяції [35]. Вибрані особини піддаються кросинговеру, під час якого відбувається комбінування генетичних матеріалів батьків на створення нового потомства. Цей процес дозволяє успадковувати ознаки батьків і, як наслідок, виробляти

більш адаптованих нащадків [35, 36].

Щоб додатково підвищити генетичну різноманітність, до кожного потомства застосовується мутація. Цей процес включає випадкові зміни в генетичному коді окремого індивідуума, що сприяє дослідженню простору рішень та допомагає підтримувати чи збільшувати генетичну різноманітність. Імовірність виникнення мутації невелика, але ця випадковість дозволяє уникнути застрягання в локальних оптимумах та сприяє глибшому дослідженню простору рішень [35, 36].

Якщо буде досягнуто максимальну кількість поколінь або виникне збіжність до однорідної популяції, що включає схожі індивідууми, або отримано оптимальне рішення, цикл виконання алгоритму завершується. Після завершення алгоритму найкраща хромосома декодується, надаючи відповідне рішення задачі.

Класична структура генетичного пошуку виявляється недостатньою для задач комбінаторної оптимізації, тому поєднання GA з різними методами пошуку є ефективним способом, що допомагає традиційному GA краще вирішувати задачі маршрутизації великої розмірності та з варіативними обмеженнями. Інтегровані методи включають рій частинок, імітацію відпалу та жадібний алгоритм [32].

2.2.2 Імітований відпал

Імітований відпал (Simulated Annealing Algorithm, SA) – це алгоритм стохастичного пошуку, заснований на процесі відпалу твердих тіл. Це різновид алгоритму «сходження на пагорб» (hill-climbing) із вбудованою здатністю виходити за межі локальних оптимумів у пошуковому просторі. Процес відпалу походить від фізичного процесу кристалізації в твердих тілах, де система намагається досягти мінімальної енергії за допомогою нагрівання матеріалу з поступовою процедурою повільного охолодження, що дозволяє атомам стабілізуватися у щільній та енергоефективній кристалічній

структурі (рисунок 2.1). Таким чином, оптимальне рішення задачі відповідає мінімальному енергетичному стану [37].

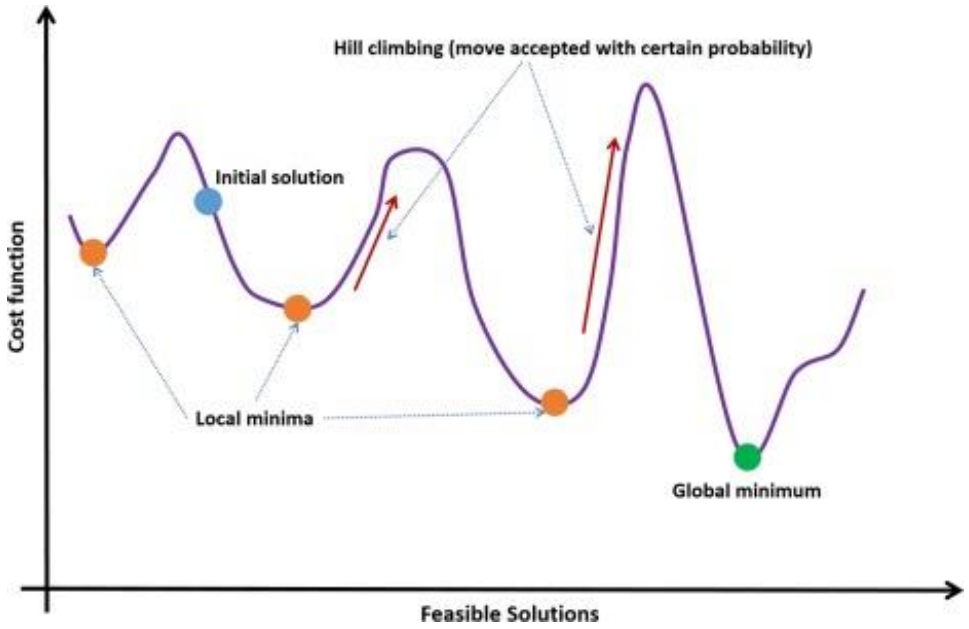

Рисунок 2.1 – Графічне представлення процесу відпалу

У процесі пошуку рішення в просторі можливих розв'язків, атоми ітеративно переміщуються між позиціями з певною ймовірністю, яка знижується разом із температурою. Початкові налаштування температури і темпу її зниження утворюють графік відпалу.

У SA для рішення задачі маршрутизації генерується початкове рішення та задається температурний параметр. Створюється структура сусідства, що визначає набір можливих переміщень з поточного рішення. Для кожного сусіда обчислюється вартість (або рівень придатності). Якщо рішення сусіда дає поліпшення цільової функції, це рішення приймається. В іншому випадку застосовується критерій прийнятності, що визначає чи приймати нове рішення на основі значення його цільової функції і поточної температури. Алгоритм використовує ймовірнісну функцію для прийняття гіршого рішення (імовірність прийняття нових рішень буде високою) і поступово фокусується на поліпшенні цільової функції, зменшуючи імовірність шляхом

зниженням температури. Цей процес триває, поки температура не наблизиться до нуля або покращення не стануть неможливими. Це є завершальним етапом відпалу, коли атоми досягають стабільного кристалічного стану [38, 41, 44].

У процесі відпалу, якщо температура падає занадто швидко, це може призвести до утворення твердої аморфної структури з високою внутрішньою енергією. У задачі оптимізації це еквівалентно одержанню локально оптимального розв'язання задачі. Для VRP загальні структури сусідства включають обмін клієнтами між маршрутами або вставку клієнта з одного маршруту до іншого маршруту зі збереженням обмежень пропускної спроможності та часового вікна [38, 41].

2.2.3 Алгоритм мурашиної колонії

Оптимізація колоній мурах (Ant Colony Optimization, ACO) – це алгоритм, який є різновидом популяційного методу оптимізації ройового інтелекту. Мурашині алгоритми використовуються для знаходження наближених рішень різних комбінаторних задач оптимізації: комівояжера, пошуку маршрутів на графах, ранця, призначення, а також завдань складання розкладів. Використання оптимізації колонії мурах моделює поведінку мурах при пошуку їжі для розв'язання задачі маршрутизації транспорту. Один із прикладів схожого підходу до рішення – алгоритм бджіл, заснований на принципах організації роботи медоносних бджіл, що також належать до соціальних комах [39].

У процесі розв’язання комбінаторної оптимізаційної задачі за допомогою ACO, задачу представляється як пошук найкоротшого маршруту в зваженому графі. ACO ініціюється випадковими пересуваннями мурах при пошуку шляхів розв'язання. Штучні мурахи ітеративно створюють рішення шляхом переходу від одного клієнта до іншого, відкладаючи на своєму шляху хімічний слід (феромон), який потім використовується для

спрямування наступних рухів мурах. Чим більше мурах проходить певним шляхом, тим більше феромону залишається на цьому шляху. Отже, імовірність вибору шляху іншими мурахами збільшується [40, 41, 42]. В результаті, коли мураха виявляє оптимальний (короткий) маршрут від колонії до джерела їжі, інші мурахи швидше починають його використовувати (рисунок 2.2), і завдяки позитивному зворотному зв'язку в результаті весь потік мурах вибирає цей маршрут.

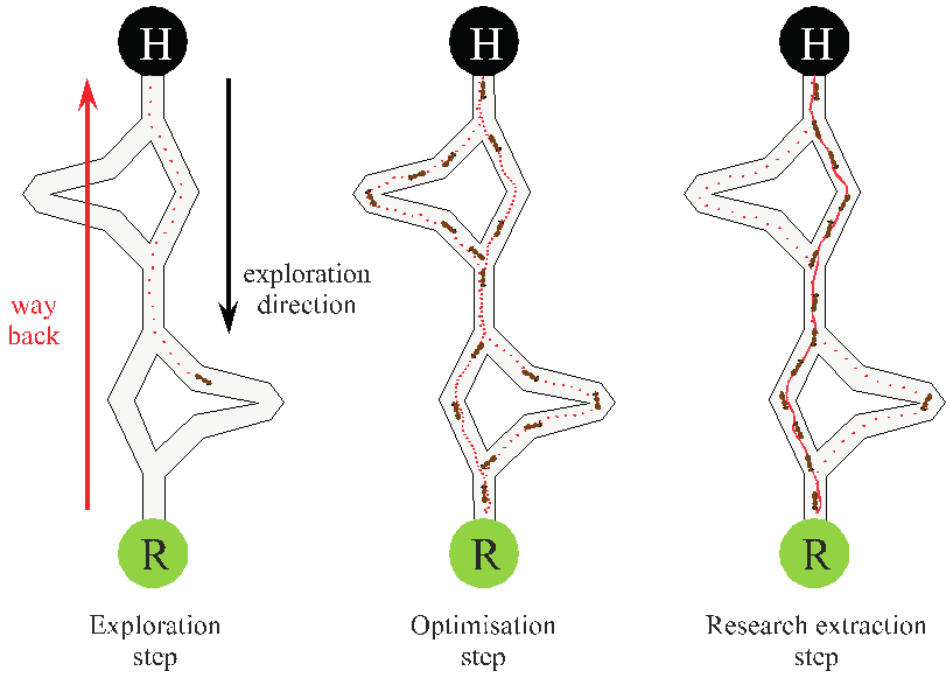

Рисунок 2.2 – Схема роботи мурашиного алгоритму

Мурахи ймовірно обирають наступного відвідувача на основі рівнів феромонів та евристичної інформації (відстані) до сусідніх клієнтів. Механізми випаровування та посилення феромонів забезпечують динамічне оновлення феромонних слідів [40, 41]. Випаровування згодом знижує рівень хімічного сліду, щоб уникнути отримання неоптимальних рішень, тоді як посилення збільшує феромонний слід для якісних рішень, знайдених мурахами. Отже, оптимальним маршрутом буде той, де концентрація феромону найвища.

Для більш складних версій VRP розроблено систему множинних колоній мурах, яка організована з ієрархією штучних колоній мурах, розроблених для послідовної оптимізації множинної цільової функції: перша колонія мінімізує кількість транспортних засобів, а друга колонія мінімізує пройдені відстані. Співпраця між колоніями здійснюється шляхом обміну інформацією за допомогою оновлення феромонів [40].

2.2.4 Пошук з заборонами

Табу-пошук (Tabu Search, TS) – це метаевристичний алгоритм, який використовує стратегію пошуку на основі пам'яті для дослідження простору рішень. Пошук з табу уникає потрапляння в пастку локального оптимуму та спрямовує пошук у певний напрямок [43].

Початкове рішення створюється за допомогою евристики вставки. Після створення початкового рішення робиться спроба покращити його за допомогою локального пошуку різними способами визначення сусідніх рішень і вибором оптимального варіанту. Пошук табу ітеративно досліджує поточне рішення, допускаючи ітерації, які покращують цільову функцію або порушують певні умови табу [37, 44]. На відміну від традиційних методів спуску, у табу-пошуку поточне рішення може ставати гіршим з однієї ітерації до іншої.

Головним елементом алгоритму є списки табу, які відслідковують останню історію пошуку та не дозволяють пошуку повернутися до раніше відвіданих рішень. Щоб уникнути циклічності, рішення з певними характеристиками нещодавно досліджених рішень тимчасово оголошуються забороненими чи недоступними. Табу-територія – період, протягом якого атрибут зберігає статус заборони, і цей термін змінюється в залежності від інтервалу часу. Заборонений статус можна зняти під час виконання певних умов. Такий підхід називається критерієм устремління та застосовується, коли рішення з накладеними заборонами краще за всі раніше знайдені

варіанти. Нове рішення не приймається, якщо воно не допомагає уникнути вже пройденого шляху. Цей процес гарантує, що нові області простору рішення будуть досліджені з метою уникнути локальних мінімумів і зрештою знайти бажане рішення [37, 43].

2.3 Технології збереження даних

2.3.1 Опис вихідних даних у форматі VRP

Формат VRP – це стандарт для представлення даних, що використовуються в задачах маршрутизації транспортних засобів. Формат VRP зберігає дані про клієнтів, депо та інші параметри задачі у структурований вигляд. Структура формату VRP може змінюватись в залежності від використовуваного програмного забезпечення та конкретних вимог завдання.

Класично VRP-файли містять таку інформацію:

- загальні параметри завдання: кількість транспортних засобів, максимальна місткість транспортних засобів (якщо застосовується), кількість клієнтів (точок доставки), тип завдання (VRPTW, CVRP);
- список координат вузлів (депо та клієнтів): депо зазвичай задається окремим рядком або першими координатами у списку, для кожного клієнта вказуються його координати (у 2D-просторі, але формат може підтримувати і 3D-простір);
- матриця чи таблиця відстаней: якщо відстані між вузлами обчислені заздалегідь, вони можуть бути представлені у вигляді симетричної або асиметричної матриці;
- запити клієнтів (demand) – кількість вантажу або послуги, яка потрібна для кожного клієнта;
- обмеження (опціонально): часові вікна обслуговування клієнтів, максимально допустимий час маршруту, вартість або час обслуговування;

- додаткові параметри (опціонально): тип оптимізованої функції (мінімізація відстані, витрат або часу), допустимі швидкості транспорту, географічні чи топологічні особливості [45].

Файли формату VRP дозволяють тестувати алгоритми на репрезентативних завданнях реального світу, забезпечуючи об'єктивні метрики для порівняння (вартість та час проїзду маршрутів). Популярні бенчмарки:

- Benchmarks від Solomon (VRPTW) – містить дані про часові вікна, вимоги клієнтів та відстані;
- Augerat CVRP Instances – використовуються для тестування рішень CVRP;
- хрестоматійні приклади Christofides – приклади для стандартних TSP і CVRP завдань [45].

2.3.2 Зберігання результатів у форматі PDF

Формат PDF (Portable Document Format) – це універсальний формат файлів, розроблений компанією Adobe Systems у 1993 році. Основна мета створення PDF – забезпечити надійне та незмінне відображення документів на різних пристроях та платформах. PDF-файли можуть містити текст, зображення, векторну графіку, а також інтерактивні елементи, такі як гіперпосилання та форми [46]. PDF-файли широко використовуються в різних сферах, включаючи бізнес, освіту та видавничу справу. Файли PDF створюються за допомогою таких інструментів, як Adobe Acrobat або іншого програмного забезпечення [47].

До переваг формату PDF відносяться:

- стандартизованість та популярність: файли PDF відкриваються на будь-яких пристроях із будь-якими операційними системами, тому PDF доцільно використовувати для обміну документами між користувачами з різними пристроями;

- збереження форматування: документи у форматі PDF зберігають своє початкове форматування, включаючи шрифти, зображення та макет. Це важливо для документів, де точне розташування елементів дотримується деяких правил, наприклад, для звітів, презентацій та маркетингових матеріалів;

- PDF-файли можуть бути стиснуті без втрати якості, що робить їх зручними для зберігання та передачі;

- безпека: користувач може налаштувати параметри безпеки для свого PDF-файлу, наприклад, встановити пароль, шифрування, заборону друку та редагування, використання електронного підпису для визначення справжності документа, що корисно при передачі важливих даних, таких як фінансові звіти або юридичні документи;

- інтерактивність: підтримка гіперпосилань, закладок, форм та інших інтерактивних елементів. Це дозволяє створювати динамічні та зручні для користувача документи, які можуть включати посилання на зовнішні ресурси, інтерактивні форми для заповнення.

При використанні формату PDF можуть виникнути складності, тому слід зазначити такі недоліки:

- редагування PDF-файлів може бути складним, особливо без спеціалізованого програмного забезпечення;

- поширені редактори PDF є платними;

- розмір файлів: хоча PDF-файли можуть бути стиснуті, вони можуть займати більше місця, ніж інші формати, такі як TXT або DOCX.

Деякі старі пристрої або програмне забезпечення можуть не підтримувати нові версії PDF. Це може спричинити проблеми при спробі відкрити або редагувати документ на застарілому обладнанні [46, 47].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

Для проектування структури програми за основу використовується багатошарова архітектура (Layered Architecture). Багаторівнева архітектура або N-рівнева архітектура є фундаментальним шаблоном проектування програмного забезпечення, який організує застосунок у кілька шарів, кожен з яких відповідає за певні функції. Такий підхід покращує поділ завдань, спрощуючи управління, тестування та розширення програми. Кожен шар в архітектурі взаємодіє тільки з шаром, розташованим безпосередньо над ним або під ним [48]. Використовуються такі шари:

- рівень презентації (presentation layer) – це перший і найвищий рівень у застосунку, на якому користувачі можуть взаємодіяти з програмою;
- шар бізнес-логіки (business logic layer) – це середній шар, який містить всю бізнес-логіку програми, описує, як бізнес-об'єкти взаємодіють один з одним, де шар уявлення та шар доступу до даних можуть опосередковано взаємодіяти один з одним;
- рівень доступу до даних (data layer) – забезпечує дотримання правил доступу до даних, надаючи спрощений доступ до даних, що зберігаються у постійному сховищі. Цей рівень фокусується лише на доступі до даних, а не на їх зберіганні [48].

Еквівалентно шарам багаторівневої архітектури структура розробленої програми виглядає так:

- шар даних представлений модулями «Model», «Graph» і «Src», який містить класи, що описують дані та їх структуру (граф, вузли, транспорт), а також набори вхідних даних;
- шар бізнес-логіки описується модулем «Algorithms» та реалізує алгоритми розв'язання задачі оптимізації;
- за шар подання відповідає MainForm, що забезпечує графічний інтерфейс користувача;

Також використовується допоміжний шар, представлений модулем «Utils», який надає допоміжні функції. За Application Layer (точку входу програми) відповідає Program.cs, який ініціалізує роботу програми та координує взаємодію між шарами.

В результаті використання підходу Layered Architecture дозволяє ізолювати логіку, дані та інтерфейс. Структура програми поділена на модулі, що забезпечує можливості для розширення та змін.

3.1 Модель представлення вхідних та вихідних даних

Множина всіх можливих маршрутів для задачі CVRPTW представлена у вигляді неорієнтованого графа, у якому вершини відповідають клієнтам або точкам доставки, а ребра – можливим шляхам між точками. Вершини графа містять інформацію про клієнтів, такі як їхні вимоги та місцеположення. Ребра графа характеризуються відстанями або часом переміщення між точками. Відстань та час необхідні на проїзд маршруту є основними критеріями для оцінки його оптимальності.

Основні сутності необхідні для зберігання даних описуються у модулі «Model», що містить класи CvrptwGraph, Node та Vehicle.

Вхідні дані для графу представлені за допомогою CvrptwGraph, що створений для зберігання основних елементів рішення у зручному форматі для роботи з алгоритмами.

Клас CvrptwGraph містить у собі наступні поля та методи:

- назву вхідного файлу, його опис та тип задачі;
- список клієнтів (вершин) графа, представлений типом Node;
- список транспортних засобів, представлений типом Vehicle;
- список депо (початкових точок маршрутів), типу Node;
- кількість транспортних засобів, депо та вершин у графі;
- обмеження місткості та максимальної пройденої відстані для кожного транспортного засобу;

- метод додавання об'єкту Node до списку депо;
- метод додавання об'єкту Node до списку клієнтів;
- метод обчислення евклідової відстані між двома вершинами (клієнтами чи депо).

Клас Node є моделю вершини графа, що використовується в задачі маршрутизації. Кожна вершина може бути споживачем чи пунктом відправлення товарів, має координати на площині, часові обмеження та додаткові властивості. Клас містить наступні дані:

- унікальний ідентифікатор вершини, що використовується для розрізнення вузлів;
- координати вершини осі X та осі Y на площині;
- необхідний обсяг доставки (попит на товар);
- час, від якого допустиме обслуговування вершини (початок часового вікна);
- час, до якого обслуговування вершини має бути завершено (кінець часового вікна);
- мітка, що вказує, чи відвідана вершина;
- лічильник, який вказує, скільки разів вершина була відвідана.

Для загального опису транспортних засобів використовується модель Vehicle. Клас Vehicle зберігає ідентифікатор транспортного засобу, поточну завантаженість (значення показує, скільки вантажу знаходиться у транспорті в даний момент), вартість, пов'язану з маршрутом транспортного засобу та список ідентифікаторів вузлів, які відвідує транспортний засіб. Також, у класі є метод, що розраховує вартість маршруту транспортного засобу, ґрунтуючись на матриці відстаней та списку всіх вершин маршруту.

3.2 Реалізація алгоритмів рішення CVRPTW

Реалізація різних алгоритмів оптимізації задачі CVRPTW знаходиться у модулі «Algorithms», а саме: алгоритм мурашиної колонії, імітації відпалу та

табу пошуку, генетичний алгоритм та жадібний алгоритм, що передбачає просту стратегію побудови маршрутів.

Також, модуль «Algorithms» містить клас Optimizations. Він містить методи для перевірки валідності змін у маршруті, обчислення довжини маршрутів, а також глобальної та локальної оптимізації маршрутів. Цей клас є допоміжним інструментом поліпшення якості рішень, знайдених іншими алгоритмами.

Метод IsValidPrecheckForGlobalOpt перевіряє можливість виконання обміну елементів між двома маршрутами з урахуванням часових обмежень. Для цього він використовує як параметри список вузлів, два різні маршрути, а також індекси вузлів, які передбачається обміняти. Метод оцінює, чи можна переміститися між вузлами у межах заданих часових вікон, та чи вміщує транспорт ще товар. Якщо умова не виконується, метод повертає false.

Метод CalcNewLenForGlobalOpt обчислює довжину маршруту після заміни одного вузла на інший. Цей метод приймає список вузлів, маршрут, індекс вузла, що змінюється, і новий вузол, який замінює поточний. Він перераховує довжину маршруту та перевіряє, чи маршрут залишається валідним з урахуванням часових вікон і місткості транспорту. Якщо маршрут стає неможливим, повертається значення нескінченності (ClassUtils.INF).

Також, використовуються методи роботи з фіктивними вузлами (нульовими) в маршрутах. Додавання нулів необхідне для коректної роботи алгоритмів, оскільки нульові вузли позначають повернення у депо в завданнях маршрутизації.

Метод GlobalSwapOptimization реалізує глобальну оптимізацію маршрутів через обмін вузлів між різними маршрутами. Він перевіряє всі можливі пари маршрутів, обчислює довжину кожного маршруту до та після обміну, і, якщо обмін покращує загальну довжину маршрутів та зберігає їхню валідність, вузли змінюються місцями.

Метод GlobalInsertOptimization відповідає за глобальну оптимізацію шляхом переміщення вузлів між маршрутами. Вузли одного маршруту

вставляються в інший, і якщо це зменшує загальну довжину і дотримуються часові обмеження і місткості, зміни зберігаються.

Метод LocalSwapOptimization реалізує локальну оптимізацію маршруту, виконуючи обміни вузлів усередині одного маршруту. Цей метод перебирає всі можливі пари вузлів та оцінює, чи покращує обмін довжину маршруту. Якщо так, зміни фіксуються.

Клас Optimizations спрямований на покращення маршрутів, мінімізацію їхньої загальної довжини та оптимізацію розподілу вузлів, одночасно дотримуючись часових обмежень.

Модуль «Utils» є службовим шаром програми та надає допоміжні функції та параметри, необхідні для реалізації алгоритмів рішення задачі. Цей модуль містить клас ClassUtils, який є утилітарним класом, що надає методи та константи, які використовуються в інших частинах проекту, що знижує дублювання коду. Цей клас включає такі компоненти:

- генерація випадкових цілих чисел у заданому діапазоні;
- генерація випадкових чисел із плаваючою точкою у заданому діапазоні. Методи генерації випадкових чисел дозволяють створювати тестові дані чи випадкові початкові варіанти рішення;
- реалізація зведення числа в цілий ступінь, що використовується при обчисленнях, пов'язаних з алгоритмами оптимізації. Це альтернатива стандартної бібліотечної функції Math.Pow, що використовується для підвищення продуктивності у специфічних випадках;
- використання констант (EPS, INF) для роботи з похибками та межами значень, що необхідно при вирішенні задач з числами з плаваючою точкою. Застосовується при розрахунках, пов'язаних із обмеженнями чи штрафами при переміщеннях;
- конвертація маршруту зі списку ідентифікаторів вузлів у список об'єктів вузлів (Node), що робить маршрути зручнішими для аналізу та візуалізації.

3.2.1 Реалізація жадібного алгоритму

Жадібний алгоритм будує маршрути транспортних засобів, починаючи з найближчого до депо вузла, який знаходиться в межах часового вікна і може бути обслужений з урахуванням поточного завантаження та пройденої транспортом відстані. Вибраний вузол додається до маршруту поточного транспортного засобу, при цьому зменшується його доступна ємність на значення попиту вузла та оновлюється загальна відстань та час маршруту. Коли додавання наступного вузла неможливе через перевищення обмежень завантаження ТЗ, часових вікон вузла та максимальної відстані маршруту, алгоритм завершує маршрут поточного транспорту і починає новий маршрут з використанням наступного.

Описаний підхід полегшує вирішення завдання маршрутизації, але не гарантує глобально оптимального рішення. Жадібний алгоритм не враховує можливі покращення маршруту за рахунок перерозподілу вузлів між транспортними засобами.

Жадібний алгоритм використовується як базовий метод для генерації початкового рішення. Основна перевага використання алгоритму у швидкості його роботи та простоті реалізації, що забезпечує доцільність при використанні для завдань невеликого масштабу.

Для виконання жадібного алгоритму маршрутизації реалізовано клас GreedySolution. Він включає два основних методи: RunMinSum і Run, кожен з яких вирішує завдання побудови маршрутів для транспортних засобів на основі наданих даних про клієнтів і депо.

Метод RunMinSum мінімізує загальну довжину маршрутів під час обслуговування клієнтів з урахуванням обмежень задачі. На початковому етапі ініціалізуються допоміжні структури даних: масиви для зберігання поточної довжини маршрутів та часу роботи транспортних засобів, список позицій кожного транспортного засобу та маршрути, побудовані для кожного транспорту.

Клієнти сортуються за часом готовності, після чого починається основний цикл, який для кожного клієнта шукає найближчий транспортний засіб, перевіряючи виконання умов з обмежень. Якщо відповідного транспортного засобу немає, метод повертає поточне рішення.

Метод Run є загальним методом для запуску алгоритму. Залежно від переданого параметра ProblemMode, обирається спосіб розв'язання задачі. Якщо завдання потребує мінімізації загальної довжини маршрутів, викликається метод RunMinSum, а альтернативному режимі виконується ітеративний бінарний пошук за часом обслуговування клієнтів для поступового поліпшення рішення, зменшуючи часові вікна та оцінюючи виконання рішення за деяку кількість ітерацій.

Також у класі GreedySolution реалізовано допоміжний метод для обміну елементів у списку, що використовується для оптимізації маршрутів. Обидва методи враховують часові вікна клієнтів, обмеження щодо довжини маршруту, місткості транспортного засобу та повертають об'єкт ProblemSolution, що містить інформацію про побудовані маршрути відвідування вузлів графу.

3.2.2 Реалізація алгоритму мурашиної колонії

Клас ACOSolution реалізує алгоритм мурашиної колонії на вирішення CVRPTW. Метод Run приймає вхідний граф задачі та параметри алгоритму у вигляді списку, такі як: параметри для управління впливом феромонів, коефіцієнт випаровування феромонів, коефіцієнти евристичних поправок, кількість ітерацій, кількість «кращих» мурах, що використовуються для оновлення феромонів, коефіцієнт обмеження списку кандидатів та обмеження за часом виконання алгоритму.

На початковому етапі ініціалізуються структури даних: матриці рівня феромонів на ребрах, ймовірностей переходу, евристик, відстаней та часових резервів та списки кандидатів для кожного вузла. Ці матриці

використовуються для оцінки якості маршрутів та ймовірностей переходів між вершинами.

Обчислюються відстані між вузлами, часові резерви та евристичні значення. Формується список кандидатів для кожного вузла з урахуванням відстаней до інших вузлів.

В основному циклі ітерацій алгоритм будує маршрути для кожного мурахи, оновлює феромони та застосовує оптимізації. На етапі побудови маршруту кожна мураха вибирає наступну вершину на основі ймовірності, обчисленої з урахуванням феромонів та евристик, перевіряючи обмеження за відстанню, часом та місткістю. Якщо додавання наступної вершини неможливе, побудова маршруту завершується, і мураха повертається до початкової точки.

Після генерації маршрутів алгоритм покращує їх за допомогою локальних (перестановка, вставка) та глобальних (заміни маршрутів) оптимізацій. Найкращі рішення оновлюють значення феромонів, реалізуючи процес «випаровування» та посилюючи ймовірність перспективних переходів.

Робота мурашиного алгоритму завершується при досягненні заданої кількості ітерацій або при перевищенні ліміту часу виконання. В результаті алгоритм повертає об'єкт ProblemSolution, що містить побудовані маршрути у вигляді списку списків вузлів, що відповідають оптимальному рішенню. Реалізація алгоритму також перевіряє недосяжні вершини та контролює обмеження задачі, щоб уникнути некоректних рішень.

3.2.3 Реалізація генетичного алгоритму

Виконання генетичного алгоритму починається з випадкового створення початкової популяції потенційних рішень (хромосом), які описують можливі варіанти розв'язання завдання. Потім кожна хромосома оцінюється за допомогою функції пристосованості. Для розмноження

(створення нового покоління, нової популяції варіантів рішень) відбираються хромосоми з найбільшою пристосованістю, генетична інформація яких об'єднується для створення нових хромосом нащадків. Потім хромосоми потомства піддаються дії генетичного оператора мутація, що вносить варіативність у популяцію.

Нова популяція хромосом оцінюється за допомогою функції пристосованості, і процес відбору, розмноження та мутації повторюється поки не буде знайдено оптимального рішення або не буде виконано критерій зупинки. Повторюючи ці кроки, генетичний алгоритм поступово наближається до оптимальних чи близьких до оптимальних рішень, керуючись принципами природного відбору та еволюції.

Для виконання генетичного алгоритму створено клас GASolution. Основні аспекти алгоритму включають створення популяції хромосом, виконання операцій схрещування, мутації та оцінки пристосованості, а також управління ітераціями та оновленням популяції.

Клас Chromosome представляє потенційне рішення у вигляді послідовності клієнтів (маршрутів) з оцінкою їхньої придатності. Конструктори класу дозволяють створювати хромосому з різних джерел, таких як списки послідовностей, маршрути чи готові рішення. Основний метод CalcFitness оцінює придатність послідовності клієнтів, перевіряючи, наскільки ефективно можна розподілити маршрути з урахуванням часових вікон, обмежень місткості та відстані.

Клас Population керує набором хромосом (популяцією) та реалізує методи додавання, видалення та сортування рішень. Він підтримує обмеження на число рішень з однаковою придатністю для запобігання передчасної збіжності в алгоритмі.

На початку роботи алгоритму ініціалізується популяція, яка містить кілька випадкових хромосом (рішень). Для цього використовуються жадний метод, алгоритм мурашиної колонії та випадкові перестановки для створення початкових маршрутів. Далі виконується цикл еволюції, що має певну

кількість ітерацій (цей лічильник можна налаштувати). В кожній ітерації популяція оновлюється шляхом виконання кросовера та мутації.

Процес створення нового покоління популяції відбувається наступним чином:

- використовуються два випадкових батьківських шляхи з поточної популяції;
- кросовер комбінує ці два шляхи в нову хромосому, вибираючи випадкову точку для розділення;
- якщо хромосома-нащадок має кращий результат (меншу вартість) за обох батьків, то вона додається до популяції. Якщо результат не покращується, то новий шлях ігнорується;
- після цього перевіряється наявність циклів у новій хромосомі, і якщо вони виявляються, то будуть видалені;
- мутація може змінити маршрути у хромосомах для покращення результату, додаючи випадкові корективи до шляху.

Після виконання операцій кросовера та мутації проводиться селекція. Вона полягає в тому, що з поточної популяції вибираються N найкращих хромосом, що мають найменшу вартість (мінімальну довжину маршруту). Інші хромосоми не враховуються.

Після завершення еволюційної ітерації, якщо кількість поколінь не досягла максимального ліміту, процес продовжується. В іншому випадку, якщо досягнуто максимальну кількість поколінь, повертається останнє знайдене рішення.

Крім того, в алгоритмі можуть бути додаткові критерії зупинки:

- знаходження оптимального рішення;
- вичерпання ліміту ітерацій або часу, виділеного на еволюцію.

Основний цикл еволюції продовжується до досягнення однієї з цих умов. Кожен етап алгоритму пов'язаний із мінімізацією довжини маршрутів, дотримуючись часових вікон, обмежень на місткість та пройдену відстань. Таким чином, алгоритм здатний адаптуватися до різних умов за рахунок

гнучкості параметрів та використання еволюційних принципів.

3.2.4 Реалізація алгоритму імітації відпалу

Клас SASolution реалізує алгоритм імітації відпалу на вирішення завдання маршрутизації CVRPTW. Основний метод Run визначає параметри алгоритму, такі як ліміт нагріву, початкова температура, швидкість охолодження та кількість перезапусків, а також ініціалізує початкове рішення за допомогою методу CreateInitialSolution. Цей метод формує початкові маршрути, послідовно вибираючи найближчі вузли, які відповідають часовим обмеженням та обмеженням на місткість і відстань, або повертається до депо, якщо відповідний вузол не знайдений.

Метод FindClosest використовується для пошуку найближчого доступного вузла до поточного положення транспортного засобу, перевіряючи обмеження часових вікон, місткості та відстані.

У методі Run створюється матриця відстаней між вузлами, обчислюються початкова вартість маршрутів та параметри для ітеративного покращення рішення. Основний цикл алгоритму проводить перезапуски, у яких виконується внутрішній цикл охолодження. На кожній ітерації випадково вибираються два транспортні засоби та вузол з одного маршруту, який може бути переміщений до іншого маршруту, якщо це зменшує вартість маршруту. Вища «температура» збільшує ймовірність прийняття рішення, що призводить до збільшення вартості рішення. Температура знижується після кожної ітерації із застосуванням геометричної прогресії.

Якщо рішення не покращується (залишається незмінним) протягом ряду ітерацій виконується операція повторного нагріву, внаслідок чого температура знову підвищується. Це дозволяє рішенню знову приймати ходи, які збільшують вартість рішення, дозволяючи алгоритму досліджувати більшу частину простору рішень і не застрягати в локальних мінімумах. Вартість перерахунку визначається змінами у відстанях маршрутів та

перевіряється за допомогою методу AllowMove, який враховує різницю вартості та поточну температуру.

Алгоритм працює протягом заданої кількості повторних нагрівань. Після завершення ітерацій алгоритм вибирає найкраще знайдене рішення та повертає об'єкт ProblemSolution, що містить оптимізовані маршрути.

3.2.5 Реалізація табу пошуку

Для реалізації алгоритму табу-пошуку використовується клас TabuSolution. Серед основних функцій слід зазначити функцію, яка створює початкове рішення, проходячи по кожному транспортному засобу та додаючи найближчий доступний вузол з урахуванням обмежень часових вікон, місткості та пройденої відстані транспортним засобом. Вузли додаються до маршруту, доки дотримуються обмеження, а при їх порушенні транспортний засіб повертається до депо. Також, при створенні початкового рішення виконується пошук найближчого доступного вузла, перевіряючи, чи був вузол відвіданий, та оцінюючи відстані для мінімізації витрат.

У функції Run реалізований основний алгоритм табу-пошуку: він використовує початкове рішення і виконує ітеративну оптимізацію маршрутів. Алгоритм локального пошуку застосовується до всіх транспортних засобів одночасно. Алгоритм перебирає кожен вузол на маршруті кожного транспортного засобу та намагається помістити його у кожну можливу позицію на маршрутах усіх транспортних засобів. Якщо хід табуйований, він не розглядається; однак якщо виконання ходу знижує вартість рішення настільки, що вона стає нижчою за найкращу вартість, що знайдена на поточний момент, хід виконується. Це дозволяє рішенню знову приймати ходи, які збільшують вартість рішення, дозволяючи алгоритму досліджувати більшу частину простору рішень і не застрягати в локальних мінімумах. Переміщення вузлів між маршрутами транспортних засобів оцінюються на основі зміни вартості маршруту, перевірок часових вікон та

табу-списків. Реалізується оптимальний хід (хід, що призводить до найбільшого скорочення або найменшого збільшення витрат). Нові сформовані зв'язки додаються до списку табу, що запобігає розриву сформованих зв'язків протягом ряду ітерацій (розмір списку табу постійний, тому кількість зв'язків відповідає кількості разів, коли виконується не табуйований хід). Пошук припиняється, якщо можливих ходів немає. Алгоритм виконується задану кількість ітерацій. Після завершення ітерацій алгоритм зберігає найкраще знайдене рішення як об'єкта ProblemSolution і виводить підсумкову вартість маршрутів.

3.3 Опис набору вхідних та вихідних даних

Після визначення моделі представлення вхідних даних та принципу роботи алгоритмів рішення, необхідно визначитися з користувацьким представленням даних та створити допоміжні класи, за допомогою яких можна привести вихідні дані до моделі, яку використовують алгоритми пошуку. У якості формату вхідних даних обрано формат .vrp опису файлу. Файл містить заголовок та мета-інформацію, інформацію про депо і клієнтів. Загалом файл має таку структуру:

- name – назва файлу або набору даних;
- comment – додатковий коментар (може бути пустим);
- type – тип задачі (CVRPTW);
- nb_depots – кількість депо;
- nb_clients – кількість клієнтів;
- nb_vehicle – кількість доступних транспортних засобів;
- capacity – максимальна місткість кожного транспортного засобу;
- дані про депо, що представлені у форматі [idname x y readytime duetime], де idname – ідентифікатор, x та y – координати розташування, readytime та duetime – часовий інтервал, коли депо готове обслуговувати транспорт;

- дані про клієнтів, представлено у форматі [idname x y readytime duetime demand], де idname – ідентифікатор, x та y – координати клієнта на карті, readytime – час, з якого клієнт готовий до обслуговування, duetime – час, до якого клієнт має бути обслуговуваний, demand – попит на товар клієнта. Формат структурований так, щоб визначити початкові умови для задачі маршрутизації, де потрібно оптимізувати маршрути транспорту, враховуючи часові обмеження та потреби клієнтів.

Лістинг 3.1 – Структура опису вхідних даних у форматі VRP

```
NAME: test3_10t.vrp
COMMENT:
TYPE: cvrptw
NB_DEPORTS: 1
NB_CLIENTS: 10
NB_VEHICLE: 10
CAPACITY: 519

DATA_DEPORTS [idName x y readyTime dueTime]:
0 0 0 0 0

DATA_CLIENTS [idName x y readyTime dueTime demand]:
1 16 57 92 172 10
2 8 1 60 116 25
3 42 79 36 174 46
4 28 48 44 155 2
5 -26 18 54 177 31
```

Для перетворення вхідного файлу у модель графу використовується клас GraphReader, що знаходиться у модулі «Graph», який реалізує основні компоненти для представлення графа та функціональність для роботи з ним.

Клас GraphReader призначений для обробки текстового файлу, що містить дані завдання CVRPTW, та перетворення цієї інформації у створену структуру даних. Цей клас полегшує підготовку вхідних даних для подальшого використання в алгоритмах рішення задачі.

Основний метод класу GraphReader – ReadCvrptwDataFile, зчитує вміст файлу, аналізує його за рядами та заповнює об'єкт типу CvrptwGraph

необхідною інформацією. Файл містить опис задачі у встановленому форматі, який включає метаінформацію та списки даних про депо та клієнтів. При читанні даних метод послідовно отримує інформацію, таку як назва завдання, коментар, тип завдання, кількість депо, клієнтів і транспортних засобів, а також місткість транспортних засобів. Після цього він поділяє дані на два блоки: дані про депо та дані про клієнтів. На підставі прочитаного рядка метод визначає, до якого блоку вона відноситься, і викликає відповідний допоміжний метод для обробки.

Функція addDepot використовується для обробки рядків, які містять дані про депо. Вона створює об'єкт типу Node, що представляє депо, та заповнює його характеристики: ідентифікатор депо, координати на площині, часові вікна, поле isVisited. Потім об'єкт депо додається до графа за допомогою методу AddDepot об'єкта CvrptwGraph. Функція addClient призначена для обробки рядків, що містять дані клієнтів. Вона також створює об'єкт типу Node, що представляє клієнта, та заповнює інформацію про нього, включаючи унікальний ідентифікатор, координати, часові вікна обслуговування, попит на товар та мітку відвідування. Після цього клієнт додається до графа через метод AddClient об'єкта CvrptwGraph.

Таким чином, клас GraphReader виконує роль перетворювача, який читає файл із даними завдання маршрутизації, перевіряє його коректність і перетворює на об'єкт графа. Цей об'єкт представляє інформацію про клієнтів, депо та обмеження, що дозволяє його використання в алгоритмах оптимізації.

У модулі «Src» зберігаються файли з розширенням .vrp із вхідними даними для тестування та розрахунків. Файли з цим розширенням можна створювати вручну, зберігаючи їх у цій папці. Також, у «Src» можна перенести готові файли необхідного формату, підготовлені заздалегідь. Це може бути корисним, якщо вхідні дані вже сформовані або отримані з інших джерел, і їх потрібно інтегрувати у проєкт. Крім того, є можливість автоматично згенерувати файл з даними в цьому форматі за допомогою

програми InputGenerator. Програма дозволяє задати кількість дронів, клієнтів, максимальну місткість транспорту та автоматично створює файл у папці «Src». Усі необхідні параметри генеруються випадковим чином, забезпечуючи підготовку даних для тестування.

Програма працює в такий спосіб:

- користувач вводить необхідні дані через консоль – кількість транспортних засобів, кількість цілей та місткість;

- перевіряється наявність вказаної у коді директорії. Якщо директорія відсутня, програма повідомляє про це та завершує виконання. У разі створення директорії створюється файл, в якому записуються дані у форматі .vrp за заданою структурою. Після успішного виконання програма виводить шлях до створеного файлу.

Клас ProblemSolution із модуля «Algorithms» призначений для зберігання даних про знайдене рішення, виконання перевірки на відповідність обмеженням задачі, запуск алгоритмів оптимізації та експорт результатів. Він використовується як основний елемент програми для перевірки рішень та взаємодії з різними алгоритмами оптимізації. Клас містить такі властивості:

- input – об'єкт типу CvrptwGraph, що представляє вхідний граф завдання, що містить дані про клієнтів, депо, обмеження та транспортні засоби;

- routes – список маршрутів, де кожен маршрут представлений списком вузлів (Node). Вузли відповідають клієнтам чи депо;

- arrivalTimes – список часу прибуття кожного маршруту. У цьому списку для кожного вузла маршруту зберігається час відвідування;

- solutionExists – логічний прапор, що вказує на існування валідного рішення (тобто виконуються всі обмеженням завдання);

- maxRouteLength – максимальна довжина одного маршруту серед усіх;

- sumOfRouteLengths – сумарна довжина всіх маршрутів;

- time – час виконання обраного алгоритму;

- feasible – логічний прапор, який вказує на те, чи є рішення допустимим з урахуванням усіх заданих обмежень;

- ProblemMode – перелік, що задає завдання оптимізації: MINSUM – мінімізація сумарної довжини всіх маршрутів, MINMAXLEN – мінімізація максимальної довжини одного маршруту;

- EAlgorithms – перелік доступних алгоритмів для вирішення задачі: алгоритм колонії мурах та імітації відпалу, генетичний та жадібний алгоритми, пошук табу.

Клас ProblemSolution також надає два конструктори. Перший створює порожній об'єкт рішення з початковими значеннями властивостей. Другий приймає вхідний граф і маршрути, після чого виконує перевірку їх коректності. Ця перевірка включає: дотримання часових вікон, щоб час прибуття до вузла знаходився у допустимих межах, кожен клієнт відвідується рівно один раз та дотримання обмежень на довжину маршрутів, кількість маршрутів та місткість транспортних засобів. Якщо хоча б одну з умов порушено, програма виводить повідомлення про помилку.

Метод PrintIntoFile використовується для запису рішення у файл. У ньому вказуються такі дані, як назва вхідного фалу з даними, кількість транспортних засобів та клієнтів, обмеження на максимальну пройдену відстань транспортним засобом, максимальна довжина маршруту, сумарна довжина маршрутів, маршрути, назва алгоритму та час його виконання та зображення графу з оптимальними маршрутами для транспортних засобів.

Метод Run призначений для запуску вказаного алгоритму оптимізації із заданими параметрами (саме цей метод запускає кожен з алгоритмів). Метод: визначає, який алгоритм з переліку використовувати, викликає відповідну реалізацію алгоритму, вимірює час виконання та виводить результати.

Для відображення знайдених маршрутів, клієнтів та депо у вигляді графу використовується функція DrawSolution, що приймає обьект типу ProblemSolution. У функції визначаються мінімальні та максимальні значення

координат всіх клієнтів і депо для того, щоб встановити межі області, що відображається, і розрахувати масштабні коефіцієнти для перетворення координат об'єктів в екранні координати. Після цього депо та маршрути візуалізуються на панелі.

Функція DrawDeport відповідає за відображення депо. Спочатку координати депо перетворюються з реальних координат на екранні з урахуванням мінімальних значень і масштабу. Потім депо малюється у вигляді кола фіксованого розміру оранжевого кольору. У центрі кола відображається 0, що означає початок та кінець маршруту. Функція DrawClient виконує аналогічні дії для клієнтів. Вона малює клієнта у вигляді кола заданого кольору та відображає ID клієнта всередині кола. Координати перетворюються за тим же принципом, що й у DrawDeport, а текст вирівнюється з центром кола.

Метод GetNewRandomColor відповідає за генерацію випадкового кольору для маршрутів. Функція генерує новий колір і перевіряє, щоб він не збігався з існуючими кольорами, які зберігаються в списку colorList. Новий колір додається до списку та повертається для використання у відтворенні маршрутів.

Функція DrawTruck використовується для відображення маршрутів транспортних засобів. Вона малює лінії між депо та клієнтами, а також між самими клієнтами. Лінії позначають маршрути та включають стрілки, що вказують напрямок руху. Спираючись на векторну арифметику, допоміжна функція GetBorderPoint визначає точки, що відповідають за межі кіл. Це потрібно для того, щоб лінії починалися і закінчуювалися на межі вершин, а не в їхніх центрах.

В результаті на панелі відображається графічне представлення маршрутів: депо позначаються помаранчевими колами з ID у центрі, клієнти – колами заданого кольору з ID, а маршрути – лініями з кольоровими стрілками, що з'єднують депо та клієнтів.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

У цьому розділі аналізуються результати обчислень та швидкодія використовуваних алгоритмів за допомогою екземплярів еталонних даних.

Замість реальних завдань, програмі може бути переданий раніше підготовлений набір даних – тест продуктивності (бенчмарк) або екземпляр еталонних даних, який створено та сформульовано для дослідників, щоб вони могли перевірити свої алгоритми. Використовуються екземпляри еталонних даних не тільки для перевірки працездатності програми, але і для порівняння отриманих результатів (вихідні маршрути, час роботи програми, загальну вартість маршрутів) з результатами роботи інших алгоритмів на цьому ж наборі вхідних даних [49].

Для завдання маршрутизації транспортних засобів ці екземпляри використовуються як вхідні дані та містять наступну базову інформацію:

- графічні координати, що представляють положення депо та точок доставки;
- розмір завдання (кількість точок доставки);
- кількість транспортних засобів;
- обмеження для доставки (міткість транспорту, часові вікна).

Для перевірки результатів алгоритмів використано класичний набір, розроблений M. Solomon у 1987 році і порівняно з найкращими відомими результатами. У цих тестах враховується кілька факторів, які впливають на поведінку алгоритмів маршрутизації та планування, а саме: географічні дані; кількість клієнтів, які обслуговуються транспортним засобом; відсоток обмежених у часі клієнтів; щільність і розташування часових вікон [50].

Тест продуктивності включає шість різних типів завдань (C1, C2, R1, R2, RC1, RC2). Координати клієнта ідентичні для всіх проблем в межах одного типу (тобто R, C і RC). Завдання відрізняються щодо ширини часових вікон. Деякі мають дуже вузькі часові вікна, тоді як інші мають часові вікна,

які не є суттєвими обмеженнями. З точки зору щільності часових вікон, тобто відсотка клієнтів із часовими вікнами, створено завдання з 25, 50, 75 і 100% часовими вікнами. Кожен набір даних містить від восьми до дванадцяти завдань із розмірністю 25, 50 або 100 вузлів [50]. На рисунку 4.1 зображено приклад набору вхідних даних типу C.

Benchmark	Instance	*n*	*K*	*Q*	UB	Opt
▼Solomon (1987)						
	C101	100		200	827.3	yes
	C102	100		200	827.3	yes
	C103	100		200	826.3	yes
	C104	100		200	822.9	yes
	C105	100		200	827.3	yes
	C106	100		200	827.3	yes
	C107	100		200	827.3	yes
	C108	100		200	827.3	yes
	C109	100		200	827.3	yes
	C201	100		700	589.1	yes
	C202	100		700	589.1	yes
	C203	100		700	588.7	yes
	C204	100		700	588.1	yes
	C205	100		700	586.4	yes
	C206	100		700	586	yes
	C207	100		700	585.8	yes
	C208	100		700	585.8	yes

Рисунок 4.1 – Приклад бенчмарку Solomon (1987)

Набір завдань C генерує кластеризовані вузли, часові вікна яких були згенеровані з відомим рішенням. Завдання типу C мають вузькі часові вікна і невелику вантажопідйомність транспорту.

Набори завдань R генерують клієнтів, розташованих рівномірно випадковим чином по площі. Завдання цього типу характеризуються широкими часовими вікнами і більшою вантажопідйомністю автомобіля. Тому розв’язки для задач типу R зазвичай передбачають меншу кількість маршрутів із більшою кількістю клієнтів на кожному маршруті, у порівнянні із задачами типу C.

Завдання типу RC об'єднують у собі як випадково розміщених, так і згрупованих клієнтів [51].

Набори завдань R1, C1 і RC1 мають короткий горизонт планування і мають лише кілька клієнтів на маршрут (приблизно від 5 до 10). Набори R2, C2 і RC2 мають довгий горизонт планування, що дозволяє багатьом клієнтам (понад 30) обслуговуватися одним транспортним засобом [51].

Експерименти здійснюються шляхом використання трьох типів задач для кожного алгоритму, які зупиняються після встановленого часу обчислення. На рисунку 4.2 зображено приклад файлу з задачею R101, де використовуються кількість доступних транспортних засобів, максимальна вантажопідйомність кожного транспортного засобу, номер клієнта, координати, попит (потреба) клієнта в товарі, час обслуговування та часові вікна [50].

Щоб забезпечити більшу точність, кожне дослідження всіх типів та різної розмірності виконувалося п'ять разів, після чого обчислювалося середнє значення отриманих результатів, що вказувалось в наступних таблицях. Це дозволяє підвищити достовірність вимірювань часу роботи алгоритмів та виявити можливу нестабільність результатів для алгоритмів.

R101

VEHICLE

NUMBER	CAPACITY
25	200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	35	35	0	0	230	0
1	41	49	10	161	171	10
2	35	17	7	50	60	10
3	55	45	13	116	126	10
4	55	20	19	149	159	10
5	15	30	26	34	44	10
6	25	30	3	99	109	10
7	20	50	5	81	91	10
8	10	43	9	95	105	10
9	55	60	16	97	107	10
10	30	60	16	124	134	10
11	20	65	12	67	77	10
12	50	35	19	63	73	10
13	30	25	23	159	169	10
14	15	10	20	32	42	10
15	30	5	8	61	71	10
16	10	20	19	75	85	10
17	5	30	2	157	167	10
18	20	40	12	87	97	10
19	15	60	17	76	86	10
20	45	65	9	126	136	10
21	45	20	11	62	72	10
22	45	10	18	97	107	10
23	55	5	29	68	78	10
24	65	35	3	153	163	10
25	65	20	6	172	182	10

Рисунок 4.2 – Файл R101 з еталонних даних Solomon

Проведено порівняння результатів тесту продуктивності Solomon кожного типу (R, C, RC) розмірністю 25, 50 та 100 вершин, з результатами використовуваних наближених алгоритмів рішення CVRPTW. Отримані результати записані в таблиці 4.1. В якості результативного показника використовується загальна довжина проїзду транспортів за всіма оптимізованими маршрутами та відхилення сумарної відстані маршрутів, знайденої алгоритмами, від оптимального значення. У першій колонці представлені екземпляри тесту продуктивності. У другій колонці знаходяться найкращі результати тесту, далі – результати, отримані за допомогою алгоритмів пошуку. Метою даного досліждення є перевірка схожості фінальних значень алгоритмів з вже відомим кращим розв’язком для кожного екземпляру, що має невелику та середню розмірність, щоб визначити на скільки правильно реалізовані алгоритми вирішують завдання CVRPTW.

Таблиця 4.1 – Показники вартості алгоритмів для тестів M. Solomon

Екземпляр	Кращий відомий розв'язок	Жадібне рішення	Рішення GA	Рішення ACO	Рішення TS	Рішення SA
R101-25	617,3	623,5	625,2	634,4	641,06	632,5
R101-50	1044	1065,3	1046,5	1087,2	1054,4	1064,6
R101-100	1637,7	1786,1	1657,37	1735,6	1687,7	1708,2
C101-25	191,3	193,9	196,5	196,5	196,5	196,5
C101-50	362,4	372,3	364.86	371,9	366,54	377,3
C101-100	827,3	911,4	833,4	885,6	842,8	865,7
RC101-25	461,1	463,2	467,35	469,8	471.12	468,81
RC101-50	944	976,8	950,7	987,25	957,3	984,68
RC101-100	1619,8	1723,1	1649,8	1717,68	1657,44	1698,17

Провівши аналіз таблиці 4.1, можна сказати, що отримані результати під час використання евристичних алгоритмів пошуку, близькі до кращих

результатів даних тестів продуктивності невеликої та середньої розмірності. В середньому відхилення від кращого рішення не перевищує 5,5%. Але, зі зростанням розмірності видно, що відхилення трохи збільшується (для метаевристик не більше 6%). Час виконання алгоритмів для цих вхідних даних наведено в таблиці 4.2.

Таблиця 4.2 – Показники часу виконання алгоритмів для тестів M. Solomon

Екземпляр	Жадібне рішення, с	GA рішення, с	ACO рішення, с	TS рішення, с	SA рішення, с
R101-25	0,0072	0,72	0,059	0,086	0,057
R101-50	0,0449	1,56	0,58	0,9	0,67
R101-100	0,131	3,2	1,2	1,05	1,28
C101-25	0,0066	0,083	0,0254	0,053	0,0278
C101-50	0,0365	1,44	0,43	0.72	0.51
C101-100	0,0987	2,7	0,94	0,8	1,12
RC101-25	0,0068	0,098	0,054	0,074	0,054
RC101-50	0,0423	1,62	0,51	0,86	0,64
RC101-100	0,123	2,96	1,1	1,92	1,26

Виходячи із наведених результатів, реалізовані алгоритми досить швидко вирішують завдання. Найшвидше вирішення отримав жадібний алгоритм. Враховуючи результати таблиць 4.1 та 4.2 можна зробити висновки, що для завдань невеликої розмірності найближче до оптимального і найшвидше вирішує жадібний алгоритм. Далі за точністю даних близькі до оптимальних рішення отримує генетичний алгоритм. Алгоритм імітації відпалу краще знаходить рішення ніж алгоритм мурашиної колонії, а табу пошук менш точний через обмежений простір пошуку. За часом виконання через обробку популяцій генетичний алгоритм найповільніший. Мурашиний алгоритм швидше виконує завдання майже за всі метаевристики. Табу пошук швидше працює ніж генетичний, але повільніший за інші алгоритми.

Для завдань середньої розмірності завдяки глобальному пошуку найкраще рішення шукає генетичний алгоритм, але також найдовше. Жадібний алгоритм все ще найшвидший, але починає втрачати точність. Мурашиний алгоритм швидше за всі підходи, крім жадібного, але менш точний порівняно з пошуком табу та імітацією відпалу. Алгоритм симуляції відпалу працює швидше ніж табу пошук, але довше ніж мурашиний.

Оскільки у еталонних даних M. Solomon (1987) максимальна розмірність клієнтів становить 100, для перевірки часу та наближеності рішень до вже відомих оптимальних рішень на даних великої розмірності використовувались тести продуктивності Homberger and Gehring (1999).

Для проведення експерименту було розглянуто три типи екземплярів Homberger and Gehring за різною розмірністю 200, 400, 600, 800 та 1000 клієнтів. Формат завдання наведено на рисунку 4.3.

C1_2_1

VEHICLE
NUMBER CAPACITY
50 200

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	70	70	0	0	1351	0
1	33	78	20	750	809	90
2	59	52	20	553	602	90
3	10	137	30	147	219	90
4	4	28	10	616	661	90
5	25	26	20	128	179	90
6	86	37	10	478	531	90
7	1	109	10	616	680	90
8	6	135	40	351	386	90
9	32	79	20	655	721	90
10	24	26	20	219	271	90
11	86	36	20	384	443	90
12	95	35	10	196	252	90
13	63	50	10	364	416	90
14	100	106	10	567	626	90
15	99	112	20	846	911	90
16	36	135	10	598	669	90
17	57	59	10	824	890	90

Рисунок 4.3 – Файл C1_2_1 з еталонних даних Homberger and Gehring

Типи даного тесту продуктивності діляться як і в тестах M. Solomon: згруповані клієнти C, рівномірно розподілені клієнти R та суміш типів R та C [50, 52]. Результати досліджень записані в таблицю 4.3. Формат таблиці залишається таким же як у попередніх тестах.

Вхідні дані Homberger and Gehring містять інформацію про транспортні засоби (VEHICLE) та споживачів (CUSTOMER), а саме: NUMBER – верхня межа кількості транспортних засобів, CAPACITY – вантажопідйомність кожного транспортного засобу, CUST NO – номер вузла (де 0 – депо, а решта – клієнти), XCOORD та YCOORD – координати вузла X і Y відповідно, DEMAND – попит клієнта, READY TIME та DUE TIME – часове вікно клієнта, SERVICE TIME – час обслуговування клієнта [50, 52].

Таблиця 4.3 – Показники для тестів Homberger and Gehring (1999)

Екземпляр	Кращий відомий розв'язок	Жадібне рішення	Рішення GA	Рішення ACO	Рішення TS	Рішення SA
C1_2_1	2704,57	3084,8	2784,2	2861,7	2857,8	2883,5
R1_2_1	4784,11	5231,2	4812,4	4894,2	4835,4	4851,6
RC1_2_1	3602,80	4001	3687,3	3746,2	3721,1	3728,8
C1_4_1	7152,02	7678,1	7204,3	7326,5	7265,6	7288,2
R1_4_1	10372,31	10876,4	10465,2	10502,2	10487,3	10528,3
RC1_4_1	8571,32	9012,3	8701,3	8764,2	8732,6	8761,3
C1_6_1	14095,64	15123,3	14356,2	14531,3	14462,3	14498,3
R1_6_1	21394,95	22678,3	21645,5	21789,6	21712,4	21765,2
RC1_6_1	16982,86	18121,2	17214,3	17513,2	17457,3	17487,1
C1_8_1	25030,36	27568,1	25514,2	26432,1	25764,3	25983,1
R1_8_1	36767,92	38956,3	37202,3	38365,3	37568,6	38123,4
RC1_8_1	30464,65	33791,2	30986,6	31823,6	31245,8	31768,9
C1_10_1	42478,95	45896,7	42968,6	44302,1	43264,3	43897,7
R1_10_1	53380,18	57987,6	53921,7	54887,8	54179,2	54589,3
RC1_10_1	45830,62	50647,2	46234,8	47668,6	46873,9	47102,3

За результатами таблиці 4.3, можна зробити висновок, що алгоритми здатні знаходити близькі до оптимальних рішення, але зі зростанням

розмірності вхідних даних відхилення збільшується, особливо у жадібного алгоритму (таблиця 4.4).

Таблиця 4.4 – Показники часу виконання алгоритмів для тестів Homberger and Gehring (1999)

Екземпляр	Жадібне рішення, с	GA рішення, с	ACO рішення, с	TS рішення, с	SA рішення, с
C1_2_1	0,1	6,8	2	2,53	2,13
R1_2_1	0,135	7,23	2,3	2,96	2,6
RC1_2_1	0,11	6,97	2,17	2,64	2,24
C1_4_1	0,15	10,78	3,93	4,97	4,29
R1_4_1	0,18	12,25	4,21	5,64	4,51
RC1_4_1	0,13	11,67	4,08	5,27	4,18
C1_6_1	0,14	12,62	5,58	8,61	7,1
R1_6_1	0,17	13,43	6,49	9,45	7,96
RC1_6_1	0,13	12,22	5,75	8,86	6,84
C1_8_1	0,18	27,46	7,5	14,83	9,4
R1_8_1	0,24	29,38	8,8	15,6	10,64
RC1_8_1	0,21	28,43	8	15,03	10,12
C1_10_1	0,22	90,84	13,2	28,76	18,74
R1_10_1	0,24	112,6	14,65	32,4	21,4
RC1_10_1	0,27	105,2	13,85	30,84	20,78

Провівши аналіз часу виконання алгоритмів з таблиці 4.4 можна зазначити, що зі збільшенням розмірності задачі збільшується час виконання метаевристичних алгоритмів. Для жадібного алгоритму час майже не змінюється зі зростанням даних через простоту алгоритму.

Аналізуючи результати таблиць 4.3 та 4.4 можна зробити висновок, що для задач великої розмірності жадібний алгоритм залишається найшвидшим, але точність його рішень значно гірше найкращого значення і, чим більше

розмір задачі, тим більше він має відхилення від оптимального рішення. Найбільш точним алгоритмом є генетичний алгоритм, але час його виконання різко зростає зі зростанням обсягу даних через збільшення обсягу популяції та числа ітерацій.

Час виконання мурашиного алгоритму серед інших метаевристичних алгоритмів пошуку оптимальних маршрутів найкращий через використання паралельності при вирішення задачі. За точністю отриманих результатів трохи гірше знаходить рішення ніж алгоритм імітації відпалу (іноді може отримати рішення точніше за SA, але зі збільшенням розміру завдання це твердження втрачає сенс) та табу пошуку.

Алгоритм симуляції відпалу має схожі з мурашиним алгоритмом фінальні значення, але більш близькі до кращого рішення та результату табу пошуку. Проте алгоритм імітації відпалу працює довше ACO через поступове зменшення температури та пошук локального мінімуму. Пошук табу отримує досить точні результати, одразу після генетичного алгоритму, але займає більше часу ніж алгоритм мурашиної колонії та імітації відпалу.

Оскільки проведені тести продемонстрували, що алгоритми знаходять рішення, близькі до оптимальних, тепер можна перевірити їх роботу на власних файлах, сформованих випадковим чином, а не на заздалегідь підготовлених даних, створених спеціально для тестування.

Для порівняння алгоритмів використовувались вхідні файли різної розмірності: 30, 70, 100, 300, 500, 700 та 1000 клієнтів (файли з розмірністю 30, 70, 300, 500 та 700 використовуються для дослідження результатів, які не були включені у попередні тести). Результати роботи алгоритмів, тобто загальна довжина всіх маршрутів на час пошуку рішення, наведено у таблицях 4.5 та 4.6. У першій колонці описується розмірність вхідних даних. У інших колонках знаходяться результати, отримані за допомогою алгоритмів пошуку (у першій таблиці знайдене рішення, у другій – час виконання алгоритмів). Час виконання алгоритмів для цих вхідних даних занесено до таблиці 4.6.

Таблиця 4.5 – Показники вартості алгоритмів на власних даних

Розмірність	Жадібне рішення	GA	ACO	TS	SA
30	1062,94	676,52	703,24	733,41	688,32
70	2315,11	1260,52	1354,42	1265,59	1344,28
100	2361,32	1376,78	1501,37	1420,25	1465,16
300	5868,37	4542,48	4768,26	4642,87	4702,45
500	16634,26	14736,45	14957,21	14786,96	14824,3
700	27684,39	24678,36	25766,98	25182,45	25436,35
1000	49348,32	43679,63	44963,87	44246,38	44629,54

Через випадковий характер створення вхідних даних (місткості транспорту, координат, часових вікон та клієнтів) спостерігається розбіжність між результатами на еталонних даних і на власних тестах.

Наприклад, для розмірності 100 клієнтів найкраще значення (таблиця 4.6) за загальними результатами алгоритмів дорівнює 833.4, а час 0.8 секунд, тоді як результати за власними даними для такого ж розміру більше (1376.78 та 1,34 секунди).

Таблиця 4.6 – Показники часу виконання алгоритмів на власних даних

Розмірність	Жадібне рішення	GA	ACO	TS	SA
30	0,0054	2,75	0,29	0,49	0,379
70	0,0067	4,67	0,87	1,2	0,94
100	0,007	8,28	1,34	1,67	1,59
300	0,0073	17,56	4,26	5,46	4,58
500	0,0078	28,21	7,63	10,38	8,49
700	0,0084	37,62	9,77	18,42	12,75
1000	0,012	116,28	15,48	34,92	23,71

Загалом висновки щодо роботи алгоритмів пошуку збігаються з результатами, отриманими за допомогою тестів продуктивності. Зі збільшенням розміру задачі точність метаевристичних алгоритмів трохи погіршується та для кожного алгоритму збільшується час виконання. Жадібний алгоритм має велику розбіжність за точністю порівняно з іншими алгоритмами, але залишається найшвидшим. Також, генетичний алгоритм залишається більш точним і найдовшим при вирішення за інші алгоритми. Табу пошук має досить точні результати, але працює швидше тільки за генетичний алгоритм. Серед метавристичних алгоритмів мурашиний алгоритм працює швидше і знаходить наближені результати до рішень алгоритму імітації відпалу. За швидкістю роботи симуляція відпалу витрачає більше часу ніж мурашиний алгорим, але знаходить більш точні значення, які гірші, але наближені до табу пошуку.

За результатами усіх проведених досліджень можна зробити висновок, що для завдань невеликого розміру алгоритм мурашиної колонії та імітації відпалу є найбільш збалансовані за часом і точністю, а жадібний можна використовувати для отримання швидкої оцінки результатів. Для завдань середньої розмірності алгоритм імітації відпалу та мурашиної колонії мають схожі значення за часом та точністю, але симуляція відпалу може бути простішою при налаштуванні вхідних параметрів. Для завдань великої розмірності в якості альтернативи для скорочення часу виконання можна використовувати мурашиний алгоритм, що має найкращий баланс між часом та точністю, трохи гірший баланс через довше виконання має алгоритм імітації відпалу (загалом для задач середнього та великого розміру мурашиний алгоритм працює швидше ніж імітація відпалу, але менш точно). Для точних рішень доцільно використовувати генетичний алгоритм або табу-пошук, але вони потребують більших часових витрат.

ВИСНОВКИ

Робота присвячена аналізу та порівнянню ефективності використання різних алгоритмів для вирішення задачі CVRPTW, яка полягає у знаходженні оптимальних маршрутів для транспорту з урахуванням обмежень щодо місткості та часових вікон. Це складна задача комбінаторної оптимізації, яка має значні практичні застосування, зокрема в логістиці та транспортних системах.

Розглянуто три групи алгоритмів, що здатні вирішувати завдання CVRPTW. В результаті аналізу класичних алгоритмів для рішення задачі зроблено такий висновок: найкращими алгоритмами для розв'язання задач VRP є наближені методи, які дають змогу долати локальний оптимум у процесі пошуку рішень та дозволяють знаходити розв'язання NP-важких завдань.

Проведено аналіз кожного з методів рішення та зроблено висновки щодо їх ефективності в умовах поставленої задачі. Були розглянуті такі евристичні та метаевристичні алгоритми: жадібний алгоритм, генетичний алгоритм, алгоритм колонії мурах, пошук із заборонами та імітація відпалу. Запропоновані алгоритми можуть використовуватися для вирішення широкого спектру задач у логістиці, підвищуючи ефективність планування маршрутів та використання ресурсів.

У задачах CVRPTW метаевристичні методи пошуку оптимального шляху у графах значно перевершують евристичні підходи в ситуаціях, коли побудова маршруту має відбуватися в реальному часі (наприклад, у логістичних задачах, де необхідно швидко адаптуватися до зміни умов). Однак важливим недоліком метавристичних алгоритмів є їхня тривала робота на задач великої розмірності, де через випадковий характер пошуку оптимального рішення алгоритми можуть витрачати багато часу через велику кількість можливих варіантів. Щодо використання метаевристичних підходів

слід зазначити:

- генетичний алгоритм забезпечує найкращу точність серед усіх методів, особливо на задачах великого розміру, але потребує найбільше часу на пошук;

- мурашиний алгоритм демонструє баланс між точністю та часом виконання, особливо на задачах середнього і великого розміру. Проте його налаштування є досить складним через необхідність вибору вагових коефіцієнтів, які суттєво впливають на результат;

- алгоритм імітації відпалу забезпечує прийнятні результати з точки зору точності та часу. Його реалізація є найпростішою серед усіх розглянутих методів;

- алгоритм табу-пошуку характеризується помірною складністю реалізації та точним виконанням. Проте за швидкістю він поступається іншим методам, крім генетичного, особливо на задачах середнього та великого розміру;

- жадібний алгоритм є найшвидшим серед усіх методів, але значно поступається за точністю. Його доцільно використовувати лише для отримання попередніх оцінок або в задачах невеликого розміру.

Для проведення експериментальних досліджень створено застосунок, який дозволяє тестувати та порівнювати різні алгоритми на прикладі конкретних наборів даних та будувати маршрути з урахуванням обмежень. У якості результатів роботи програми отримується інформація про вхідні дані (кількість ТЗ та клієнтів, обмеження за місткістю), обраний алгоритм, результати його роботи (загальна довжина всіх маршрутів, найбільша довжина маршруту та час виконання) та знайдене рішення – маршрути між клієнтами та депо у вигляді графу. Застосунок може бути розширений для додавання нових алгоритмів або модифікації існуючих, що сприяє гнучкості аналізу. Для реалізації програмного застосунка використана мова програмування C#, з використанням Windows Forms в середовищі Microsoft Visual Studio 2022.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Сорока, А. Г. Smart routes: система для розробки та порівняння алгоритмів рішення завдання оптимізації маршрутів з реалістичними обмеженнями [Текст] / А. Г. Сорока, Г. В. Міхельсон, А. В. Міщеряков // Автоматика і телемеханіка. – 2024. – Т. 3. – С. 101–118.

2. Дзундза, В. С. Програмна система розв’язання задач маршрутизації транспортних засобів [Текст] / В. С. Дзундза, Г. Й. Михальчук // Актуальні проблеми автоматизації та інформаційних технологій. – 2017. – Т. 21. – С. 52–59.

3. Маций, О. Б. Клас задач маршрутизації, які зводяться до задачі комівояжера [Текст] / О. Б. Маций // Автомобіль і електроніка. Сучасні технології. – 2017. – Т.12. – С. 167–170.

4. Скукис, А. Е. Оптимізаційні завдання у транспортній логістиці [Текст] / А. Е. Скукис // Теорія оптимальних рішень. – 2015. – С. 106–113.

5. Uchoa, E. New benchmark instances for capacitated vehicle routing problem [Текст] / E. Uchoa, D. Pecin, A. Pessoa, M. Poggi // European Journal of Operational Research. – 2017. – Т. 257. – С. 845-858.

6. Goel, R. Vehicle routing problem and its solution methodologies: a survey [Текст] / R. Goel, R. Maini // International Journal of Logistics Systems and Management. – 2017. – Т. 28. – С. 419–435.

7. Гуляницький, Л. Ф. До класифікації задач маршрутизації транспортних засобів [Текст] / Л. Ф. Гуляницький, А. А. Коткова // Науковий вісник Ужгородського університету Серія Математика і інформатика. – 2020. – Т. 36. – С. 73–84.

8. Beresneva, E. N. Analysis of Mathematical Formulations of Capacitated Vehicle Routing Problem and Methods for their Solution [Текст] / E. N. Beresneva, S. Avdoshin // Proceedings of the Institute for System Programming of RAS. – 2018. – Т. 30. – С. 233–250.

9. Toth, P. Vehicle Routing Problems, Methods, and Applications Second Edition [Текст] / P. Toth, D. Vigo // The Society for Industrial and Applied Mathematics and the Mathematical Optimization Society. – 2014. – Т. 12. – С. 351–381.

10. Kumar, S.N. A Time-Dependent Vehicle Routing Problem with Time Windows for E-Commerce Supplier Site Pickups Using Genetic Algorithm [Текст] / S. N. Kumar, R. Panneerselvam // Intelligent Information Management. – 2015. – Т. 7. – С. 181–194.

11. Кубіл, В. Н. A review of dynamic vehicle routing problems [Текст] / В. Н. Кубіл, Ю. О. Чернишев // Міжнародний журнал «Програмні продукти та системи». – 2020. – Т. 21. – С. 491–501.

12. Wang, Z. Vehicle routing problem with drones [Текст] / Z. Wang, J. B. Sheu // Transportation Research. – 2019. – Т. 122. – С. 350–364.

13. Haidari, L. A. The economic and operational value of using drones to transport vaccines [Текст] / L. A. Haisari, S. T. Brown, M. Ferguson, E. Bancroft // Vaccine. – 2016. – Т. 34. – С. 4062–4067.

14. Buhrkal, K. The Waste Collection Vehicle Routing Problem with Time Windows in a City Logistics Context [Текст] / K. Buhrkal, A. Larsen, S. Ropke // Procedia – Social and Behavioral Sciences. – 2012. – Т. 39. – С. 241–254.

15. Battarra, M. Exact Algorithms for the Clustered Vehicle Routing Problem [Текст] / M. Battarra, G. Erdogan, D. Vigo // Operations Research. – 2014. – Т. 62. – С. 58–71.

16. Oxenstierna, J. Warehouse Vehicle Routing using Deep Reinforcement Learning [Текст] / J. Oxenstierna // Department of Information Technology. – 2019. – С. 1–73.

17. Dantzig, G. B. The Truck Dispatching Problem [Текст] / G. B. Dantzig, J. H. Ramser // Management Science. – 2010. – Т. 6. – С. 80–91.

18. Eksioglu, B. The vehicle routing problem: A taxonomic review [Текст] / B. Eksioglu, A. Vural, A. Reisman // Computers & Industrial Engineering. – 2009. – Т. 57. – С. 1472–1483.

19. Caceres-cruz, J. Rich Vehicle Routing Problem: Survey [Текст] / J. Caceres-Cruz, P. Arias, D, Guimarans, D. Riera // ACM Computing Surveys. – 2014. – Т. 47. – С. 1–28.

20. Kumari, M. Utilizing a hybrid metaheuristic algorithm to solve capacitated vehicle routing problem [Текст] / M. Kumari, K. Chaudhuri, P. Narang // Results in Control and Optimization. – 2023. – Т. 13. – С. 2–15.

21. Mamoun, K. A. Vehicle Routing Optimization Algorithms for Pharmaceutical Supply Chain: A Systematic Comparison [Текст] / K. A. Mamoun, L. Hammadi, A, E. Ballouti, A. Novaes // Transport and Telecommunication. – 2024. – Т. 25. – С. 161–173.

22. Siti, N. I. A review on delivery routing problem and its approaches [Текст] / N. I. Siti, K. Ku Ruhana, A. Syariza// Journal of Theoretical and Applied Information Technology. – 2017. – Т. 95. – С. 367–380.

23. Savelsbergh, M. W. Local search in routing problems with time windows [Текст] / M. W. Savelsbergh // Annals of Operations Research. – 1985. – Т. 4. – С. 285–305.

24. Tan, S. Y. The Vehicle Routing Problem: State-of-the-Art Classification and Review [Текст] / S. Y. Tan, W. C. Yeh // Smart Manufacturing Networks for Industry 4. – 2021. – Т. 11. – С. 1–28.

25. Greedy vs. Heuristic Algorithm [Електронний ресурс] – Режим доступу: www/ URL: https://www.baeldung.com/cs/greedy-vs-heuristic-algorithm/ – 06.11.2024 р. – Загол. з екрану.

26. Sowole, O. S. A Comparative Analysis of Search Algorithms for Solving the Vehicle Routing Problem [Текст] / O. S. Sowole // Optimization Algorithms – Classics and Recent Advances. – 2023. – Т. 11. – С. 200–202.

27. Скаков, Е. С. Метод пошуку із заборонами для рішення оптимізаційних завдань [Текст] / Е. С. Скаков // Нове слово у науці та практиці. – 2023. – С. 166–171.

28. Desrosiers, J. Time Constrained Routing and Scheduling [Текст] / J. Desrosiers, Y. Dumas, M. M. Solomon, F. Soumis // Handbooks in Operations

Research and Management Science. – 1995. – Т. 2. – С. 35–139.

29. Wang, Y. Review on greedy algorithm [Текст] / Y. Wang // Theoretical and Natural Science. – 2023. – Т. 14. – С. 233–239.

30. Dijkstra, L. Greedy and randomized heuristics for optimization of k-domination models in digraphs and road networks [Текст] / L. Dijkstra, A. Gagarin, P. Corcoran, R. Lewis // the International Network Optimization Conference INOC. – 2024. – С. 1–27.

31. Khayya, E. A survey of the vehicle routing problem and its variants: formulations and solutions [Текст] / E. Khayya, I. Medarhri, R. Zine // Mathematical Modeling and Computing. – 2024. – Т. 11. – С. 333–343.

32. Liu, F. Heuristics for Vehicle Routing Problem: A Survey and Recent Advances [Текст] / F. Liu, C. Lu, L. Gui, Q. Zhang, X. Tong, M. Yuan // Artificial Intelligence. – 2023. – С. 1–67.

33. Avdoshin, S. Local search metaheuristics for Capacitated Vehicle Routing Problem: a comparative study [Текст] / S. Avdosin, E. N. Beresneva // Institute for System Programming of RAS. – 2019. – Т. 31. – С. 121–138.

34. Kumar, V. Relative Performance of Certain Meta Heuristics on Vehicle Routing Problem with Time Windows [Текст] / V. Kumar // International Journal of Information Technology and Computer Science. – 2015. – Т. 7. – С. 40–49.

35. Zhou, W. Multiobjective Vehicle Routing Problem with Route Balance Based on Genetic Algorithm [Текст] / W. Zhou, T. Song, F. He, X. Liu // Discrete Dynamics in Nature and Society. – 2013. – Т. 2. – С. 1–9.

36. Bansal, S. Genetic Algorithm with Elitism for Vehicle Routing Problem [Текст] / S. Bansal, R. Goel, V. Rehami // International Conference On Recent Trends In Computing. – 2018. – Т. 7. – С. 559–565.

37. Reeves, C. R. Heuristic Search Methods: A Review [Текст] / C. R. Reeves // Operational Research Society. – 1996. – С. 122–149.

38. Kirkpatrick, S. Optimization by Simulated Annealing [Текст] / S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi // Science. – 1983. – Т. 220. – С. 671–680.

39. Potvin, J. Y. A Review of Bio-Inspired Algorithms for Vehicle Routing [Текст] / J. Y. Potvin // Studies in Computational Intelligence. – 2008. – С. 1–37.

40. Rangel, T. Ant Colony Optimization Algorithm [Текст] / T. Rangel. – 2019. – С. 1–4.

41. Asih, A. M. Comparison study of metaheuristics: Empirical application of delivery problems [Текст] / A. M. Asih, B. M. Sopha, G. Kriptaniadewa // Journal of Engineering Business Management. – 2017. – Т. 9. – С. 1–12.

42. Sulyukova, L.F. Algorithms for solving problems routing of transport and their application in information systems of cargo transportation management [Текст] / L. F. Sulyukova, Z. I. Akhmedzhanova // International journal of theoretical and applied issues of digital technologies. – 2022. – Т. 2. – С. 40–52.

43. Huang, M. Large scale vehicle routing problem: An overview of algorithms and an intelligent procedure [Текст] / M. Huang, X. Hu // International journal of innovative computing, information. – 2012. – Т. 8. – С. 5809–5819.

44. Ding, J. Review of Research on Vehicle Routing Problem and Related Algorithms [Текст] / J. Ding // Frontiers in Computing and Intelligent Systems. – 2023. – Т. 2. – С. 106–108.

45. Opening a VRP File [Електронний ресурс] – Режим доступу: www/ URL: https://filext.com/file-extension/VRP/ – 28.12.2024 р. – Загол. з екрану.

46. Формат PDF: що це та як його використовувати [Електронний ресурс] – Режим доступу: www/ URL: https://sky.pro/wiki/digital-art/format-pdf-chto-eto-i-kak-ego-ispolzovat/ – 28.12.2024 р. – Загол. з екрану.

47. Portable Document Format (PDF) [Електронний ресурс] – Режим доступу: www/ URL: https://www.techtarget.com/whatis/definition/Portable-Document-Format-PDF/ – 28.12.2024 р. – Загол. з екрану.

48. How to build and deploy a three-layer architecture application with C# [Електронний ресурс] – Режим доступу: www/ URL: https://enlabsoftware.com/development/how-to-build-and-deploy-a-three-layer-architecture-application-with-c-sharp-net-in-practice.html/ – 28.12.2024 р. – Загол. з екрану.

49. Gunawan, A. Vehicle routing: Review of benchmark datasets [Текст] / A. Gunawan, G. Kendall, B. McCollum, H. V. Seow, L. S. Lee // Journal of the operational research society. – 2021. – Т. 72. – С. 1794–1807.

50. VRPTW benchmark problems [Електронний ресурс] – Режим доступу: www/ URL: http://web.cba.neu.edu/~msolomon/problems.htm/ – 28.12.2024 р. – Загол. з екрану.

51. Kallehauge, B. Vehicle Routing Problem with Time Windows [Текст] / B. Kallehauge, J. Larsen, O. B. G. Madsen, M. M. Solomon // Column Generation. – 2006. – Т. 3. – С. 67–98.

52. Capacited Vehicle Routing Problem Library [Електронний ресурс] – Режим доступу: www/ URL: http://vrp.galgos.inf.puc-rio.br/index.php/en/ – 28.12.2024 р. – Загол. з екрану.

53. Бондаренко К. В. Порівняльний аналіз алгоритмів визначення маршруту переміщення в дорожній мережі [Текст] / К. В. Бондаренко, Г. С. Іващенко // III Всеукраїнська студентська наукова конференція «Науковий простір: аналіз, сучасний стан, тренди та перспективи». – 16 червня 2023. – С. 159–160.

54. Бондаренко, К.В. Порівняльний аналіз алгоритмів вирішення задачі CVRPTW [Текст] / К.В. Бондаренко, Г.С Іващенко // Problems of Informatization: the twelfth international scientific and technical conference. – 2024. – Т. 2. – С. 80.