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)

Гібридний метод рішення задачі маршрутизації транспорту з урахуванням додаткових обмежень
(тема)

Виконав:
студент II курсу, групи СПм-21-2
Скляров А.С.
(прізвище, ініціали)

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

Тип програми освітньо-наукова
(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 ______ (підпис) Коваленко А.А. (прізвище, ініціали)

2023 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-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Склярову Артему Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Гібридний метод рішення задачі маршрутизації транспорту з урахуванням додаткових обмежень

затверджена наказом по університету від " 03 " квітня 2023 р. № 318 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 травня 2023 р.

3. Вхідні дані до роботи

1) документація мови програмування C# та платформи .NET;
2) алгоритми вирішення комбінаторних задач на графах;
3) середовище розробки MS Visual Studio 2020 Community;
4) середовище розробки JetBrains Rider 2022.

4. Перелік питань, що потрібно опрацювати у роботі

1) аналіз предметної області;
2) алгоритми рішення задачі маршрутизації транспорту;
3) вибір технології розробки та інструментальних засобів;
4) програмна реалізація тестового програмного застосунку;
5) аналіз результатів дослідження;
6) висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайд-презентація – 21 слайд

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	Аналіз проблеми та огляд існуючих рішень	04.04.23-07.04.23	
2	Вибір та обґрунтування методики дослідження	08.04.23-13.04.23	
3	Вибір інструментальних засобів	14.04.23-18.04.23	
4	Розробка програмного забезпечення	19.04.23-25.04.23	
5	Проведення експериментів	26.04.23-03.05.23	
6	Оформлення матеріалів кваліфікаційної роботи	04.05.23-08.05.23	
7	Подання кваліфікаційної роботи керівникові та	10.05.23-11.05.23	
	її попередній захист		
8	Подання кваліфікаційної роботи на	12.05.23-16.05.23	
	рецензування		

Дата видачі завдання 03 квітня 2023 р.

Студент ____________________
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 104 с., 18 рис., 7 табл., 3 дод., 29 джерел.

ГРАФ, МАРШРУТ, ЗАДАЧА МАРШРУТИЗАЦІЇ ТРАНСПОРТУ, ЧАСОВІ ВІКНА, ВАНТАЖОПІДЙОМНІСТЬ, ЖАДІБНИЙ АЛГОРИТМ, МЕТОД ГІЛОК З ВІДСІКАННЯМ, АЛГОРИТМ КЛАРКА-РАЙТА, ГЕНЕТИЧНИЙ АЛГОРИТМ, КРОСИНГОВЕР, МУТАЦІЯ, ГІБРИДИЗАЦІЯ.

Метою кваліфікаційної роботи є дослідження та розробка гібридних методів вирішення задачі маршрутизації транспорту.

У ході виконання кваліфікаційної роботи був проведений аналіз існуючих рішень, їх переваг та недоліків. Було розроблено та досліджено методи для вирішення задачі маршрутизації транспорту на основі генетичного та мурашиного алгоритмів, жадібного алгоритму та його модифікації, методу гілок з відсіканням та алгоритму Кларка-Райта. Особливістю дослідження є можливість врахування алгоритмами додаткових обмежень, таких як вантажопідйомність транспортних засобів та часові вікна клієнтів. Створено тестове програмне забезпечення з графічним інтерфейсом користувача, що забезпечує можливість дослідження впливу параметрів алгоритмів на ефективність роботи.

ABSTRACT

Master's thesis: 104 pages, 18 figures, 7 tables, 3 appendices, 29 sources.

GRAPH, ROUTE, VEHICLE ROUTING PROBLEM, TIME WINDOWS, CARRYING CAPACITY, GREEDY ALGORITHM, BRANCH-AND-CUT ALGORITHM, CLARKE-WRIGHT ALGORITHM, GENETIC ALGORITHM, CROSSOVER, MUTATION, HYBRIDIZATION.

The major goal of this thesis is to research and development of hybrid methods for solving vehicle routing problem.

During the work, an analysis of existing solutions, their advantages and disadvantages was described. Methods for solving the vehicle routing problem based on genetic and ant algorithms, the greedy algorithm and its modification, the branch and cut method and the Clarke-Wright algorithm were developed and researched. A feature of the research is the ability of algorithms taking into account additional restrictions, such as the carrying capacity of vehicles and customer time windows. Test software with a graphical user interface has been created, which provides the opportunity to study the influence of algorithm parameters for the efficiency of their work.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 10
1.1 Опис ЗМТ 10
1.1.1 Класифікація ЗМТ 10
1.1.2 Математична модель ЗМТ 14
1.1.3 Актуальність проблеми вирішення ЗМТ 16
1.2 Методи вирішення ЗМТ 17
1.2.1 Точні алгоритми 17
1.2.2 Евристичні алгоритми 19
1.2.3 Метаевристичні алгоритми 20
1.3 Аналіз існуючих рішень 22
1.4 Постановка задачі 24
2 ВИКОРИСТОВУВАНІ МЕТОДИ ВИРІШЕННЯ 25
2.1 Жадібний алгоритм 25
2.2 Модифікація жадібного алгоритму 26
2.3 Алгоритм Кларка-Райта 29
2.4 Метод гілок з відсіканням 30
2.5 Гібридний генетичний алгоритм 32
2.5.1 Створення початкової популяції 34
2.5.2 Стратегії вибору батьківських особин 35
2.5.3 Оператори кросинговеру 36
2.5.4 Оператори мутації 38
2.6 Модифікований мурашиний алгоритм 39
2.6.1 Загальний опис 39
2.6.2 Мурахи 41

3 ПРОГРАМНА РЕАЛІЗАЦІЯ 42
3.1 Загальна структура проекту 42
3.2 Опис базових класів 44
3.3 Реалізація алгоритмів 47
3.3.1 Жадібний алгоритм 47
3.3.2 Модифікація жадібного алгоритму 48
3.3.3 Метод Кларка-Райта 49
3.3.4 Метод гілок з відсіканням 51
3.3.5 Генетичний алгоритм 52
3.3.6 Мурашиний алгоритм 56
3.4 Тестування алгоритмів 58
3.5 Збір даних про роботу алгоритмів 60
4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ 61
4.1 Параметри ЗМТ та критерії ефективності роботи алгоритмів 61
4.2 Дослідження роботи генетичного алгоритму 62
4.2.1 Дослідження впливу налаштувань жадібного алгоритму 62
4.2.2 Дослідження впливу кількості ітерацій 65
4.2.3 Дослідження алгоритмів генерації початкової популяції 69
4.3 Дослідження роботи мурашиного алгоритму 71
4.3.1 Порівняння різних типів мурашиного алгоритму 71
4.3.2 Дослідження впливу кількості елітарних мурах 73
ВИСНОВКИ 75
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 76
ДОДАТОК А Графічний матеріал кваліфікаційної роботи 79
ДОДАТОК Б UML-діаграми класів розроблених алгоритмів 91
ДОДАТОК В Вихідний код програмного забезпечення 94

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГА – генетичний алгоритм

ЗК – задача комівояжера

ЗМТ – задача маршрутизації транспорту

CX – циклічний кросинговер (англ., Cycle Crossover)

LINQ – мова запитів до джерела даних (англ., Language Integrated Query)

MVVM – архітектурний патерн «модель – представлення – модель представлення» (англ., «Model – View – ViewModel»)

OX – кросинговер з однією точкою схрещування (англ., Order Crossover)

PMX – кросинговер з частковим відображенням (англ., Partially Mapped Crossover)

WPF – бібліотека для створення графічного інтерфейсу (англ., Windows Presentation Foundation)

ВСТУП

Транспортна логістика займає важливе місце у різних галузях та сферах людської діяльності: доставка вантажу споживачам, розвезення товарів за пунктами, поштові відправки, збирання та транспортування виробничої сировини. З метою збереження конкурентоздатності підприємствам доводиться концентруватись на максимальному задоволенні клієнтів. Воно полягає у високій якості товарів і послуг. При цьому ціни на товари найбільше залежать від логістичних витрат [1]. Таким чином, будь-яке підприємство зацікавлене у плануванні раціональних маршрутів різного призначення, що дозволить максимально дешево та швидко переміщувати продукцію. Зазвичай, це призводить до вирішення певного варіанту ЗМТ (задачі маршрутизації транспорту).

ЗМТ відноситься до класу NP-повних задач, що накладає обмеження розмірності задачі для доцільного застосування точних детермінованих алгоритмів [2]. Через це для вирішення завдань великої розмірності отримали поширення евристичні та мета-евристичні алгоритми, а також їх можливі комбінації. На практиці часто виникають додаткові обмеження та умови, які призводять до нових постановок завдань, для вирішення яких необхідні нові математичні моделі та алгоритми.

Актуальність роботи обумовлена тим, що навіть за наявності великої кількості вже реалізованих алгоритмів, необхідно й надалі реалізовувати та покращувати дані алгоритми для пошуку найкращого результату, так як застосування автоматизованих систем у транспортній логістиці – один із способів заощадити ресурси [1, 2].

В роботі приділено увагу гібридним методам, що дозволяють знаходити рішення ЗМТ з обмеженям на вантажопідйомність та часовими вікнами. Розглянуті та досліджені існуючі методи та алгоритми вирішення ЗМТ. Реалізована програмна система для тестування розглянутих методів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис ЗМТ

ЗМТ є однією з широко відомих задач комбінаторної оптимізації [3]. У класичному варіанті ЗМТ є необмежений парк ідентичних транспортних засобів і кінцева множина споживачів. Потрібно побудувати такий набір маршрутів, щоб усі клієнти були обслуговані, а сумарна пройдена відстань була мінімальною. Кожен маршрут починається та закінчується у депо.

Вперше поняття ЗМТ з’явилося в статті [4]. Джордж Данциг та Джон Рамсер в 1959 році запропонували математичне формулювання і алгоритмічний підхід до вирішення практичної задачі постачання бензину від кінцевої станції магістрального трубопроводу до великої кількості обслуговуючих терміналів. Через декілька років Кларк і Райт [5] вперше включили більше одного транспортного засобу в постановку задачі, а також запропонували більш ефективну евристику на основі жадібного алгоритму. З того часу було запропоновано велику кількість моделей та алгоритмів, присвячених пошуку точного та наближеного вирішення багатьох варіантів даної задачі, згодом об’єднаних у загальну групу ЗМТ.

1.1.1 Класифікація ЗМТ

В реальних умовах, в ЗМТ часто виникають додаткові обмеження та умови, що не дозволяє використовувати класичні методи та вимагає вдосконалення існуючих математичних моделей та алгоритмів рішення [6]. Серед більшості ЗМТ з додатковими умовами та обмеженнями, можна виділити наступні класи, які розглянуто далі.

Асиметрична ЗМТ відрізняється від симетричної тим, що вона моделюється орієнтованим графом. Відповідно, матриця цін є асиметричною.

У більшості наукових праць, присвячених дослідженню ЗМТ, розглядається симетричний варіант задачі, що передбачає симетричність матриці цін. Таке припущення часто не співвідноситься з реальними умовами. На практиці найкоротший шлях між двома точками, як правило, відрізняється залежно від напряму [5].

ЗМТ з обмеженням вантажопідйомності схожа на класичну, але з умовою, що обсяг вантажу на кожному маршруті не повинен перевищувати задану величину, однакову для всіх транспортних засобів. Фіксований парк транспортних засобів однакової місткості з загальним депо повинен з мінімальними витратами задовольнити попит на товар кожного споживача і при цьому не перевищити власну вантажопідйомність [3, 6].

ЗМТ з обмеженням відстані враховує те, що довжина кожного маршруту не може перевищувати задане значення. Такий варіант задачі зручно використовувати, коли транспортний засіб може заправлятися тільки в депо та необхідно враховувати обмежений об'єм паливного бака [4].

ЗМТ з часовими вікнами використовується при обмеженому часовому діапазоні прийому або вивезення продукції [2]. Для виконання запиту кожного споживача існує відомий проміжок часу. У разі прибуття раніше нижньої межі інтервалу, враховується час очікування її настання. Прибуття пізніше верхньої межі інтервалу неприпустимо, однак у деяких випадках обслуговування клієнта в часовому вікні не є критично важливою умовою, але її порушення додає штрафне значення до цільової функції. Іноді враховується сервісний час, необхідний для обслуговування клієнта.

ЗМТ з декількома депо використовуються для таких випадків як розподіл продукції між кількома постачальниками загальній групі споживачів [5, 6]. Ця задача є узагальненням класичної ЗМТ, в якому існує більше одного депо, з яких здійснюється обслуговування клієнтів, при цьому кожен транспортний засіб починає і закінчує маршрут у власному депо. Такий варіант задачі є складнішим, оскільки виникає необхідність розподілити споживачів по різним депо. Для цього зазвичай доводиться

визначати, які клієнти відносяться до кожного депо або використовувати двофазні алгоритми, в яких спочатку здійснюється розбиття графа на підграфи, а потім окремо будуються маршрути всім депо.

У періодичної ЗМТ на відміну від класичної використовується розширений період планування до кількох днів [6]. Для різних клієнтів потрібна різна кількість відвідувань у вказаний період, при цьому дні обслуговування заздалегідь не визначені, але задано список можливих дат відвідування кожного споживача. Таким чином, ЗМТ вирішується для кожного дня планування. У ряді випадків ця особливість має важливе значення, зокрема, при вирішенні проблеми збору відходів.

ЗМТ з вивозом та доставкою представляє собою задачу, в якій клієнти можуть як отримувати, так і відправляти товари [5]. Таким чином, товари не перевозяться від одного клієнта до іншого, а відправляються з депо клієнтам або надходять від клієнтів в депо.

У ЗМТ з роздільною доставкою відсутня заборона на багаторазове відвідування споживача. Тобто один і той же клієнт може бути обслугований багато разів кількома транспортними засобами, якщо це дозволяє зменшити загальні витрати. Такий різновид ЗМТ виправданий, коли запити клієнтів перевищують вантажопідйомність транспортних засобів [2, 3].

Стохастична ЗМТ представляє собою задачу з випадковими даними. Один або кілька компонентів задачі можуть мати випадкову поведінку:

- попит кожного клієнта пов’язаний з заданим ймовірнісним розподілом, замість конкретного значення, а реальне значення визначається лише після прибуття транспортного засобу;
- множина споживачів не є точно відомою, кожен клієнт існує з деякою ймовірністю;
- час переміщення (відстань) між пунктами не детерміновано;
- час обслуговування кожного клієнта не детерміновано.

Нечітка ЗМТ використовується у випадку, коли немає можливості отримати точні значення запитів, часу шляху, кількості та розташування

клієнтів, меж часових вікон та інших величин. Такі задачі містять елементи невизначеності та суб'єктивності, тому при їх моделюванні використовують методи теорії нечітких множин [6].

Динамічна ЗМТ передбачає те, що можуть відбуватися деякі зміни параметрів задачі у процесі її вирішення. До таких зазвичай належать поломки транспортних засобів, дорожні затори, нові замовлення, раптові дзвінки тощо. Зазвичай розглядається випадок, коли нові клієнти можуть з'являтись протягом дня, але після того як транспортний засіб залишить депо. Таким чином, в міру обслуговування клієнтів умови задачі змінюються та необхідно динамічно перераховувати маршрути з урахуванням нової інформації, що надходить [5, 6].

Відкрита ЗМТ – це незамкнений варіант задачі, в якому нема потреби повертатися в депо в кінці маршруту. Таким чином, віддаленість останнього відвіданого клієнта від депо не впливає на загальну вартість рішення. Найчастіше ця задача використовується, коли підприємство не має власного парку транспортних засобів, а укладає контракт із зовнішніми кур'єрами [1, 2]. Відповідно, наймані транспортні засоби не повинні повертатися до розподільного центру підприємства (депо) і можуть закінчити маршрут у будь-якому місці.

ЗМТ з різнорідним парком транспортних засобів представляє собою задачу, у якій клієнти обслуговуються кількома типами транспортних засобів з різними характеристиками, такими як вантажопідйомність, швидкість, вартість використання тощо [6].

При виборі конкретного класу ЗМТ потрібно враховувати кінцеві цілі та вимоги, які необхідно досягти за допомогою вирішення задачі (мінімізація витрат на транспортування, максимізація кількості замовлень, зменшення часу виконання, максимальне врахування часових вікон клієнтів). Також потрібно проаналізувати наявні ресурси (парк транспортних засобів, кількість депо, типи вантажу) і технічні обмеження (вантажопідйомність, швидкість руху, розмір паливного баку транспорту).

1.1.2 Математична модель ЗМТ

ЗМТ визначена у вигляді повного направленого графа $G = (V, H, c, t)$, де $V = \{0, 1, \ldots, n\}$ – множина вершин [4]. Кожна вершина з індексом $i \in V/\{0\}$ відповідає клієнту, що має невід'ємний попит $d_i \leq Q$ та часове вікно $[a_i, b_i]$, а вершина 0 являє собою депо (рисунок 1.1). В депо розміщений фіксований парк із p транспортних засобів з вантажопідйомністю Q. Матриця $H = \{(i, j) : i, j \in V, i \neq j\}$ описує множину дуг, що відповідає транспортній мережі між вузлами. З кожною дугою пов'язана вартість переміщення c_{ij} та час переїзду t_{ij}, де $(i, j) \in H$.

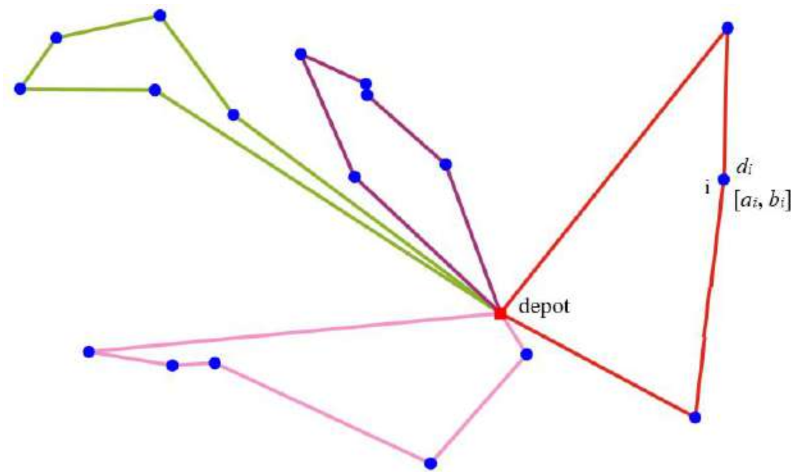

Рисунок 1.1 – Граф ЗМТ

ЗМТ потребує визначення набору маршрутів m, сумарна вартість яких зводиться до мінімуму і така, що:

- маршрути починаються та закінчуються в депо;
- кожен клієнт відвідується тільки один раз в одному маршруті;
- загальна потреба клієнтів, що обслуговуються в одному маршруті, не перевищує пропускну здатність Q;
- кількість наявних маршрутів m не перевищує кількість доступних транспортних засобів p;

- обслуговування клієнта повинно здійснюватися в рамках його часового вікна.

Нехай бінарна змінна x_{rij} визначає переміщення транспортного засобу $r \in \{1, 2, \ldots, p\}$ дугою (i, j) в рішенні задачі, а час початку обслуговування вершини i позначається як τ_i.

Математична модель ЗМТ [4, 5] з урахування додаткових обмежень може бути описана наступним чином. Мінімізація цільової функції:

$$\sum_{r=1}^{p}\sum_{i=0}^{n}\sum_{j=0,i\neq j}^{n} c_{ij}x_{rij} \rightarrow \min. \quad (1.1)$$

З урахуванням умов:

$$\sum_{r=1}^{p}\sum_{i=0,i\neq j}^{n} x_{rij} = 1, \forall j \in \{1,\ldots,n\}, \quad (1.2)$$

$$\sum_{j=1}^{p} x_{r0j} = 1, \forall r \in \{1,\ldots,p\}, \quad (1.3)$$

$$\sum_{i=0}^{n}\sum_{j=1,i\neq j}^{n} d_j x_{rij} \leq Q, \forall r \in \{1,\ldots,n\}, \quad (1.4)$$

$$a_i \leq \tau_i \leq b_i, \forall i \in n, \quad (1.5)$$

$$x_{rij} \in \{0,1\}, \forall r \in \{1,\ldots,p\}, i, j \in \{0,\ldots,n\}, i \neq j. \quad (1.6)$$

Мета функції (1.1) полягає у мінімізації загальних витрат на перевезення. Обмеження моделі (1.2) гарантує, що кожен клієнт може бути обслужений лише одним транспортним засобом. Обмеження (1.3) забезпечує те, що кожен транспортний засіб може покинути депо тільки один раз. В (1.4) визначається обмеження вантажопідйомності транспорту. Умова відвідування клієнтів в межах відповідного часового вікна формалізована за допомогою обмеження (1.5). Вирази (1.6) визначають області значень змінних.

1.1.3 Актуальність проблеми вирішення ЗМТ

В умовах реального часу оптимізація доставки вантажів є актуальною проблемою для великої кількості підприємств. Зі збільшенням обсягів виробництва зростає складність розподілу ресурсів, транспортні засоби можуть використовуватись неефективно, відбувається зростання конкуренції і, як наслідок, може знижуватись якість послуг [1].

У зв’язку з цим фірмам потрібна автоматизація розподілу ресурсів, попит на транспортно-логістичні послуги підвищується щороку. Особливо актуальним є вирішення ЗМТ при внутрішніх перевезеннях містом, так як в даний час 75% від структури вантажоперевезень становлять перевезення дрібно партійних товарів [7].

Мінімізація витрат на перевезення стає серйозною конкурентною перевагою як серед представників послуг вантажоперевезень, так і серед виробників товарів. При вирішенні практичних задач планувальникам потрібна автоматизована система, яка може окрім знаходження рішення для ЗМТ, враховувати наявні обмеження та умови, у зв’язку з чим виникають додаткові складнощі. Всі різновиди ЗМТ є NP-важкими, тобто на даний момент для їх рішення не розроблені алгоритми з поліноміальним часом роботи, але й не доведено, що таких алгоритмів не існує [8].

Одним з перспективних напрямків для покращення способів знаходження рішень ЗМТ є розробка гібридних методів на основі поєднання вже існуючих алгоритмів [8, 9]. Гібридизація дозволяє поєднувати переваги та позбутися недоліків різних методів, обстежувати простір рішень, який неможливо розглянути за допомогою класичних методів, а також врахувати конкретні вимоги та обмеження задачі.

Таким чином, формалізація ЗМТ та розробка гібридних методів для їх вирішення є актуальним напрямом досліджень у галузі комбінаторної оптимізації.

1.2 Методи вирішення ЗМТ

На даний момент, відома достатньо велика кількість алгоритмів для рішення ЗМТ [10]. Їх поділяють на точні, евристичні та метаевристичні (рисунок 1.2). Так як ЗМТ є NP-важкою, то для її вирішення найбільше використовуються евристичні та метаевристичні алгоритми, тобто ті, які знаходять рішення наближені до найкращого та за прийнятний час.

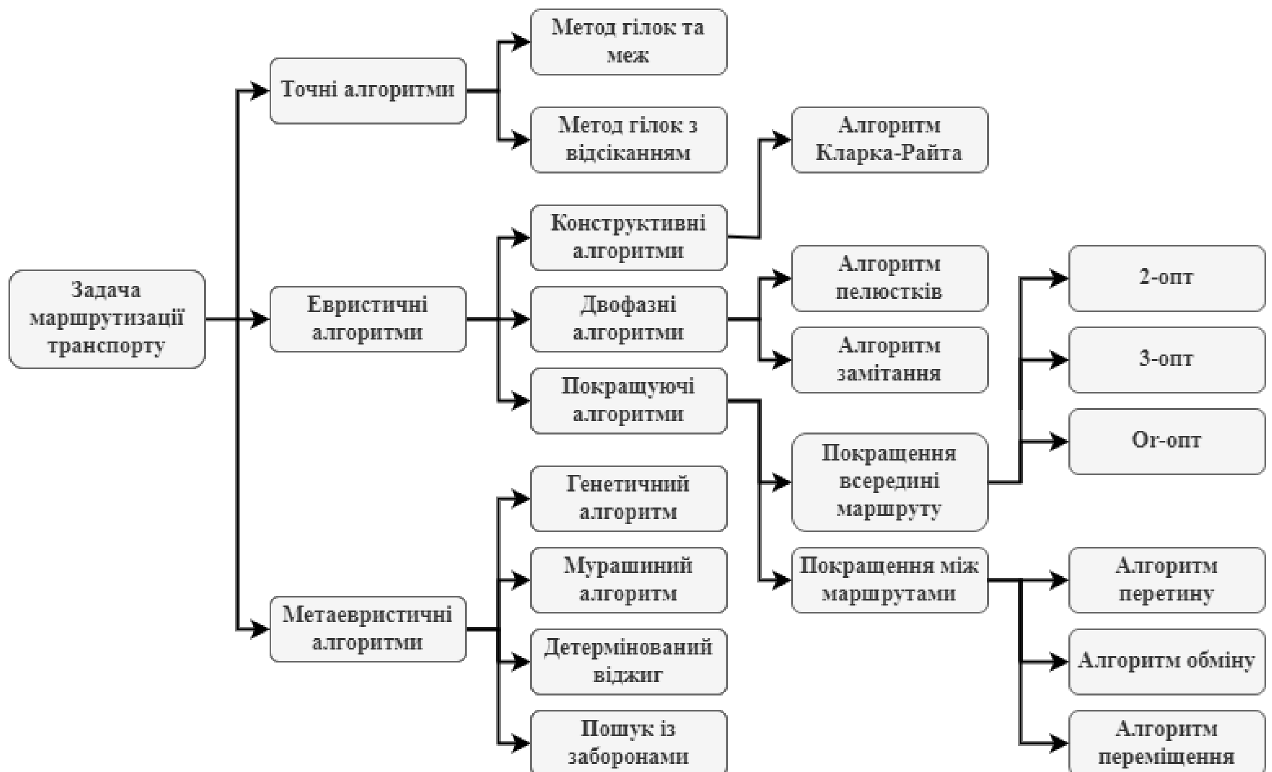

Рисунок 1.2 – Алгоритми для вирішення ЗМТ

1.2.1 Точні алгоритми

Точні методи вирішення ЗМТ дозволяють знайти найкраще рішення. У загальному випадку виконується обчислення всіх можливих рішень, доки не буде досягнуто найкращого з них. Такий підхід призводить до того, що точні алгоритми не можуть застосовуватись для задач з великою кількістю вхідних даних, так як вони вимагають значної кількості обчислювальних ресурсів та

можуть виконуватись неприпустимо багато часу. Таким чином, точні методи не підходять для вирішення реальних задач, які зазвичай мають складні обмеження та велику кількість клієнтів. Але якщо модифікувати такі алгоритми, виключивши з розгляду частину рішень, то їх можна застосовувати у поєднанні з іншими алгоритмами (наприклад, для генерації початкового набору рішень) [7].

Одним з таких методів є метод гілок і меж [11]. Цей метод базується на розбитті задачі на менші підзадачі шляхом фіксування змінної розгалуження. Надалі кожна з цих підзадач може бути розгалужена на ще менші підзадачі. Процес розгалуження продовжується до тих пір, поки не буде знайдений найкращий розв'язок.

Змінна розгалуження є ключовим елементом методу гілок та меж. В процесі розгалуження обирається змінна, що має найбільший вплив на цільову функцію. В кожній з підзадач змінна має своє значення (в ЗМТ воно визначає, чи включається обране переміщення в маршрут). Таким чином, змінна розгалуження дозволяє розбити вихідну задачу на менші підзадачі, що дає можливість знизити складність обчислень та прискорити пошук найкращого розв'язку.

Іншим алгоритмом є метод гілок з відсіканням [8, 12], що заснований на наступному принципі. Якщо нижня межа значень функції на підзадачі A дерева пошуку більше, ніж верхня межа на будь-якій раніше переглянутої підзадачі B, то A може бути виключена з подальшого розгляду (правило відсіву). Такі виключення дозволяють позбавитись від зайвих підзадач та розв’язків, що значно зменшує обсяг обчислень.

Як правило, мінімальну з отриманих верхніх оцінок записують у глобальну змінну m. Будь-який вузол дерева пошуку, нижня межа якого більша за значення m, може бути виключений з подальшого розгляду. Якщо нижня межа для вузла дерева збігається з верхньою межею, то це значення є мінімумом функції та досягається у відповідній підзадачі.

1.2.2 Евристичні алгоритми

Евристичні алгоритми, які засновані на певних правилах (евристиках), не завжди побудованих зі строгих математичних принципів, у переважній більшості випадків дають рішення, наближене до найкращого. Ці правила використовуються для зменшення області пошуку та зазвичай засновані на досвіді або інтуїції [10]. Вони не завжди гарантують достатньо оптимальний результат, але вони можуть збільшити ймовірність знайти один з найкращих варіантів рішення або наближений до нього.

Евристичні алгоритми поділяються на:

- конструктивні алгоритми, які крок за кроком вибудовують рішення, враховуючи загальну вартість, що розраховується у ході вирішення;
- двофазні алгоритми, що складаються з двох етапів – збирання вершин у групи та побудова маршрутів для кожної групи;
- алгоритми, в яких спочатку відбувається формування допустимого рішення (будь-якого), а потім його покращення шляхом застосування послідовних невеликих змін.

Метод Кларка-Райта належить до конструктивних алгоритмів та є одним з найвідоміших способів рішення ЗМТ [13]. Він заснований на процесі злиття дрібних маршрутів у більші, що проводиться до тих пір, доки є можливість зменшити сумарну вартість об'їзду. Особливу роль при цьому грає поняття «заощадження» – величина зниження загальної вартості рішення, яке отримується при об’єднанні двох маршрутів.

Одним з двофазних алгоритмів є метод пелюсток [14], який виконує побудову великої кількості кластерів, що перекриваються (і маршрутів, які з ними асоційовані), а потім обирає з них найкращу підмножину.

Іншим двофазним методом є алгоритм замітання [15], що використовується для первинної обробки ЗМТ. У процесі його роботи наповнення кластерів визначається поворотом променю, що виходить з депо. Потім для кожного кластера окремо вирішується ЗМТ. У деяких варіантах

алгоритму є фаза подальшої оптимізації, в якій проводиться обмін вершинами між сусідніми кластерами, з коригуванням маршрутів.

Алгоритми, що поступово покращують рішення для ЗМТ, можуть обробляти або один окремий маршрут за раз, або декілька маршрутів [16]. У випадку для одного маршруту застосовуються будь-які алгоритми оптимізації для задачі комівояжера. Для другого варіанту зазвичай використовуються алгоритми, які аналізують структуру з декількома маршрутами та на основі результатів аналізу виконують необхідні зміни.

1.2.3 Метаевристичні алгоритми

Метаевристичні алгоритми на відміну від евристичних здійснюють пошук у просторі всіх можливих варіантів та мають можливості уникнення локальних мінімумів, що збільшує шанс знайти найкращий розв’язок. Метаевристичні методи використовують більш складні алгоритми пошуку, мають ускладнену структуру даних, використовують рекомбінацію та модернізацію рішень [10, 11].

Відмінною рисою таких алгоритмів є наявність великої кількості параметрів, які необхідно підбирати при практичному застосуванні. При цьому в деяких випадках процедуру підбору значень параметрів необхідно виконувати для кожного нового набору даних. Це ускладнює використання даних алгоритмів та потребує високої кваліфікації користувача програмних систем, що побудовані на їх основі.

Метаевристичні алгоритми пропонують компроміс між якістю рішення та обчислювальним часом, а також можуть бути легко адаптовані для задоволення потреб більшості типів ЗМТ з додатковими обмеженнями та умовами [9, 16]. До таких алгоритмів відносяться ГА (генетичний алгоритм), мурашиний алгоритм, детермінований віджиг, пошук з заборонами тощо.

ГА полягають у пошуку найкращого рішення з використанням принципів, схожих на природний відбір у природі [17]. ГА – це загальна

методика вирішення задач, для реалізації якої потрібна відносно невелика кількість інформації про предметну область. Як наслідок цей підхід може бути використаний для достатньо великої кількості погано структурованих задач, де спеціалізовані методи не показують вдалих результатів.

Мурашиний алгоритм базується на імітації природних механізмів самоорганізації мурах, яка здійснюється завдяки взаємодії між усіма мурахами [18]. Кожна мураха є самостійною одиницею, не здатною вирішити завдання без інших мурах. Самоорганізація дозволяє знайти рішення всією системою, завдяки низькорівневій взаємодії її елементів. Система не має централізованого керування, її взаємодією є непрямий обмін, реалізований за допомогою хімічних речовин – феромонів, що залишені на шляху після переміщення мурахи. Мурахи проходять дорогами попередників та посилюють концентрацію феромона на ньому. Залежно від кількості феромона на шляху, шлях стає більш або менш кращим для проходження мурахи по ньому. Шлях, що містить найбільшу кількість феромонів стає домінуючим та призводить до знаходження найкращого розв'язку.

Метод детермінованого віджигу, як і метод імітації віджигу, заснований на імітації фізичного процесу охолодження металевого тіла після його нагрівання до температури плавлення [19]. Передбачається, що процес протікає при температурі, що поступово знижується. Загальна ідея методів полягає в переході атома з одного стану в інший, внаслідок нестійкості кристалічних решітки. Атом може переходити в стан з меншим рівнем енергії або залишатися в тому самому з деякою ймовірністю. Метод імітації віджигу працює повільніше внаслідок стохастичного вибору поточного рішення. Детермінований віджиг дозволяє вирішити цю проблему, використовуючи детерміновані критерії вибору. Виділяють два типи цього алгоритму: методи руху від рекорду до рекорду та порогового прийняття.

Алгоритм пошуку із заборонами [20] характеризується формуванням списку рішень, що поповнюється на кожній ітерації. На початковому етапі список рішень порожній, створюється початкове рішення задачі. На

наступних етапах отримане рішення додається до списку заборон і стає поточним, формується його околиця і проводиться локальний пошук рішення у даній околиці. Знайдене локальне рішення також додається до списку заборон і стає поточним. Процес продовжується, доки не буде виконано умову зупинки. Метод дозволяє рухатися від одного локального екстремуму до іншого, не застряючи в них, з метою досягти глобального оптимуму.

1.3 Аналіз існуючих рішень

Точні методи вирішення ЗМТ зазвичай не використовуються через неможливість їх застосування для великої кількості клієнтів або специфічних умов та обмежень, що зустрічаються на практиці [11]. Евристичні методи можуть потрапляти в локальні мінімуми та не враховувати різні обмеження, в наслідок чого, не завжди знаходять рішення, достатньо наближене до найкращого. Враховуючи всі ці недоліки, найчастіше застосовують метаевристичні алгоритми або їх модифікації [16, 19].

Для рішення ЗМТ з обмеженням на вантажопідйомність транспортних засобів та часовими вікнами у [21] пропонується реалізація ГА з розширеним представленням особин (рішень). Розширення полягає у тому, що всі маршрути рішення представляються як один, але з багатократним включенням в нього депо, яке служить роздільником між двома сусідніми маршрутами. Класичний оператор схрещування PMX та оператор мутації RAR були адаптовані для використання в нових умовах. На кожній ітерації вони застосовуються до тих пір, доки не буде отримано необхідну кількість рішень, що задовольняють обмеженням, а інші варіанти відкидаються. Автор застосовував спеціальний оператор локального спуску (local descend operator), заснований на чотирьох різних типах зміни порядку виконуваних кроків, та викликав його тільки для найкращих особин у поточній популяції. Такий варіант реалізації дозволяє збільшити швидкодію алгоритму, але не призводить до покращення знайдених рішень.

В [22] наведено ще один варіант ГА, що може вирішувати ЗМТ з часовими вікнами. Особливість запропонованого методу в тому, що в ньому використовується класичний підхід із фрагментацією загального маршруту (генетичного ланцюжку), отриманого після рішення ЗК на загальній множині вершин. Це дозволяє використовувати традиційну схему кодування особин без розділення маршрутів багатократним включенням депо. Для поділу на маршрути кожного окремого транспортного засобу застосовується процедура замітання, починаючи роботу з першої вершини загального маршруту. Перехід до наступного маршруту відбувається, коли перевищується обмеження на вантажопідйомність. Таким чином генетичний ланцюжок може бути завжди перетворений на допустиме рішення ЗМТ.

В роботі [23] реалізовано алгоритм мурашиних колоній для вирішення ЗМТ. Пропонується комплексний підхід до використання мурашиних колоній у поєднанні з оптимізацією 2-опт і застосуванням ймовірнісного підходу прийняття рішень, що нагадує модельований віджиг. Автор приділяє особливу увагу впливу різних параметрів на якість одержуваних рішень.

В іншому варіанті [24], автори поєднали алгоритм мурашиних колоній з оптимізацією 2-опт кожного маршруту на кожній ітерації до виконання процедури оновлення феромонів на шляхах. Цей алгоритм може враховувати вантажопідйомність та різні комбінації обмежень при виборі наступної вершини для обходу мурахою. Також в роботі розглядається кілька напрямків покращення розробленого алгоритму:

- заміна поняття вантажопідйомності (через те, що його використання призводить до дорогих обчислень), використаного в процесі вибору вершини, на поняття заощадження у параметричному вигляді;
- вибір для подальшого відвідування тільки частини від усіх вершин;
- використання фіксованої кількості найкращих рішень та врахування рангу знайдених рішень для оновлення феромонів.

Наведені зміни дозволяють зменшити час обчислень та покращити знайдені розв'язки, але ускладнюють налаштування параметрів алгоритму.

1.4 Постановка задачі

Метою роботи є розробка гібридних методів вирішення ЗМТ з урахуванням обмеження на вантажопідйомність та часових вікон, а також програмна реалізація розроблених методів та проведення експериментальних досліджень.

Для тестування та дослідження алгоритмів необхідно створити програмний застосунок, який дозволить:

- генерувати ЗМТ з заданими параметрами (кількість та потреби клієнтів, вантажопідйомність та кількість наявних транспортних засобів, межі для значень в матрицях вартостей та часу переміщень, межі для часових вікон споживачів), а також зберігати їх як окремі файли або завантажувати наявні файли в програму;
- налаштовувати параметри розроблених алгоритмів та фітнес функції, а також використовувати обрані методи для вирішення згенерованих або завантажених задач;
- запускати набори налаштованих алгоритмів для вирішення наборів згенерованих або завантажених ЗМТ, з можливістю подальшого перегляду та порівняння отриманих результатів роботи (у вигляді таблиць, графіків та стовпчастих діаграм);
- отримувати детальну інформацію про внутрішню роботу обраного методу (статистичні дані, характеристики знайдених рішень, стан алгоритму на кожному етапі процесу пошуку, час роботи окремих компонентів, з яких складається алгоритм).

У ході досліджень вимірюються такі критерії ефективності розроблених гібридних методів, як час роботи алгоритму, загальна вартість переїздів у всіх маршрутах, кількість транспортних засобів або їх наповнення вантажем, значення фітнес-функції у знайдених рішеннях тощо. Проводиться порівняльний аналіз результатів роботи розроблених алгоритмів на різних наборах даних.

2 ВИКОРИСТОВУВАНІ МЕТОДИ ВИРІШЕННЯ

ЗМТ з обмеженням на вантажопідйомність та часовими вікнами є однією з варіантів задач, що часто застосовуються на практиці [21, 25]. Існує велика кількість різноманітних методів вирішення даної задачі. В реальних умовах найчастіше застосовуються різні модифікації метаевристичних алгоритмів або гібридні методи на їх основі у поєднанні з класичними методами [20].

В даній роботі для дослідження та реалізації гібридного підходу для вирішення ЗМТ обрано використання жадібного алгоритму та його модифікації, алгоритму Кларка-Райта, методу гілок з відсіканням, генетичних алгоритмів, а також алгоритмів мурашиної колонії.

2.1 Жадібний алгоритм

Жадібний алгоритм – метод, в якому приймається локально найкраще рішення на кожному етапі, з припущенням, що такий підхід призведе до глобально найкращого рішення в кінці роботи алгоритму. При вирішенні ЗМТ алгоритм зазвичай не знаходить найкращий розв’язок, але може формувати початкове рішення для інших методів, які в ході своєї роботи будуть його покращувати [16, 26].

Алгоритм адаптовано під умови задачі, з метою врахування додаткових обмежень ЗМТ (рисунок 2.1). Врахування вантажопідйомності та часових вікон відбувається на етапі відбору клієнтів для розгляду. Відбір відбувається в три етапи, на кожному з яких враховується обмеження вантажопідйомності. Перехід до наступного етапу відбувається, якщо на попередньому немає відібраних клієнтів. На першому етапі обираються клієнти для яких задовольняється умова часового вікна, на другому – ті, що не мають часового вікна, на третьому – відбір відбувається без урахування часових вікон.

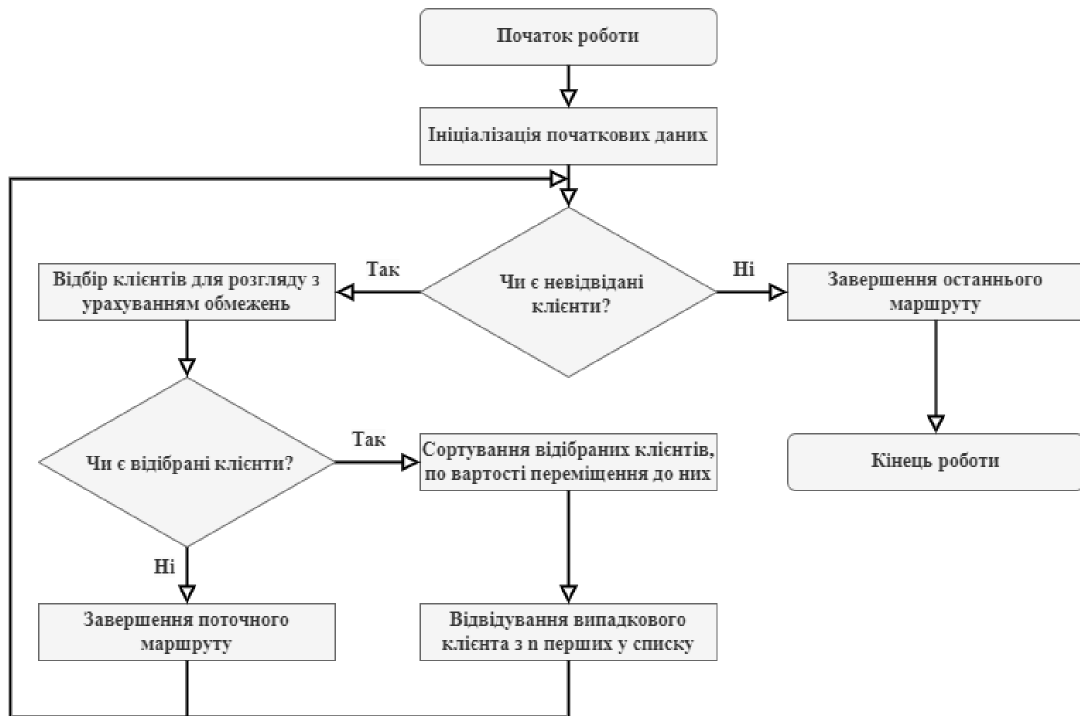

Рисунок 2.1 – Блок-схема жадібного алгоритму рішення ЗМТ

Оскільки алгоритм використовується для формування початкового набору рішень, його логіку було розширено з метою збільшення різноманітності у наборі знайдених рішень. Різноманітність рішень досягається за рахунок випадкового вибору клієнта для відвідування з n найкращих, де n задається як параметр алгоритму.

2.2 Модифікація жадібного алгоритму

Класичний жадібний алгоритм для вирішення ЗМТ полягає у виборі на кожному етапі переміщення з найменшою вартістю (без урахування того, як завантажені транспортні засоби) [23]. На відміну від класичного варіанту, дана модифікація заснована на максимізації завантаження транспортних засобів з метою мінімізації їх кількості (рисунок 2.2). При цьому модифікація не враховує вартості переміщень та часові вікна.

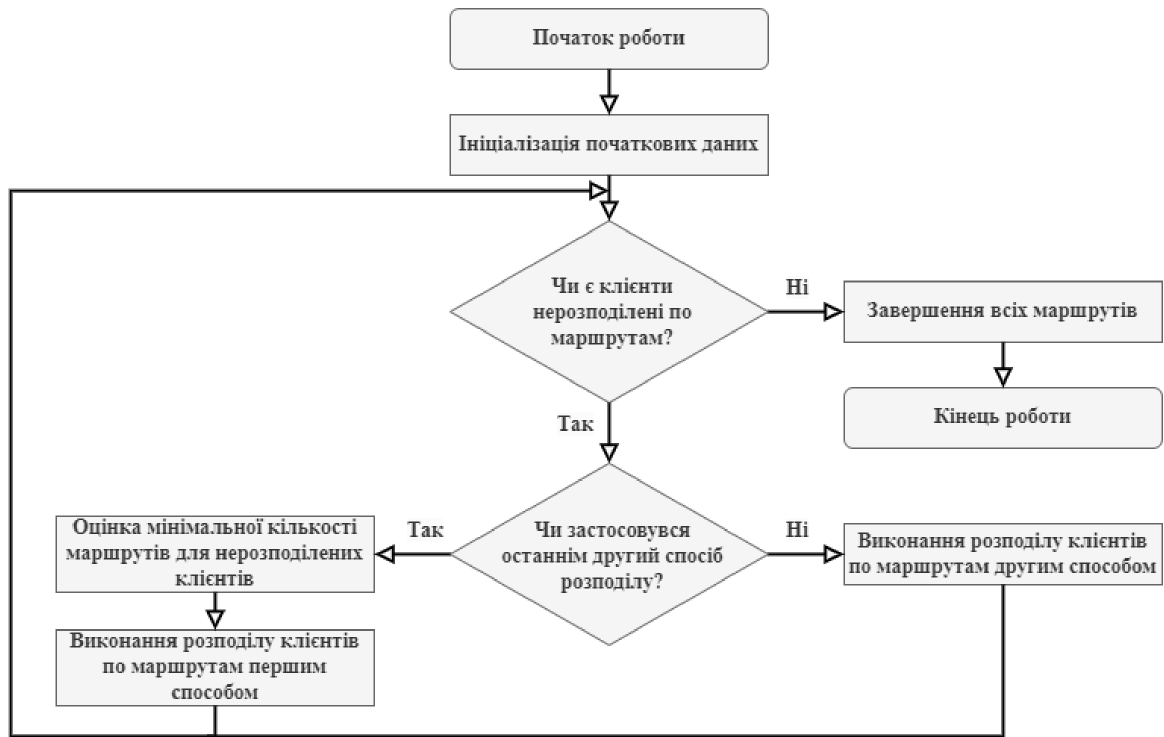

Рисунок 2.2 – Блок-схема модифікації жадібного алгоритму рішення ЗМТ

Етап оцінки мінімальної кількості маршрутів для нерозподілених клієнтів представляє собою розрахунок кількості маршрутів згідно (2.1), по яким будуть розподілятись клієнти першим способом.

$$M_{\min} = round(\frac{1}{Q}\sum_{i=1}^{n} d_i x_i), \tag{2.1}$$

де Q – вантажопідйомність транспортного засобу;

n – загальна кількість клієнтів;

d – потреба клієнта;

x – бінарна змінна, що позначає, чи є клієнт нерозподіленим.

Перший спосіб розподілу по маршрутам заснований на ітеративному проходженні по нерозподіленим клієнтам в порядку спадання потреб та розподіленні клієнтів по маршрутам за встановленими правилами (рисунок 2.3). Потрібно встановити перший маршрут поточним. Якщо

можливо розмістити клієнта в поточний маршрут, то це здійснюється та поточним маршрутом стає наступний у списку (проходження по списку відбувається циклічно). Інакше необхідно знайти підходящий маршрут методом перебору та помістити клієнта в нього.

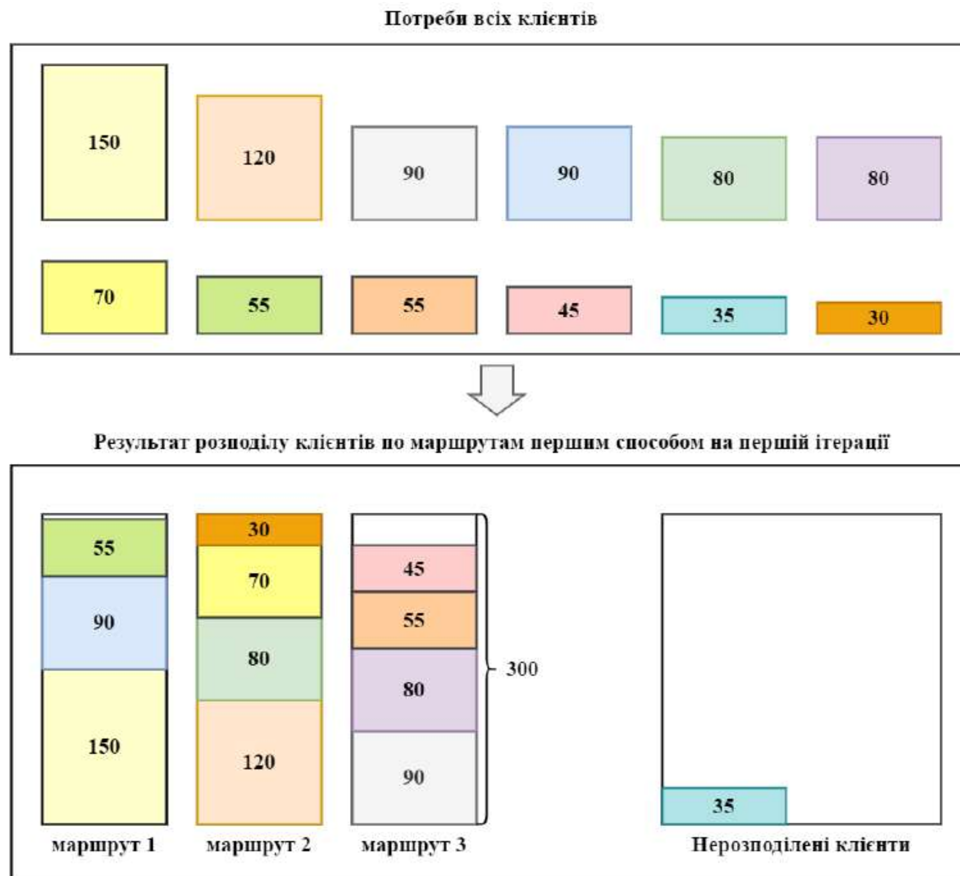

Рисунок 2.3 – Приклад роботи розподілу першим способом

Розподіл клієнтів другим способом потрібен для того, щоб спробувати розподілити клієнтів (не розподілених першим способом) в існуючі маршрути, звільнивши місце для них за рахунок перестановок розподілених клієнтів між маршрутами. Вибір клієнтів для перестановки здійснюється методом перебору, це не призводить до дорогих обчислень, так як більшість маршрутів відкидаються з розгляду через те, що не містять потенційного клієнта, перестановка якого може звільнити місце. Якщо не вдалося розподілити клієнтів другим способом, то потрібно повернутись до оцінки

мінімальної кількості маршрутів та виконання першого способу, але вже тільки для тих клієнтів, що залишились нерозподіленими.

Під час етапу завершення всіх маршрутів, кожному маршруту додається кінцевий пункт (депо). Під час роботи алгоритму маршрути не замикаються, так як потенційно до них можна розподілити клієнта.

2.3 Алгоритм Кларка-Райта

Алгоритм Кларка-Райта (рисунок 2.4) заснований на злитті дрібних маршрутів у більші, спираючись на величину збереження (зниження загальної вартості рішення, отриманого при об'єднанні двох маршрутів) [5, 27]. Алгоритм виконується, доки є можливість покращувати рішення. Даний метод в процесі вирішення не враховує часові вікна клієнтів, але не дозволяє порушувати обмеження на вантажопідйомність транспортних засобів.

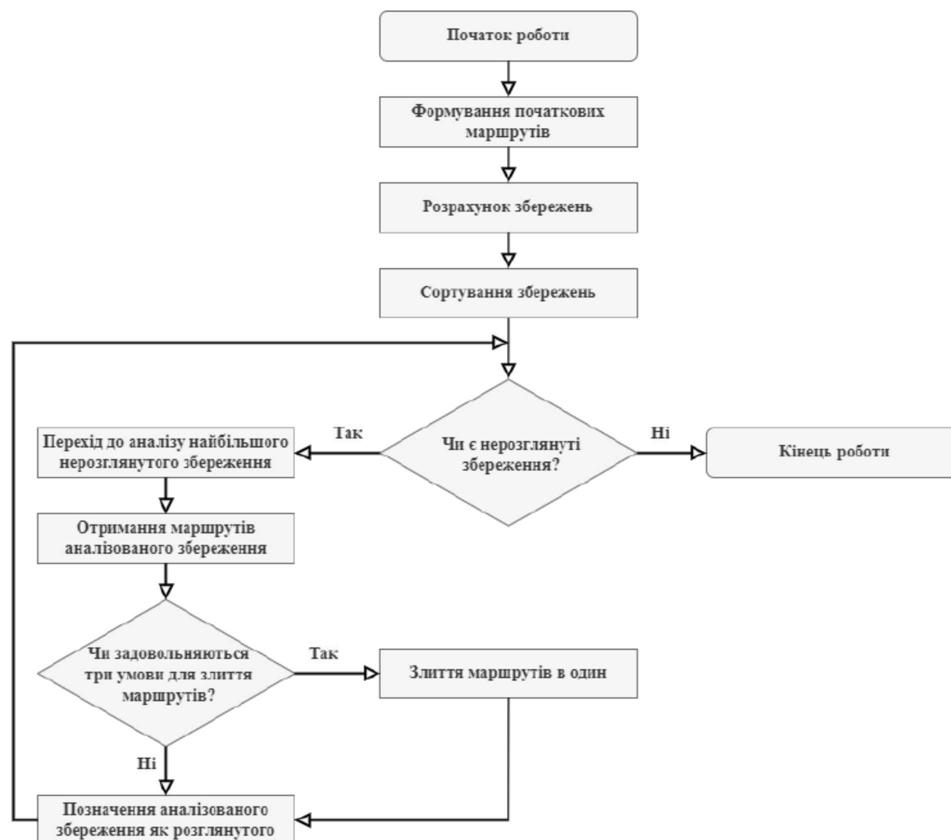

Рисунок 2.4 – Блок-схема алгоритму Кларка-Райта для рішення ЗМТ

На етапі формування початкових маршрутів створюється рішення, в якому кожен маршрут містить тільки одного клієнта. Розрахунок величини збереження (2.2) проводиться для всіх можливих пар маршрутів, що належать початковому рішенню [5].

$$saving_{ij} = c_{i0} + c_{0j} + c_{ij}\,, \tag{2.2}$$

де c – вартість переміщення.

Після розрахунку всіх збережень відбувається їх сортування у порядку зменшення, з метою подальшого ітерування по ним.

Для виконання процесу злиття маршрутів в один необхідно виконання трьох умов:

- маршрути повинні мати крайніми вершинами (без урахування депо) ті, для яких розраховувалось збереження;
- маршрути не повинні бути об’єднаними (це можливо у випадку, коли раніше виконалося злиття цих маршрутів при розгляді іншого збереження);
- при злитті маршрутів не повинно порушуватись обмеження вантажопідйомності.

Алгоритм знаходить рішення достатньо наближені до найкращого коли в задачі задана симетрична матриця вартостей переміщень. У випадку асиметричної матриці, результати роботи алгоритму значно гірші. Незважаючи на це, алгоритм доцільно використовувати для генерації початкового рішення в інших методах.

2.4 Метод гілок з відсіканням

Метод гілок з відсіканням [11, 28] схожий на метод гілок та меж, але на відміну від нього дозволяє відсікати з розгляду не перспективні гілки, що зменшує час роботи алгоритму. Ще одним способом зменшити час обчислень (а також обсяг використовуваної пам’яті) є модифікація таким чином, щоб

він не повертався до раніше розглянутих гілок, якщо оцінка поточної гілки показала, що вона не містить найкращого рішення. В результаті виходить реалізація алгоритму (рисунок 2.5), що підходить для гібридизації з іншими методами, так як знаходить рішення за прийнятний час, хоча й не обов'язково найкраще.

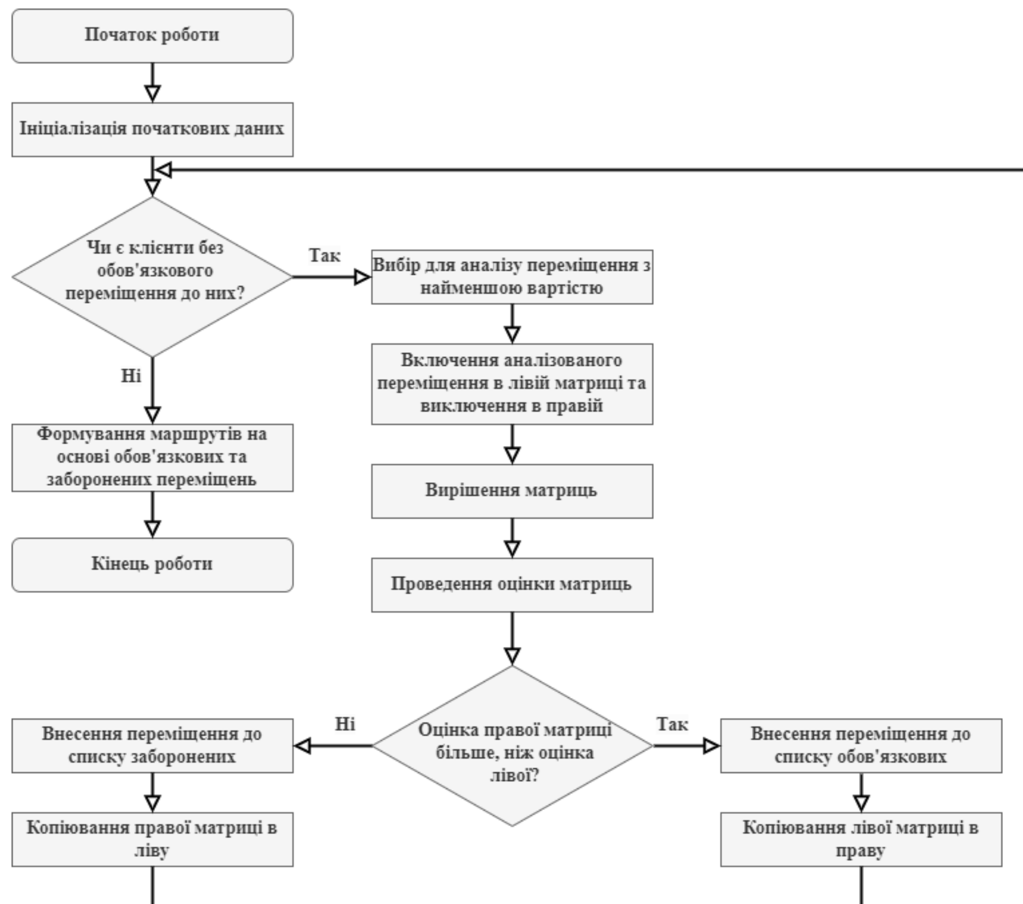

Рисунок 2.5 – Блок-схема методу гілок з відсіканням для рішення ЗМТ

Для створення кінцевого рішення алгоритм повинен сформувати список обов'язкових та заборонених переміщень. Щоб це зробити потрібно на кожній ітерації обирати переміщення з найменшою вартістю, яке буде розділяти область допустимих рішень на дві гілки: з включенням (ліва матриця) та виключенням (права матриця) обраного переміщення. Після цього необхідно провести оцінку матриць та вирішити їх. На основі отриманих оцінок внести переміщення у відповідний список.

Вирішення матриці представляє собою знаходження мінімального значення в кожному рядку матриці та віднімання знайденого значення від кожного елементу в рядку. При цьому значення мінімальних елементів зберігаються для подальшого використання при оцінці матриці. Потім дана процедура повторюється для стовпців матриці. Оцінкою матриці є сума мінімальних елементів кожного рядку та стовпця матриці.

При формуванні маршрутів на основі обов’язкових та заборонених переміщень необхідно враховувати існуючі обмеження, зокрема вантажопідйомність транспортних засобів.

2.5 Гібридний генетичний алгоритм

ГА є еволюційним алгоритмом, що дозволяє вирішувати ЗМТ з різними складними обмеженнями або їх комбінаціями [25, 29]. ГА може бути вдосконалений шляхом гібридизації з класичними алгоритмами, поєднуючи переваги різних методів.

Для вирішення ЗМТ за допомогою ГА потрібно визначити, як будуть кодуватись особини (рішення). Оскільки рішенням є сукупність маршрутів транспортних засобів для обслуговування клієнтів, традиційне бітове представлення особин можна замінити на набір маршрутів, які описуються у вигляді послідовностей чисел, де кожне число позначає вершину. Положення чисел у ланцюгу відповідає порядку вершини в маршруті. Кожен маршрут транспортного засобу починається та закінчується в депо. Генетичні оператори кросинговеру та мутації потрібно модифікувати, щоб вони були сумісні з обраним способом кодування рішень.

Процес еволюції (рисунок 2.6) починається із створення початкової популяції особин (рішень) за допомогою одного з класичних методів, після чого здійснюється перевірка досягнення одного з критеріїв зупинки алгоритму. Якщо критерій зупинки досягнутий, то алгоритм завершує роботу, інакше – застосовуються оператори схрещування та мутації [25].

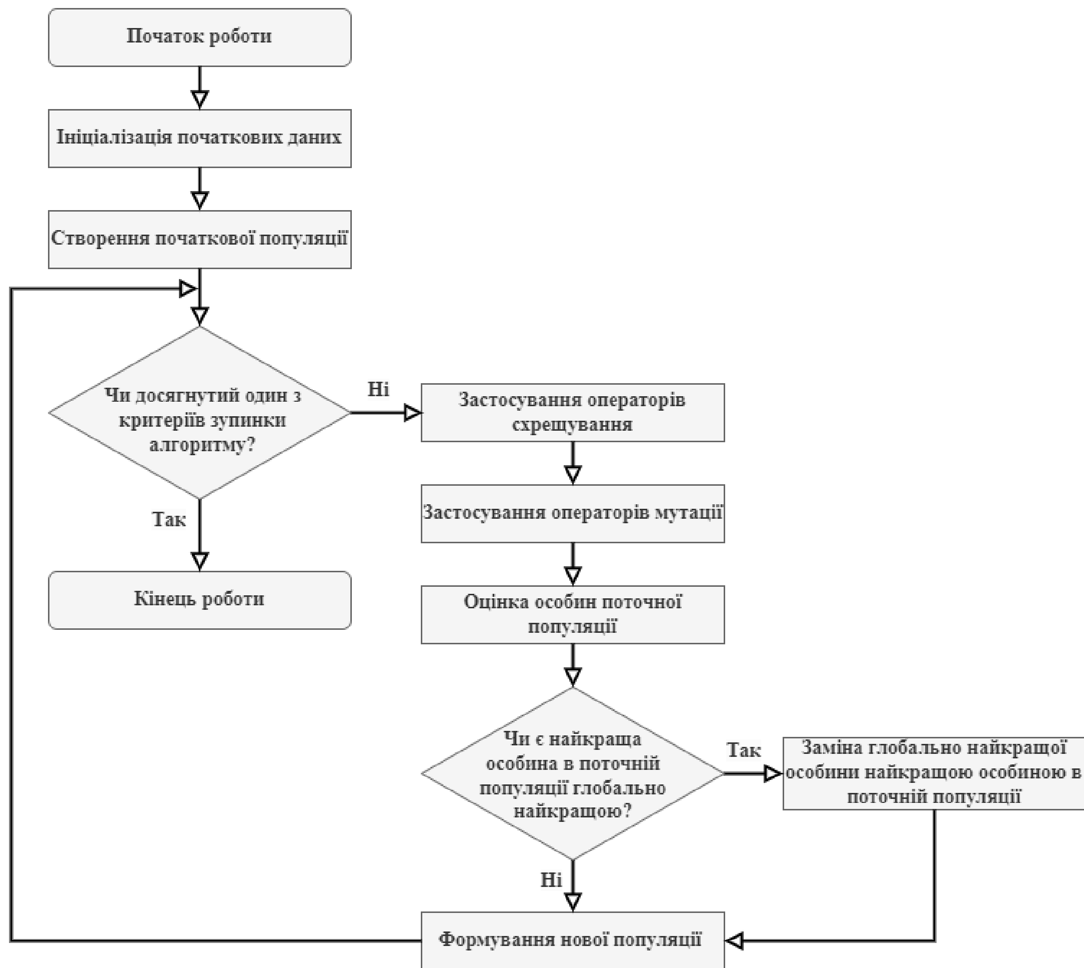

Рисунок 2.6 – Блок-схема ГА для рішення ЗМТ

Наступним етапом кожна з особин оцінюється за допомогою функції пристосованості (фітнес-функції), яка визначає, наскільки особина відповідає заданим критеріям оптимізації. Потрібно змінити глобально найкращу особину, якщо особина з найменшим значенням фітнес-функції в поточній популяції є кращою. Наступним кроком є формування нового покоління (множини варіантів рішень ЗМТ). Після цього відбувається перехід до перевірки досягнення критерію зупинки.

В якості критерію зупинки використовується один або деяка комбінація наступних:

- виконання алгоритмом заданого числа ітерацій;
- виконання алгоритмом заданого числа ітерацій без покращення поточного найкращого рішення;

- досягнення рішення із значенням фітнес-функції, що краще або дорівнює заданому значенню.

Для оцінки особин використовується фітнес-функція (2.3), яка враховує сумарну вартість переміщень, наповненість вантажем транспортних засобів, дотримання часових вікон та кількість маршрутів [22].

$$f = c_{заг.} C_{коеф.} + d_{віл.} D_{коеф.} + mM_{коеф.} + kK_{коеф.}, \qquad (2.3)$$

де $C_{коеф.}$, $D_{коеф.}$, $M_{коеф.}$, $K_{коеф.}$ – коефіцієнти, що впливають на значимість характеристик отриманого рішення;

$c_{заг.}$ – загальна вартість переміщень у рішенні;

$d_{віл.}$ – загальний обсяг вільного місця у всіх транспортних засобів;

m – кількість маршрутів у рішенні;

k – кількість вершин, для яких обслуговування відбувається за межами часового вікна.

Підбір коефіцієнтів фітнес-функції є важливим для правильної роботи алгоритму, необхідно враховувати початкові параметри та мету вирішення задачі (відносну важливість критеріїв оптимального рішення).

З метою прискорення роботи ГА та уникнення рішень, у яких кількість маршрутів більше заданого значення, використовується модифікований варіант фітнес-функції (2.3). Якщо кількість маршрутів не задовольняє заданим умовам, то функція пристосованості не проводить оцінку рішення, а повертає достатньо велике значення, яке свідчить що знайдений розв’язок є одним з найвіддаленіших від найкращого.

2.5.1 Створення початкової популяції

Генерація початкового набору рішень є важливим етапом у ГА, оскільки від цього залежить, наскільки ефективним буде процес пошуку

найкращого рішення. Одним з найпростіших способів сформувати початкові рішення є створення набору випадково згенерованих особин [21]. Однак, такий підхід недоцільно використовувати при вирішенні ЗМТ, через її складну структуру та велику кількість параметрів, а також наявність певних обмежень та умов. Таким чином, для створення множини початкових особин є раціональним використання класичних алгоритмів таких як жадібний алгоритм та модифікація на його основі, алгоритм Кларка-Райта, а також метод гілок з відсіканням.

Для етапу створення початкової популяції особин реалізована можливість створити перше рішення за допомогою класичного алгоритму, а інші – клонувати з отриманого. Це значно підвищує швидкодію при гібридизації з алгоритмами, що генерують одне й те ж саме рішення на однаковому наборі даних. Кількість особин у початковій множині рішень задається як параметр алгоритму.

2.5.2 Стратегії вибору батьківських особин

Процес схрещування в ГА потребує участі двох батьківських особин, які повинні бути обрані певним способом (стратегією відбору). В результаті схрещування нащадок наслідує генну інформацію (частину маршрутів або підмаршрутів) від обох обраних батьків. Таким чином, стратегія вибору є важливим компонентом ГА, що впливає на процес знаходження найкращого рішення. В роботі реалізовано декілька стратегій для вибору батьківських особин – турнірний відбір, рулеточний, панміксія, інбридинг.

Турнірний відбір представляє собою вибір найкращого рішення з заданого числа випадкових особин. Таким чином обирається необхідна кількість батьківських особин (яка залежить від налаштувань алгоритму). Перевагою даної стратегії є відсутність додаткових обчислень [17].

При рулеточному відборі кожна особина має ймовірність бути обраною. Ймовірність залежить від значення функції пристосованості

рішення. Чим більше значення фітнес-функції, тим більше ймовірність. При цьому, одна особина може бути обраною декілька разів [24].

Панміксія є однією з найпростіших стратегій відбору, але при цьому універсальною для вирішення задач різного класу. Кожній особині з популяції ставиться у відповідність інша випадково обрана особина. В підсумку сформується набір пар рішень, який буде використовуватись для процесу схрещування [25].

Останнім реалізованим способом відбору є інбридинг [26], коли першого батька обирають випадковим чином, а другий батько є членом популяції, що має найближче значення функції пристосованості до значення першої особини.

2.5.3 Оператори кросинговеру

Оператори кросинговеру виконують процес схрещування відібраних батьківських особин з метою наповнення поточної популяції новими рішеннями. В роботі розширено процес розмножування та додано можливість застосовувати декілька операторів кросинговеру одночасно, що призводить до обстеження більшої області можливих рішень [27]. Для кожної сформованої пари батьківських рішень із заданою ймовірністю застосовується кожен з обраних операторів кросинговеру. При цьому можна обрати однакові типи кросинговеру, але встановити їм різні параметри.

Особливостями операторів схрещування в даному алгоритмі є їх адаптація до умов задачі та генерація лише одного нащадка. Кожен з операторів на початку роботи перетворює батьківські рішення в послідовності вершин (чисел) для їх сумісності із класичними операторами кросинговеру. В кінці процесу схрещування із ланцюжку чисел формується рішення, яке представляється набором маршрутів.

Одним з використовуваних операторів є OX-кросинговер [20]. Спочатку обирається точка схрещування, а потім вершини першого батька

розміщені до цієї точки копіюються в нащадка без змін. Вершини, які знаходяться після точки, також поміщаються в нащадка, але в тому порядку, в якому ці вершини зустрічаються в другій особині.

Наступним оператором, реалізованим в роботі, є CX-кросинговер (рисунок 2.7), що заснований на визначенні циклічних вершин у першій особині та перенесенні їх у нащадка [23]. Інші вершини переносяться з другої особини, зберігаючи своє положення.

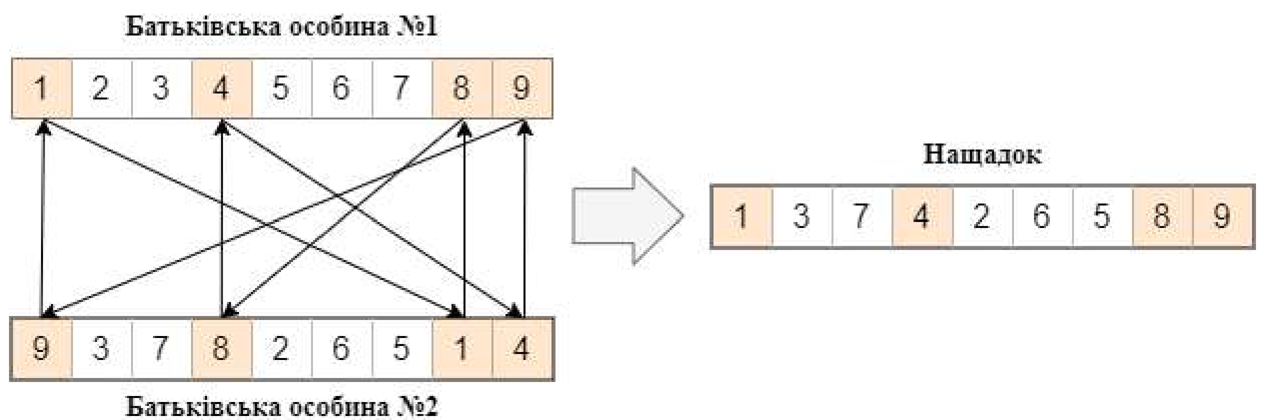

Рисунок 2.7 – Приклад роботи CX-кросинговеру

Визначення циклу починається з випадкового вибору та фіксації вершини першого рішення. Далі знаходиться позиція цієї вершини у другій особині та використовується для вибору наступної вершини у першому розв’язку. Ці дії виконуються, доки в першому рішенні не буде повторно обрано зафіксовану вершину. Всі вершини, що обирались в ході розглянутих дій, є циклічними. Використання цього оператору дозволяє створити нащадків, у яких кожна вершина має позицію від одного з батьків.

PMX-кросинговер [25] – метод схрещування, що створює нащадків шляхом вибору частини послідовності елементів від одного батька та зберігає порядок і положення якомога більшої кількості елементів від іншого батька (рисунок 2.8). Частина послідовності елементів обирається шляхом вибору двох випадкових точок розрізу (які є межами обраної послідовності).

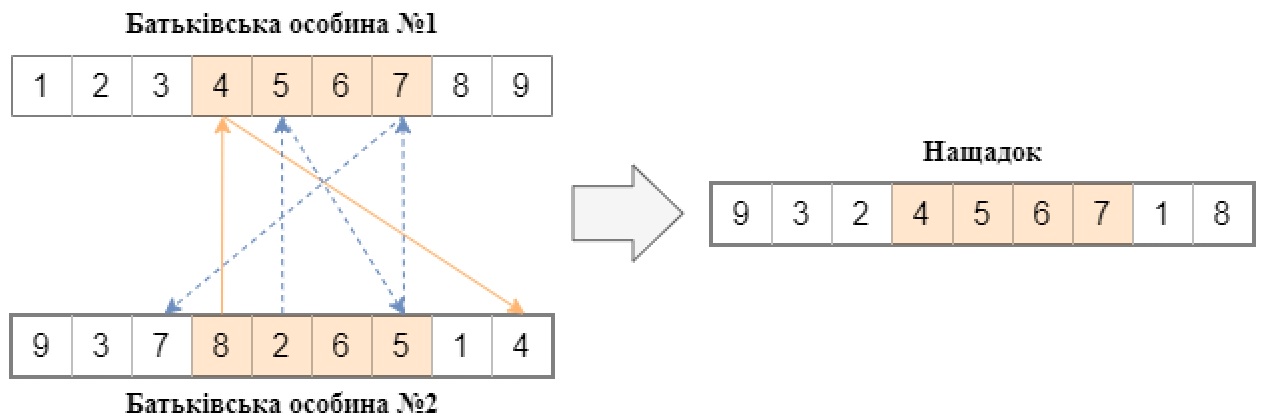

Рисунок 2.8 – Приклад роботи PMX-кросинговеру

Реалізований PMX-кросинговер має параметр, який дозволяє налаштовувати максимальну кількість вершин, що може бути перенесена з першої особини в нащадка. Мінімальна кількість не налаштовується та завжди дорівнює двом.

2.5.4 Оператори мутації

Для внесення різноманіття у популяцію, а також дослідження більшого діапазону можливих рішень реалізовано три види мутацій, що адаптовані під обраний спосіб кодування. Так же як і в процесі схрещування, є можливість застосовувати кілька операторів мутації з заданою ймовірністю до кожної особини у популяції. Зазвичай, в ГА, ймовірність спрацьовування кросинговеру встановлюється досить високою (більше 0.5), а ймовірність мутації досить малою (менше 0.1). Це призводить до того, що схрещування виконується майже завжди, а мутації – досить рідко. На відміну від розглянутих операторів кросинговеру, використовувані мутації не потребують перетворення рішення у єдину послідовність вершин.

Інверсна мутація (зміна порядку слідування вершин в маршруті) та мутація випадкової перестановки підмножини вершин у межах одного маршруту дозволяють проводити пошук кращого локального рішення, що

дозволяє зменшити витрати на переміщення між вузлами та досягти умови часових вікон [17].

Мутація обміну підмножин вершин (фрагментів маршрутів) між різними маршрутами у рамках однієї особини дозволяє розглядати рішення, що мають різну кількість маршрутів, покращувати наповнення транспортних засобів вантажем та шукати краще глобальне рішення з меншими витратами на переїзди [16].

2.6 Модифікований мурашиний алгоритм

2.6.1 Загальний опис

Мурашині алгоритми [18, 25] з'явилися в результаті аналізу поведінки реальної мурашиної колонії (рисунок 2.9). Мурахи знаходять найкоротший шлях від мурашника до джерела їжі за допомогою феромона (хімічної рідини із стійким запахом), залишеного іншими мурахами. Зустрівши шлях, що містить сліди феромонів, мураха з більшою ймовірністю піде по ньому. При цьому мураха розпилює свій феромон, що посилює існуючі сліди феромонів на шляху. В результаті шлях з найбільшою кількістю феромонів стає найкращим.

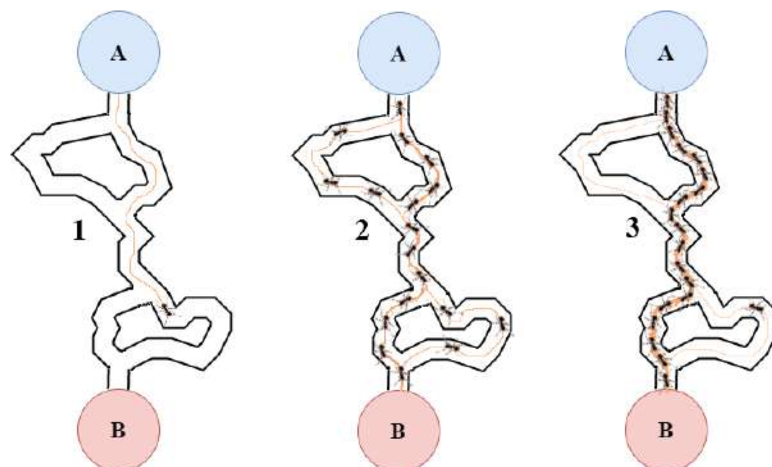

Рисунок 2.9 – Приклад роботи мурашиного алгоритму

Реалізований мурашиний алгоритм (рисунок 2.10) дозволяє знаходити рішення для ЗМТ з додатковими обмеженнями. Для цього достатньо врахувати обмеження вантажопідйомності при реалізації логіки мурах та умови часових вікон при оцінці знайдених рішень.

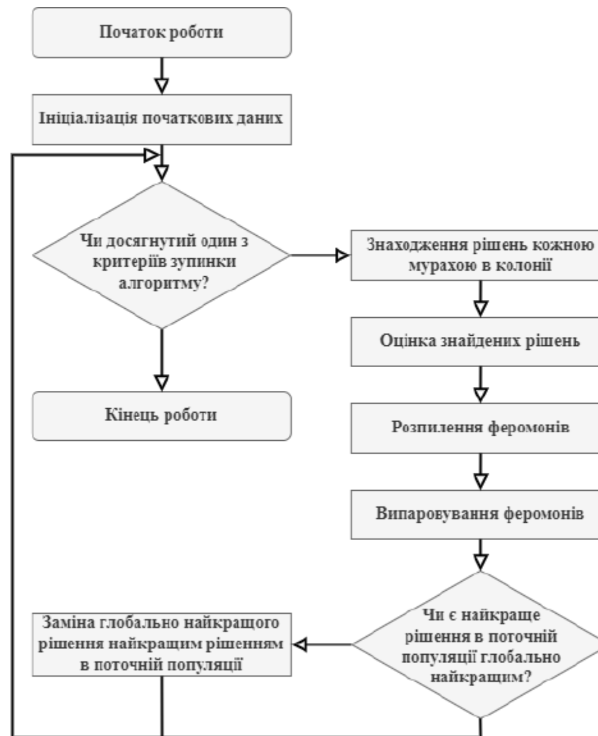

Рисунок 2.10 – Блок-схема мурашиного алгоритму для рішення ЗМТ

Критерії зупинки алгоритму, функція оцінки знайдених рішень та збереження глобально найкращого рішення (при умові, що знайшовся розв’язок кращий, ніж поточний) в розглянутому методі є ідентичними до тих, що використовуються в ГА.

Розпилення та випаровування феромонів відбувається після того як кожна мураха знайде допустимий розв’язок ЗМТ. Кількість феромона при розпиленні залежить від налаштувань мурах та значення фітнес-функції знайденого рішення. Кількість феромона, що випаровується, завжди однакова для всіх переходів між вершинами та задається як параметр алгоритму.

2.6.2 Мурахи

В розглянутому мурашиному алгоритмі мурахи використовуються для знаходження рішень на основі розрахованих ймовірностей переміщень, які залежать від кількості феромонів на шляху та вартості переміщення (2.4).

$$P_{ij} = \frac{\tau_{ij}^{\alpha} \eta_{ij}^{\beta}}{\sum_{k=1}^{n} \tau_{ik}^{\alpha} \eta_{ik}^{\beta} x_k}, \tag{2.4}$$

де P_{ij} – ймовірність переміщення з пункту i в пункт j;

τ – кількість феромонів на шляху;

η – величина зворотна вартості переміщення;

α, β – константі значення, що є параметрами мурахи;

x – бінарна змінна, що показує чи доступна вершина для переміщення;

n – загальна кількість вершин.

Алгоритм дозволяє використовувати декілька типів мурах (при цьому дозволяється обирати однакові типи) з різною конфігурацією параметрів. Кожен тип мурахи має свої особливості в знаходженні рішень, а деякі з них мають додаткові параметри [17].

Стандартна мураха завжди розглядає для переміщення всі доступні вершини, а вибір відбувається з урахуванням ймовірностей переходу. Елітна мураха завжди обирає вершину для переміщення з найбільшою ймовірністю (якщо таких вершин декілька, то обирає одну з них випадковим чином). Лінива мураха при виборі пункту для переміщення розглядає тільки частину рішень, задану при налаштуванні параметрів мурахи [17, 27].

Рангова мураха розпиляє феромон, кількість якого залежить від рангу рішення, яке вона знайшла. При цьому, якщо ранг знайденого рішення буде перевищувати максимальний (заданий як параметр мурахи), то ця мураха не буде розпилювати феромон на поточній ітерації [19, 26].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Загальна структура проекту

Для дослідження та тестування розроблених гібридних алгоритмів вирішення ЗМТ реалізовано програмну систему з використанням технології WPF та патерну MVVM (рисунок 3.1). Проект складається з трьох частин: View, ViewModel, Model. View визначає інтерфейс, через який користувач взаємодіє з програмою. ViewModel пов’язує частини Model та View через механізм прив’язки даних. Model описує дані, що використовуються в застосунку, а також логіку пов’язану з цими даними.

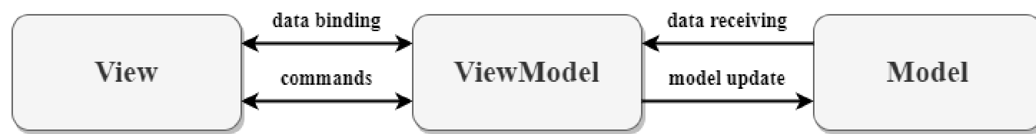

Рисунок 3.1 – Структура MVVM

Реалізація всіх обраних алгоритмів, а також структур даних, необхідних для роботи з ними, знаходяться в частині Model. Враховуючи це, далі, в більшості випадків буде розглядатися саме ця складова проекту, опускаючи деталі роботи інших компонентів.

Вихідний код моделі має деревоподібну структуру (рисунок 3.2). Каталог Algorithms містить в собі все, що пов’язано з реалізованими алгоритмами, а також деякі допоміжні класи, що дозволяють покращити структуризацію коду. У каталозі Base, що знаходиться в папці Model, розміщені базові класи для інших розділів моделі. Вся логіка та необхідні структури даних для відображення результатів роботи алгоритмів у вигляді графіків та стовпчастих діаграм знаходяться у папці Charts. Розділ Map зберігає класи, що дозволяють описувати ЗМТ або допомагають з ними

працювати. Код в каталозі PerformanceTests відповідає за логіку запуску набору алгоритмів для вирішення набору ЗМТ, з подальшим збереженням результатів роботи. Папка Validators містить в собі файли з кодом, що відповідає за перевірку коректності налаштованих параметрів ЗМТ або алгоритмів. Головним класом, з якого починається робота всієї моделі, є AppModel.

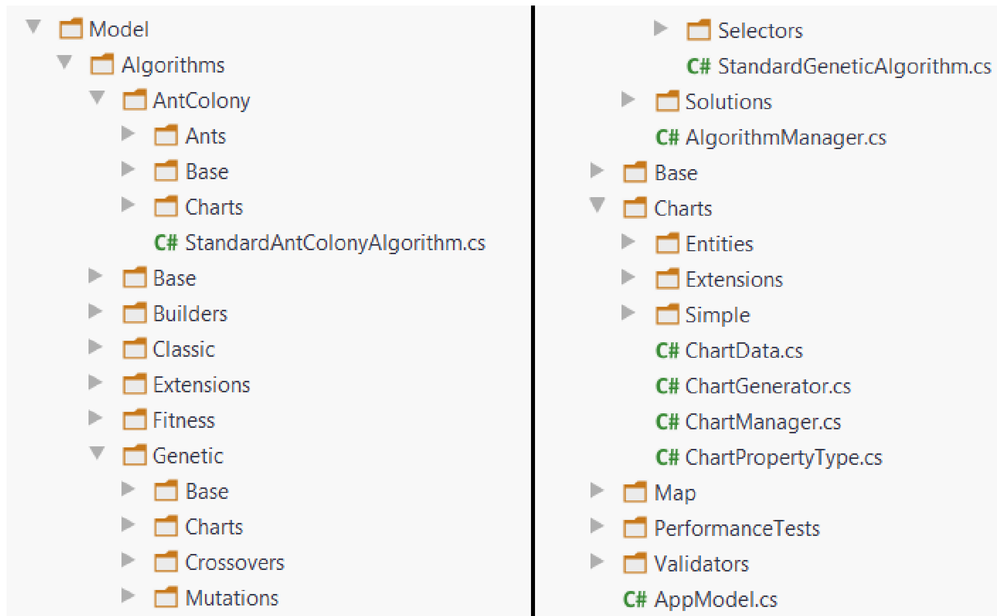

Рисунок 3.2 – Структура model

Клас AppModel містить в собі чотири поля для зручності роботи з різними розділами моделі. Ці поля відповідають за роботу з ЗМТ (з формуванням графіків та діаграм), алгоритмами та їх тестування. Кожне поле має тип, що реалізує інтерфейс IManager з методами Init та Close, які викликаються при запуску та виключенні застосунку відповідно. Зазвичай, якщо дія стосується тільки одного розділу, то викликається метод класу, що відповідає за потрібну частину. У випадку, коли потрібна взаємодія двох або більше класів, що відповідають за різні складові моделі, викликаються методи з класу AppModel, в яких реалізована необхідна логіка по обміну даними між задіяними полями.

3.2 Опис базових класів

Для опису ЗМТ використовується клас VrpMap (лістинг 3.1), що містить поля помічені атрибутом JsonProperty (який дозволяє задати ім’я поля для серіалізації) та деякі допоміжні властивості, що не серіалізуються. Weight та Time представляють собою матриці вартостей та часу переміщень між різними клієнтами. MaxTruckCapacity є максимальною вантажопідйомністю кожного транспортного засобу, а MaxTruckCount обмежує їх кількість. RandomWeight показує як сформовані значення вартостей переїздів – випадково чи розрахунком відстані між двома точками на координатній площині.

Лістинг 3.1 – Клас VrpMap

```
public class VrpMap
{
    [JsonProperty("q")] public List<Node> Nodes;
    [JsonProperty("w")] public double[][] Weight;
    [JsonProperty("y")] public double[][] Time;
    [JsonProperty("e")] public int MaxTruckCapacity;
    [JsonProperty("r")] public int MaxTruckCount;
    [JsonProperty("t")] public bool RandomWeight;
    [JsonIgnore] public IEnumerable<Node> NodesWithoutDepot =>
       Nodes.Where(x => !x.IsDepot);
}
```

Колекція Nodes відповідає за зберігання інформації про кожен пункт в задачі. Клас Node містить наступні поля:

- ідентифікатор пункту (депо завжди має значення 0);
- тип пункту (депо чи клієнт);
- потреба клієнта (депо має пусте значення);
- часове вікно (окремий клас з полями MinTime та MaxTime);
- координати x та y (для відображення на мапі в програмі).

Для налаштувань параметрів генерації ЗМТ використовується клас MapSettings, що дозволяє обрати межі значень для матриць вартостей та часу,

кількість клієнтів та вантажопідйомність транспортних засобів, мінімальні та максимальні значення для потреб та часових вікон споживачів. VrpMap формується класом MapBuilder, який потребує налаштованого MapSettings.

Розв’язок ЗМТ описується класом Solution, який реалізує інтерфейс ISolution, що містить властивість FitnessValue (значення фітнес-функції) та метод Clone (копіювання рішення). Даний клас має список маршрутів, кожен з яких представлений класом Route. В кожному маршруті є поля NodeIds та DeliveredResources, що зберігають колекцію ідентифікаторів пунктів та кількість потреб клієнтів відповідно.

Для застосування алгоритмів в розробленій програмній системі, кожен з них повинен реалізовувати інтерфейс IAlgorithm (лістинг 3.2). Властивість Type представляє конкретний тип алгоритму. Метод GetSolution запускає процес знаходження рішення алгоритмом та повертає результат цього пошуку. SetMap встановлює конкретну ЗМТ, яку буде вирішувати алгоритм. Reset дозволяє скинути всі кешовані дані використовуваного методу. WithCollectStats вмикає збір детальної інформації про роботу алгоритму. UML діаграми із всіма класами, що пов'язані з розробленими алгоритмами, наведені у додатку Б.

Лістинг 3.2 – Інтерфейс IAlgorithm

```
public interface IAlgorithm<in TMap, out TSolution>
    where TMap : class where TSolution : ISolution<TSolution>{
    AlgorithmType Type { get; }
    TSolution GetSolution();
    IAlgorithm<TMap, TSolution> SetMap(TMap map);
    IAlgorithm<TMap, TSolution> Reset();
    IAlgorithm<TMap, TSolution> WithCollectStats();
}
```

Для налаштування параметрів алгоритму використовується клас AlgorithmInfo, що містить інформацію про тип, назву та специфічні параметри алгоритму. Такі параметри задаються в класах ClassicInfo, AntColonyInfo та GeneticInfo. Для кожного алгоритму може бути

ініціалізований тільки один з перерахованих класів. У випадку ГА для налаштування алгоритму створення початкової популяції всередині GeneticInfo є окреме поле.

Класи для оцінки розв'язків, знайдених алгоритмами, повинні реалізовувати інтерфейс IFitness, головним методом якого є EvalFor. Оскільки деякі варіанти фітнес-функції можуть змінювати лише частину стандартної логіки розрахунку, було додано абстрактний клас BaseFitness (лістинг 3.3), що наслідують всі реалізовані функції пристосованості.

Лістинг 3.3 – Абстрактний клас BaseFitness

```
public abstract class BaseFitness : IFitness<VrpMap, Solution>{
    public abstract FitnessType Type { get; }
    protected VrpMap Map;
    protected readonly double WeightModifier;
    protected readonly double CountModifier;
    protected readonly double ResourcesModifier;
    protected readonly double TimeModifier;
    protected BaseFitness(double weightMod = 1, double countMod
= 1, double resourcesMod = 1, double timeMod = 1) =>
        (WeightModifier, CountModifier, ResourcesModifier,
TimeModifier) = (weightMod, countMod, resourcesMod, timeMod);
    public IFitness<VrpMap, Solution> SetMap(VrpMap map) { … }
    public virtual IFitness<VrpMap, Solution> Reset(){ … }
    public virtual double EvalFor(Solution solution) { … }
    protected double CalcStandardFitness(double weight, int
count, int resources, double time) { … }
    protected (double, int, int)
GetWeightAndResourcesFrom(Solution solution){ … } }
```

Розглянутий клас дозволяє перевизначити кожен з етапів розрахунку (методи EvalFor, CalcStandardFitness, GetWeightAndResourcesFrom), а також уникнення необхідності реалізовувати в класах-нащадках логіку для SetMap та Reset, якщо достатньо стандартної поведінки. Поля WeightModifier, CountModifier, ResourcesModifier, TimeModifier є коефіцієнтами функції пристосованості та застосовуються в методі CalcStandardFitness. Щоб розрахувати необхідні значення, що відносяться до конкретного рішення, застосовується метод GetWeightAndResourcesFrom.

3.3 Реалізація алгоритмів

3.3.1 Жадібний алгоритм

Код жадібного алгоритму для вирішення ЗМТ знаходиться у класі GreedyAlgorithm. Алгоритм працює в циклі доки є доступні клієнти. Важливою частиною роботи алгоритму на кожній ітерації є відбір потенційних клієнтів для подальшого відвідування (лістинг 3.4). При цьому використовується приватне поле _selectionCount, яке визначає кількість клієнтів, з якої буде обиратися наступний для відвідування.

Лістинг 3.4 – Фрагмент коду жадібного алгоритму

```
Node[] nextNodes = null;
var nextNodesWithTimeWindows = availableConsumers
    .Where(x => IsResourcesEnough(x, availableCapacity))
    .Where(x => IsInTimeWindow(x.TimeWindow, routeTime +
Map.TimeMatrix[prevNodeId][x.Id]))
    .OrderBy(x => x.TimeWindow.MaxTime)
    .ThenBy(x => Map.Weight[prevNodeId][x.Id])
    .Take(_selectionCount).ToArray();
if (!nextNodesWithTimeWindows.IsNullOrEmpty())
    nextNodes = nextNodesWithTimeWindows;
if (nextNodes == null){
    var nextNodesWithoutTimeWindows = availableConsumers
        .Where(x => x.TimeWindow == null)
        .Where(x => IsResourcesEnough(x, availableCapacity))
        .OrderBy(x => Map.Weight[prevNodeId][x.Id])
        .Take(_selectionCount).ToArray();
    if (!nextNodesWithoutTimeWindows.IsNullOrEmpty())
        nextNodes = nextNodesWithoutTimeWindows; }
```

Спочатку обираються споживачі, які задовольняють умовам вантажопідйомності та часових вікон, після чого зберігаються у змінну nextNodesWithTimeWindows. Метод IsResourceEnough здійснює перевірку місткості транспортних засобів, а IsInTimeWindow – чи буде обслуговуватися клієнт в межах часового вікна. Наступним етапом відбору є перевірка, чи є відібрані клієнти на першому етапі. У випадку коли немає, відбираються

клієнти за таким же принципом, але замість виклику IsInTimeWindow здійснюється перевірка на відсутність часового вікна у клієнта (в полі TimeWindow має бути значення null). Останнім етапом, якщо на перших двох не вдалось знайти підходящих споживачів, є підбір клієнтів без врахування часових вікон, але не порушуючи обмеження вантажопідйомності.

Якщо потенційних клієнтів не виявилось, поточний маршрут завершується (додається пункт депо в кінець маршруту). В іншому випадку із списку nextNodes обирається споживач випадковим чином та вноситься в поточний маршрут. Після закінчення доступних клієнтів знайдене рішення повертається як результат методу GetSolution.

3.3.2 Модифікація жадібного алгоритму

Клас MaxPackerAlgorithm реалізує логіку модифікованого жадібного алгоритму, заснованого на максимізації завантаження транспорту. В методі GetSolution (лістинг 3.5) виконується розрахунок суми потреб всіх клієнтів, оцінка мінімальної кількості транспортних засобів (необхідної для того, щоб вмістити всі потреби споживачів) та розподіл клієнтів двома способами. Ці дії виконуються в циклі, доки є нерозподілені клієнти.

Лістинг 3.5 – Метод GetSolution модифікованого жадібного алгоритму

```
public override Solution GetSolution(){
    var totalRes = Map.NodesWithoutDepot.Sum(x =>
x.RequiredResources!.Value);
    var count = EstimateCount(totalRes);
    var finalRoutes = new List<Route>(count);
    var notPacked = PackNodesTo(finalRoutes, count, Map.Nodes);
    while (notPacked.Count > 0){
        totalRes = notPacked.Select(x => Map.Nodes[x]).Sum(x =>
x.RequiredResources!.Value);
        count = EstimateCount(totalRes);
        notPacked = PackNodesTo(finalRoutes, count,
notPacked.Select(x => Map.Nodes[x])); }
    foreach (var route in finalRoutes) route.Finish();
    return new Solution(finalRoutes);}
```

В методі PackNodesTo відбувається спочатку розподіл першим способом. Для цього здійснюється ітеративне проходження по нерозподіленим клієнтам у порядку спадання. Кожен клієнт розподіляється за допомогою методу MakeFirstTryToPackNode або в поточний маршрут, або в перший доступний (за умовою вантажопідйомності). Якщо доступних маршрутів немає, то споживач вноситься в список notPackedOnFirstTry. Після цього, якщо є нерозподілені клієнти після першої спроби, відбувається друга спроба розподілу методом MakeSecondTryToPackNode.

Другий спосіб розподілу здійснює перебір всіх пар вже сформованих маршрутів і для кожної пари маршрутів перебір всіх пар клієнтів. Кожна пара споживачів перевіряється на можливість звільнення місця для поточного клієнта (для якого виконується розподіл) за рахунок їх перестановки. Перед розглядом кожної пари маршрутів здійснюється перевірка, чи можуть дані маршрути потенційно звільнити місце, що значно зменшує вартість обчислень. Якщо після другого розподілу залишились нерозподілені клієнти, вони розміщуються в колекцію notPacked та будуть розподілятись на наступних ітераціях алгоритму.

Після закінчення розподілу всіх клієнтів по маршрутам здійснюється завершення кожного маршруту за допомогою методу Finish в класі Route та алгоритм повертає сформований розв’язок.

3.3.3 Метод Кларка-Райта

Реалізація методу Кларка-Райта представлена класом SavingsAlgorithm. На початку роботи алгоритму в методі GetSolution (лістинг 3.6) створюється список маршрутів за допомогою функції BuildInitialRoute та деяких стандартних методів LINQ. Кожен сформований маршрут містить одного клієнта та є завершеним (початковим та кінцевим пунктами маршруту є депо). Наступним етапом є розрахунок значень збережень за допомогою методу GetSavings та їх сортування функцією OrderByDescending.

Лістинг 3.6 – Метод GetSolution алгоритму Кларка-Райта

```
public override Solution GetSolution(){
    var routeByNodeId = Map.NodesWithoutDepot
        .Select(BuildInitialRoute)
        .ToDictionary(x => x.NodeIds.First(y => y !=
Map.DepotId), x => x);
    foreach (var savingItem in GetSavings().OrderByDescending(x
=> x.Saving)){
        var routeI = routeByNodeId[savingItem.NodeI];
        var routeJ = routeByNodeId[savingItem.NodeJ];
        if (!(SatisfySecondCondition(savingItem, routeI, routeJ)
    && SatisfyThirdCondition(routeI, routeJ)
    && SatisfyFirstCondition(savingItem, routeI))) continue;
        routeI.Merge(routeJ);
        foreach (var nodeId in routeJ.NodeIds)
            routeByNodeId[nodeId] = routeI;}
 return new Solution(routeByNodeId.Values.Distinct().ToList());}
```

Метод GetSavings повертає список з елементів, які описуються структурою SavingItem. Ця структура має поля NodeI та NodeJ, які визначають пов’язані маршрути з даним значенням збереження (поле Saving).

В ході розгляду кожного елементу SavingItem, для злиття двох маршрутів необхідно, щоб виконувались три умови:

- маршрут routeI не повинен містити споживача NodeJ (реалізується методом SatisfyFirstCondition);
- першим або останнім пунктами в маршрутах (без урахування депо) повинні бути клієнти з ідентифікаторами NodeI або NodeJ (здійснюється функцєю SatisfySecondCondition);
- сума потреб клієнтів обох маршрутів не повинна перевищувати вантажопідйомність транспортних засобів (виконується методом SatisfyThirdCondition).

Злиття відбувається за допомогою функції Merge класу Route. Після злиття в колекції routeByNodeId оновлюються маршрути для всіх клієнтів, задіяних в сформованому маршруті. Після розгляду всіх збережень, над колекцією routeByNodeId здійснюється операція Distinct, яка видаляє дублюючі маршрути із кінцевого списку.

3.3.4 Метод гілок з відсіканням

Метод гілок з відсіканням описується класом BranchAndCutAlgorithm. В методі GetSolution викликаються дві функції, спочатку GetRequiredTransitions (лістинг 3.7), а потім MakeSolution. Перший метод відповідає за формування списку обов'язкових переміщень, а другий за створення розв'язку на основі цього списку.

Лістинг 3.7 – Фрагмент функції GetRequiredTransitions

```
var requiredTransitions = new Dictionary<int, int>();
var (zeroI, zeroJ) = SelectZeroElement(leftMatrix,
SolveMatrix(leftMatrix)); var isLeft = true;
while (_availableIs.Count > 1){
    if (isLeft)CopyMatrix(rightMatrix, leftMatrix);
    else CopyMatrix(leftMatrix, rightMatrix);
    leftMatrix[zeroJ][zeroI] = double.PositiveInfinity;
    var leftResult = SolveMatrix(leftMatrix, zeroI, zeroJ);
    rightMatrix[zeroI][zeroJ] = double.PositiveInfinity;
    var rightResult = SolveMatrix(rightMatrix);
    isLeft = leftResult.Sum <= rightResult.Sum;
    if (isLeft){
       requiredTransitions.Add(zeroI, zeroJ);
       _availableIs.Remove(zeroI);
       _availableJs.Remove(zeroJ);
       (zeroI, zeroJ) = SelectZeroElement(leftMatrix,
leftResult, zeroI, zeroJ);}
    else
 (zeroI, zeroJ) = SelectZeroElement(rightMatrix, rightResult);
}
```

В методі GetRequiredTransitions на початку ініціалізуються матриці leftMatrix та rightMatrix даними з матриці вартостей переміщень. Потім для першої ітерації обирається ліва матриця, здійснюється її вирішення та вибір нульового елемента (переміщення, що має найменшу вартість). Далі всі дії виконуються в циклі, доки не закінчаться клієнти без обов'язкових переміщень. Метод CopyMatrix копіює одну матрицю в іншу.

Функція SolveMatrix спочатку ітеративно проходиться по рядкам матриці, знаходить мінімальні елементи та віднімає їх значення від інших

значень на кожному рядку. Потім такі ж дії виконуються з кожним стовпцем матриці. Розглянутий метод повертає структуру MatrixResult, яка містить мінімальні елементи (індекси матриці та значення), а також їх загальну суму.

Після вирішення кожного з варіантів матриць, якщо сума мінімальних елементів лівої матриці менше або дорівнює сумі правої матриці, то переміщення, аналізоване на поточній ітерації, вноситься в список обов’язкових, а наступний нульовий елемент обирається з лівої матриці. В іншому випадку, аналізоване переміщення вважається забороненим та вибір наступного нульового елементу здійснюється з правої матриці.

Функція MakeSolution формує маршрути з урахуванням вантажопідйомності транспортних засобів на основі списку обов’язкових переміщень. Формування маршрутів починається з першого пункту в списку, а кожна наступна вершина визначається обов’язковим переміщенням для поточної вершини. Якщо на одному з етапів місткості транспортного засобу недостатньо, поточний маршрут завершується та створюється новий.

3.3.5 Генетичний алгоритм

ГА реалізований за допомогою класу StandardGeneticAlgorithm, який наслідує клас BaseGeneticAlgorithm (що містить в собі стандартну логіку ГА, яку можна перевизначити в класах-нащадках). Базовий клас містить в собі набори компонентів для виконання окремих етапів алгоритму (наприклад, компоненти для здійснення мутацій або схрещування), а також екземпляр класу GeneticAlgorithmState (який описує поточну популяцію та глобально найкраще рішення).

В методі GetSolution (лістинг 3.8) поле CollectStats визначає, яку версію алгоритму запускати (зі збором детальної інформації чи без). У випадку роботи алгоритму в режимі без збору інформації, спочатку виконується ініціалізація всіх компонентів та створення початкової популяції за допомогою обраного класичного способу.

Лістинг 3.8 – Функція GetSolution генетичного алгоритму

```
public override TSolution GetSolution(){
    if (CollectStats) return WCS_GetSolution();
    Init();
    while (!State.IsEnd){
        MakeNewPopulation();
        MakeCrossover();
        MakeMutation();
        MakeFitness();
        MakeSorting();
        ChangeBestSolutionIfNeeded(); }
    return State.BestSolution; }
```

Генерація початкового набору розв’язків може виконуватись двома способами в залежності від параметру CloneInitialSolutions. Коли параметр має значення false, всі рішення генеруються обраним класичним алгоритмом, інакше, перше рішення генерується алгоритмом, а інші – клонуються з першого.

Метод MakeNewPopulation формує нову популяцію за допомогою компоненту PopulationSelector. Відбір батьківських особин здійснюється компонентом ParentsSelector. Ці компоненти повинні реалізовувати інтерфейс ISelector. В роботі розроблено наступні класи, що реалізують даний інтерфейс:

- клас TakeFirstSelector – обирає задану кількість перших особин;
- клас TournamentSelector працює за принципом турнірного відбору;
- клас RouletteSelector, що представляє собою рулеточний відбір;
- клас PanMixSelector – реалізує панміксію;
- клас InBreedingSelector працює на основі інбридингу.

На етапі схрещування (виклик функції MakeCrossover) до відібраних особин застосовується кожен з наявних компонентів кросинговеру з деякою заданою ймовірністю. Кожен такий компонент реалізує інтерфейс ICrossover, що містить метод Cross для схрещування двох особин. Так як кросинговери, реалізовані в даній роботі, мають спільну логіку, для зручності додано клас BaseCrossover, який наслідує кожен кросинговер.

SinglePointCrossover – клас, в якому описаний OX-кросинговер. Спочатку батьківські особини перетворюються в послідовності вершин за допомогою методу ParentAsArray. Потім генерується випадкова точка, яка ділить послідовність на дві частини. Перша частина з першої особини переноситься в нащадка без змін, а друга в тому порядку, в якому вершини знаходяться у другій особині. Ці дії відбуваються в методі MakeChild.

PartiallyMappedCrossover (лістинг 3.9) – клас, що реалізує PMX-кросинговер. Після перетворення батьків у послідовності вершин, генеруються змінні startIndex та amount, що визначають підмножину вершин, яка буде переноситись з першого батька в нащадка без змін. Потім формуються індекси (позиції) для вершин з другого батька. Для цього використовується функція PartiallyMappedIndexes. В кінці, використовуючи отримані індекси метод MakeChild генерує нащадка.

Лістинг 3.9 – Реалізація класу PartiallyMappedCrossover

```
public class PartiallyMappedCrossover : BaseCrossover {
    …
    public PartiallyMappedCrossover(CrossoverInfo info) :
base(info) =>
    _maxSubRouteNodes = info.MaxSubRouteNodes!.Value;
    public override Solution Cross(Solution first, Solution
second){ var firstNodes = ParentAsArray(first);
        var secondNodes = ParentAsArray(second);
        var startIndex = ConcurrentRandom.Next(0,
firstNodes.Length - MinNodesCount);
        var amount = ConcurrentRandom.Next(MinNodesCount,
Math.Min(secondNodes.Length - startIndex, _maxSubRouteNodes));
        var indexes = PartiallyMappedIndexes(firstNodes,
secondNodes, startIndex, amount);
        return MakeChild(second.Routes.Count, () =>
ChildNodes(secondNodes, indexes)); } }
```

CycleCrossover – клас, що реалізує CX-кросинговер. Даний оператор має реалізацію схожу на PMX-кросинговер, але без вибору підмножини вершин та замість функції PartiallyMappedIndexes використовується функція CycleIndexes, яка повертає індекси вершин виявленого циклу.

Етап проведення мутацій здійснюється методом MakeMutation. Всі наявні оператори мутації застосовуються для кожної особини в популяції з заданою ймовірністю. Оператори мутації повинні реалізовувати інтерфейс IMutation, який має функцію Mutate. Так же як і для кросинговерів, додано клас BaseMutation, що спрощує реалізацію кожної окремої мутації.

InverseMutation – клас з реалізацією інверсної мутації. Спочатку обирається випадковий маршрут за допомогою функції TryGetRandomRoute базового класу. Потім змінюється порядок проходження випадкової підмножини вершин. Зміна порядку відбувається при ітеративному проходженні половини обраної підмножини за допомогою допоміжного методу Swap.

ShiftInRouteMutation – клас, в якому описується мутація випадкової перестановки підмножини вершин у межах одного маршруту. Після вибору випадкового маршруту функцією TryGetRandomRoute генеруються локальні змінні startIndex, amount та shift. Потім відбувається зрушення обраної підмножини в циклі з використанням методу Swap. При цьому зрушення є циклічним.

Мутація обміну підмножин вершин між різними маршрутами реалізована в класі SwapBetweenRoutesMutation. Перший маршрут обирається функцією TryGetRandomRoute. Після цього в ньому обирається підмножина вершин та розраховуються потреби клієнтів, що туди потрапили. Далі методом TryGetSecondRoute отримується другий маршрут, в якому є підмножина вершин з потребами клієнтів, які можна розмістити в першому маршруті (щоб не порушувати обмеження вантажопідйомності). Якщо таких маршрутів нема, то мутація завершує свою роботу. Якщо такі маршрути є, то після вибору другого маршруту, у ньому обирається підмножина клієнтів та розраховуються їх потреби. Потім обрані фрагменти з обох маршрутів виносяться в локальні змінні та видаляються із своїх маршрутів. Після цього за допомогою функції InsertRange підмножини вершин вставляються в потрібні маршрути та перераховуються потреби клієнтів.

Після процесу мутації виконується оцінка поточних рішень в популяції (за допомогою компоненту, що реалізує інтерфейс IFitness) та їх сортування. Ці дії здійснюються методами MakeFitness та MakeSorting.

Логіка зміни глобально найкращого рішення реалізована в методі ChangeBestSolutionIfNeeded.

3.3.6 Мурашиний алгоритм

Мурашиний алгоритм реалізований в класі, що наслідує клас BaseAntColonyAlgorithm – StandardAntColonyAlgorithm. Знайдені рішення та феромони на поточній ітерації зберігаються в класі AntColonyState. Метод GetSolution (лістинг 3.10) схожий на метод в ГА, але має інші етапи. В методі Init ініціалізуються феромони та компоненти алгоритму (мурахи та фітнес-функція). Виклик функції Clear класу AntColonyState очищує поточний список рішень.

Лістинг 3.10 – Функція GetSolution мурашиного алгоритму

```
public override Solution GetSolution() {
    if (CollectStats) return WCS_GetSolution();
    Init();
    while (!State.IsEnd) {
        State.Clear();
        MakeAntsSimulation();
        MakeFitness();
        MakeDispersion();
        MakeEvaporation();
        ChangeBestSolutionIfNeeded();
    }
    return State.BestSolution;
}
```

Метод MakeAntsSimulation виконує симуляцію знаходження рішень кожною наявною мурахою в поточному налаштуванні алгоритму. Клас кожного типу мурах повинен реалізовувати інтерфейс IAnt, що містить метод GetSolution та DispersePheromone. Для зручності реалізації кожного типу

мурах та уникнення дублювання коду було додано клас BaseAnt із стандартною логікою мурахи. В класах-нащадках деякі функції базового класу можна перевизначити.

StandardAnt – клас, що представляє поведінку стандартної мурахи (вся логіка розміщена в BaseAnt). Основними функціями, що реалізують стандартну логіку для знаходження розв’язку, є GetNextNode (яка описує вибір наступної вершини для відвідування) та CalcNextNodeProb (яка виконує розрахунок ймовірності переходу в задану вершину). Для розрахунку кількості феромона, що буде розпилений мурахою на шляху, використовується метод CalculatePheromone.

Клас EliteAnt (лістинг 3.11) описує поведінку елітної мурахи, яка завжди обирає вершину з найбільшою ймовірністю переходу в неї. Для досягнення такої поведінки достатньо перевизначити метод GetNextNode з базового класу. Версія цього методу для елітної мурахи проходиться по всім доступним вершинам та запам’ятовує пункт та найбільшу ймовірність переходу в змінні bestNode та bestProb. Після чого повертає змінну bestNode. Інша логіка нічим не відрізняється від стандартної.

Лістинг 3.11 – Реалізація класу EliteAnt

```
public class EliteAnt : BaseAnt {
    public override AntType Type => AntType.Elite;
    public EliteAnt(AntInfo info) : base(info){ }
    protected override Node GetNextNode(List<Node> nodes, int 
prevNodeId, int capacity) {
        Node bestNode = null!;
        var bestProb = double.MinValue;
        foreach (var node in nodes.Where(x => 
x.RequiredResources <= capacity)){
            var prob = 
CalcNextNodeProb(State.Pheromones[prevNodeId][node.Id], 
Map.Weight[prevNodeId][node.Id]);
            if (prob <= bestProb) continue;
            bestNode = node; bestProb = prob;
        }
        return bestNode;
    }
}
```

Лінива мураха представлена класом LazyAnt. В своїй реалізації даний клас перевизначає метод GetNextNode з метою розгляду тільки заданого числа споживачів при виборі наступного споживача.

RankingAnt – клас, який реалізує логіку рангової мурахи. Даний клас перевизначає метод CalculatePheromone, щоб враховувати при розрахунку феромона ранг знайденого рішення. Ранг для рішень проставляється в методі MakeFitness у випадку коли в налаштуванні алгоритму є як мінімум одна рангова мураха.

Після симуляції роботи мурах виконується оцінка знайдених рішень за допомогою компоненту, що реалізує IFitness. Відбувається розпилення феромона кожною мурахою та випаровування деякого числа феромонів на кожному переміщенні з одного пункту в інший. Змін у логіці функції ChangeBestSolutionIfNeeded порівняно з ГА немає.

3.4 Тестування алгоритмів

Для тестування та порівняння алгоритмів розроблений клас PerformanceManager та PerformanceTester. Перший клас відповідає за зберігання налаштованих наборів алгоритмів та ЗМТ, а також за ініціацію запуску тестування наборів. Другий клас інкапсулює в собі логіку проведення тестів та формування результатів тестування.

Набори алгоритмів та ЗМТ описуються класом TestSetup, який зберігає інформацію про шляхи до файлів або налаштування параметрів ЗМТ, обрані алгоритми та функції пристосованості. Результати запуску тестів зберігаються в класі TestResultsData, що містить детальну інформацію про запуск кожного алгоритму для вирішення кожної ЗМТ, а також дубльовану інформацію з TestSetup про налаштування алгоритмів.

Клас PerformanceTester виконує тести в окремому потоці паралельно основному потоку програми, тому графічний інтерфейс користувача не блокується (при цьому, як тільки виконання кожного тесту завершується,

його результат одразу з’являється в списку результатів в програмі). Робота даного класу починається з функції RunTests (лістинг 3.12), яка виконує проведення тесту (разом із замірюванням часу роботи та збором інших статистичних даних) для кожного алгоритму для рішення кожної ЗМТ.

Лістинг 3.12 – Метод RunTests класу PerformanceTester

```
private void RunTests(){
    if (_testSetup.MapFiles != null)
        foreach (var mapFile in _testSetup.MapFiles){
            var map = File.ReadAllText(mapFile)
                .FromJsonStr<VrpMap>();
            foreach (var algId in _testSetup.AlgorithmIds)
                MakeTest(map, algId); }
    if (_testSetup.MapSettings != null)
        foreach (var settings in _testSetup.MapSettings){
            var map = MapBuilder
                .BuildMap(settings, settings.NodeCount * 100,
settings.NodeCount * 100);
            foreach (var algId in _testSetup.AlgorithmIds)
                MakeTest(map, algId);
        }
}
```

Метод MakeTest зберігає інформацію про поточний запуск, вимірює час роботи за допомогою стандартного інструменту StopWatch, запускає знаходження рішення алгоритмом та перевіряє коректність знайденого результату. Після цього отримує значення всіх функцій пристосованості та зберігає їх. Всі результати тестів записуються в колекцію TestResults, з якої потім сформується екземпляр класу TestResultsData.

Для перевірки використовуваних ЗМТ, алгоритмів, а також результатів їх роботи використовуються класи AlgorithmValidator, MapValidator та TestSetupValidator. Кожен з розглянутих класів відповідає за перевірку на окремих етапах проведення тестування. TestSetupValidator здійснює перевірку перед запуском тестів, MapValidator перевіряє тільки ЗМТ, які зчитуються з файлу. AlgorithmValidator перевіряє налаштування алгоритму та результат його роботи (знайдений розв’язок).

3.5 Збір даних про роботу алгоритмів

Для збору детальної інформації про роботу алгоритму в інтерфейс IAlgorithm додано метод WithCollectStats, виклик якого вмикає збір даних в процесі роботи алгоритму. Щоб вимкнути збір даних, треба викликати функцію Reset.

Процес збору детальних даних про роботу розробленого методу є унікальним для кожного алгоритму (тобто потребує окремої індивідуальної реалізації). Наприклад, в ГА для зберігання статистичної інформації на кожній ітерації алгоритму, до класу GeneticAlgorithmState додано поле з типом GeneticAlgorithmStats (лістинг 3.13). При цьому на кожному етапі екземпляр класу GeneticAlgorithmState клонується в окрему колекцію.

Лістинг 3.13 – Клас GeneticAlgorithmStats

```
public class GeneticAlgorithmStats{
    [JsonProperty("w")] public int SelectedForMutation;
    [JsonProperty("e")] public Dictionary<long, int>
MutatedCountById;
    [JsonProperty("r")] public Dictionary<long, long>
MutationTicksById;
    [JsonProperty("t")] public int SelectedForCrossover;
    [JsonProperty("y")] public Dictionary<long, int>
CrossedCountById;
    [JsonProperty("u")] public Dictionary<long, long>
CrossoverTicksById;
    [JsonProperty("i")] public TimeSpan TotalFitnessTime;
    [JsonProperty("o")] public TimeSpan TotalSortingTime;
    public GeneticAlgorithmStats Clone(){
       var clone = (GeneticAlgorithmStats) MemberwiseClone();
       return clone; } }
```

Всі методи ГА продубльовані з приставкою WCS_ та розширені для зберігання статистики. Якщо збір даних ввімкнений, то будуть викликатись саме ці методи. Можливість запуску алгоритму зі збором статистичних даних реалізована тільки для одиночного запуску вирішення ЗМТ, так як такий запуск через PerfrormanceTester потребував би значної кількості пам'яті.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

4.1 Параметри ЗМТ та критерії ефективності роботи алгоритмів

На початку досліджень необхідно сформувати набір ЗМТ, з яким будуть проводитись експерименти. Через наявність обмеження вантажопідйомності та часових вікон, генерація ЗМТ має багато параметрів для налаштувань. Підбір параметрів генерації ЗМТ є важливим етапом досліджень, так як обрані параметри визначають області допустимих значень для даних ЗМТ (матриці вартостей та часу, потреби та часові вікна клієнтів), тип матриць (симетрична або асиметрична), кількість клієнтів. Підібрані параметри повинні бути коректними, щоб для сформованої ЗМТ можна було гарантовано знайти хоча б одне допустиме рішення.

Для проведення експериментів були використані згенеровані ЗМТ, які описуються повнозв'язними графами, що складаються з 200, 400, 600, 800 та 1000 вершин. Кожна сформована ЗМТ має наступні параметри:

- вантажопідйомність кожного транспортного засобу 3000;
- вартість переміщення між двома вузлами лежить в діапазоні від 1 до 2000, матриця – асиметрична;
- потреби клієнтів у вантажі лежать в діапазоні від 1 до 1000;
- час переміщення між двома вузлами в межах від 1 до 1000;
- кожен клієнт має часове вікно, яке генерується в діапазоні від 1 до 3000, протяжність часового вікна в межах від 1 до 2000.

Критеріями ефективності роботи алгоритмів є час знаходження рішення, наближеного до оптимального, та оцінка знайденого рішення за допомогою обраної фітнес-функції. Для оцінки рішень застосовується фітнес-функція (2.3) з наступними коефіцієнтами: $C_{коеф.} = 0.1$, $D_{коеф.} = 1$, $M_{коеф.} = 100$, $K_{коеф.} = 10$. Така комбінація коефіцієнтів забезпечує достатньо збалансовану оцінку рішень для ЗМТ з обраними параметрами.

4.2 Дослідження роботи генетичного алгоритму

ГА мають велику кількість параметрів, які необхідно налагодити для ефективної роботи алгоритму. Розроблене програмне забезпечення надає можливість налаштовувати розмір популяції, класичний алгоритм для створення початкової популяції (разом з параметрами класичного алгоритму, якщо вони є), ймовірність виконання кожного типу схрещування та мутації, стратегію вибору батьківських особин. Підбір параметрів здійснюється експериментальним шляхом. В даній роботі, враховуючи обрані параметри ЗМТ, ГА налаштований наступним чином:

- для генерації початкової популяції використано жадібний алгоритм та модифікацію на його основі, алгоритм Кларка-Райта, метод гілок та меж;
- для схрещування застосовуються оператор PMX з ймовірністю 0.7 та оператор OX з ймовірністю 0.3. Підмножина вершин при роботі PMX кожен раз генерується в межах від 2 до 10;
- вибір батьківських особин відбувається шляхом турнірного відбору з 3 учасників турніру;
- використовуються три типи мутації: інверсна, випадкова перестановка вершин в одному маршруті та обмін підмножин вершин між різними маршрутами. З ймовірністю 0.1, 0.1 та 0.4 відповідно.

Результати фіксувалися при досягненні одного з критеріїв зупинки: алгоритм виконав 5000 ітерацій або 150 без покращення поточного найкращого рішення.

4.2.1 Дослідження впливу налаштувань жадібного алгоритму

Серед розглянутих класичних алгоритмів для створення початкової популяції власний параметр має тільки жадібний алгоритм. Цей параметр визначає випадковим чином на кожному етапі роботи алгоритму кількість найкращих вершин, з яких буде обиратися наступна. Це дозволяє наповнити

початкову популяцію різноманітними рішеннями. Досліджувався вплив даного параметру в інтервалі від 1 до 50 (таблиця 4.1).

Таблиця 4.1 – Порівняння результатів застосування ГА з різними значеннями параметра жадібного алгоритму

Параметр алгоритму	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1	Час, с	31.91	68.78	126.96	176.23	256.92
	Оцінка	98599	173639	269539	435949	459268
2	Час, с	33.17	72.02	131.45	183.17	266.95
	Оцінка	87419	171041	270623	427308	449858
3	Час, с	33.42	72.66	130.44	183.51	267.18
	Оцінка	96970	155205	262917	392754	453697
5	Час, с	34.89	71.82	131.44	182.47	267.7
	Оцінка	80088	155188	250030	385670	452982
7	Час, с	34.88	71.89	131.48	182.65	270.07
	Оцінка	80007	159351	236839	389716	478995
10	Час, с	34.57	72.29	131.72	182.93	268.65
	Оцінка	85807	182037	262860	379924	460272
20	Час, с	33.91	71.99	133.33	185.59	269.85
	Оцінка	98082	191239	246106	374523	470571
30	Час, с	33.76	72.36	132.06	182.86	270.26
	Оцінка	90994	173622	269426	417582	453074
40	Час, с	33.87	74.06	131.82	182.32	268.09
	Оцінка	90325	156448	256651	404398	485103
50	Час, с	33.78	72.32	132.65	183.42	270.28
	Оцінка	88490	164113	254035	411807	470396

В межах параметру жадібного алгоритму від 2 до 50 час роботи ГА на будь-якій кількості вершин приблизно однаковий та не відрізняється більше

ніж на 3.33 секунди. Це свідчить про те, що зміна параметру жадібного алгоритму із значеннями більше 1 не впливає на час роботи ГА. Така поведінка пояснюється тим, що кількість найкращих вершин, які використовуються для вибору наступної, не впливає на складність обчислень. У всіх випадках при відборі вершин розглядаються всі вершини, а вибір наступної здійснюється за допомогою генерації випадкового індексу.

ГА із значенням параметру жадібного алгоритму 1 дозволяє зменшити час обчислень на 2-10 секунд. Це зменшення досягається за рахунок того, що жадібний алгоритм з параметром 1 на однаковому наборі даних генерує одне й те ж саме рішення, а це в свою чергу дозволяє створювати перше рішення за допомогою алгоритму, а інші – клонувати з першого.

Рішення, наближені до найкращого, знаходить ГА з параметром жадібного алгоритму в межах від 3 до 7 (рисунок 4.1). Це пов’язано з тим, що такі значення параметра дозволяють створити достатньо різноманітну початкову популяцію, при цьому в кожному рішенні кожен маршрут складається з переїздів, що достатньо наближені до локальної мінімальної вартості.

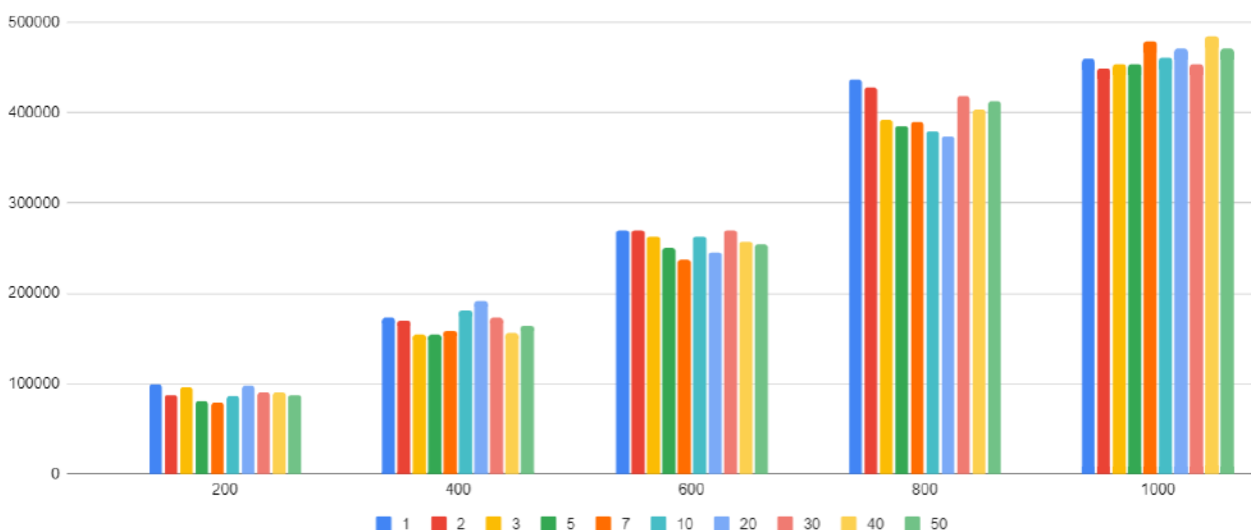

Рисунок 4.1 – Діаграма оцінок знайдених рішень ГА з різним параметром жадібного алгоритму на графах з різною кількістю вершин

Найгірші рішення отримані при використанні параметра із значеннями 1 та 2. Результат пояснюється тим, що в такому випадку формується початкова популяція без різноманітності рішень, внаслідок чого збільшується ймовірність застрягти в локальному екстремумі.

Отриманий результат для параметру в межах від 10 до 50 пов’язаний з тим, що при збільшенні значення параметру збільшується ступінь випадковості рішення. Тобто при значенні параметра, яке дорівнює кількості вершин, знайдені рішення будуть повністю випадковими.

4.2.2 Дослідження впливу кількості ітерацій

ГА зупиняє свою роботу, якщо досягнута задана кількість ітерацій. Проведено дослідження впливу кількості ітерацій на ефективність роботи ГА у поєднанні з жадібним алгоритмом, який має параметр 5 (таблиця 4.2).

Таблиця 4.2 – Порівняння результатів застосування ГА на основі жадібного алгоритму з різною кількістю ітерацій

Кількість ітерацій	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1000	Час, с	6.24	15.51	25.96	46.66	60.83
	Оцінка	98539	308699	531634	749668	895693
2000	Час, с	12.07	29.42	67.03	85.66	108.83
	Оцінка	96459	209009	383191	509342	633535
3000	Час, с	17.7	42.86	91.13	125.71	158.52
	Оцінка	92029	175981	290088	436623	494180
4000	Час, с	23.45	57.1	116.29	147.9	206.06
	Оцінка	85119	166799	270589	404777	463069
5000	Час, с	34.89	71.82	131.44	182.47	267.7
	Оцінка	80088	155188	225030	385670	452982

Для ЗМТ з кількістю вершин більше 400, після 2000 ітерацій значного покращення поточного найкращого рішення протягом кожних 1000 ітерацій не спостерігається (від 2% до 7%) у порівнянні з першими ітераціями (від 15% до 33%). На малій кількості вершин (до 400), ГА знаходить рішення достатньо наближене до найкращого протягом перших 1000 ітерацій. Покращення в межах кожних наступних 1000 ітерацій складає не більше 7%.

Час роботи алгоритму збільшується разом із збільшенням кількості ітерацій. При цьому, чим більше вершин задано в ЗМТ, тим більше часу алгоритму необхідно для виконання кожної ітерації. Наприклад, для ЗМТ з кількістю вершин 200, збільшення кожні 1000 ітерацій лежить в межах від 5 до 13.5 секунд, а для 1000 вершин – від 47.5 до 61.1 секунд.

Якщо проаналізувати результати роботи ГА на основі модифікації жадібного алгоритму (таблиця 4.3), то можна зробити висновок, що час виконання кожної ітерації також залежить від кількості вершин.

Таблиця 4.3 – Порівняння результатів застосування ГА на основі модифікації жадібного алгоритму з різною кількістю ітерацій

Кількість ітерацій	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1000	Час, с	5.84	13.98	28.05	33.99	48.82
	Оцінка	125132	271217	559571	975853	1313604
2000	Час, с	11.39	26.79	59.92	69.33	99.22
	Оцінка	88143	206379	378008	555161	738740
3000	Час, с	16.88	41.59	78.82	101.23	143.72
	Оцінка	76949	186086	277517	402267	513963
4000	Час, с	23.4	53.87	103.21	134.75	192.95
	Оцінка	75263	173191	244633	366641	427765
5000	Час, с	32.61	68.48	127.29	168.32	246.53
	Оцінка	71009	144387	225764	336910	366076

ГА на основі модифікації жадібного алгоритму з великою кількістю ітерацій (більше 3000) недоцільно використовувати для вирішення ЗМТ з кількістю вершин менше 200, так як після досягнення 3000 ітерацій, максимальне покращення оцінки кожні 1000 ітерацій складає 5.7%. До моменту досягнення 3000 ітерацій покращення оцінки кожні 1000 ітерацій лежить в межах від 13% до 30%.

ГА у поєднанні з алгоритмом Кларка-райта до 4000 ітерацій для вирішення ЗМТ з кількістю вершин більше 400 має покращення кожні 1000 ітерацій від 14% до 41% (таблиця 4.4).

Таблиця 4.4 – Порівняння результатів застосування ГА на основі алгоритму Кларка-Райта з різною кількістю ітерацій

Кількість ітерацій	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1000	Час, с	6.57	13.97	26.26	37.01	48.77
	Оцінка	129824	285221	673068	1159178	1461772
2000	Час, с	12.76	27.99	53.37	74.4	97.94
	Оцінка	94281	236280	446550	693849	901761
3000	Час, с	18.47	43.3	77.03	109.79	149.84
	Оцінка	91723	194835	322991	527264	671798
4000	Час, с	27.84	58.31	103.25	139.44	207.79
	Оцінка	87251	166183	266162	431469	547171
5000	Час, с	33.68	67.37	129.66	169.59	245.98
	Оцінка	83573	157343	251058	409035	523466

При використанні ГА на основі алгоритму Кларка-Райта для вирішення ЗМТ з кількістю вершин до 400 достатньо 2000 ітерацій. Після досягнення 2000 ітерацій покращення кожні 1000 ітерацій лежить в межах від 2.7% до 4.9%. Змін у залежності часу роботи алгоритму від кількості ітерацій порівняно з іншими розглянутими гібридними ГА не спостерігається.

ГА на основі методу гілок з відсіканням при вирішенні ЗМТ з кількістю вершин більше 800 протягом всіх ітерацій має покращення кожні 1000 ітерацій від 9% до 50% (таблиця 4.5). Такий результат пояснюється тим, що метод гілок з відсіканням створює початкову популяцію, яка складається з рішень з великими значеннями фітнес-функції (більше 1200000).

Таблиця 4.5 – Порівняння результатів застосування ГА на основі методу гілок з відсіканням з різною кількістю ітерацій

Кількість ітерацій	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1000	Час, с	5.94	15.54	34.55	52.51	86.01
	Оцінка	118110	300831	668114	1284351	1608057
2000	Час, с	12.28	29.05	60.59	86.25	131.91
	Оцінка	96814	262540	457046	648703	1009541
3000	Час, с	18.45	42.54	88.81	123.17	180.19
	Оцінка	95525	211510	362919	549231	772791
4000	Час, с	26.14	58.51	112.72	166.97	230.73
	Оцінка	91283	186315	298737	500656	658376
5000	Час, с	31.47	69.53	140.75	189.95	277.52
	Оцінка	88652	182473	293594	420247	585999

У випадку застосування ГА на основі методу гілок з відсіканням для вирішення ЗМТ з кількістю вершин менше 200, достатньо 2000 ітерацій, так як максимальне покращення кожні наступні 1000 ітерацій складає 4.5%. Для вирішення ЗМТ з кількістю вершин від 200 до 800 достатньо 4000 ітерацій алгоритму через те, що покращення кожні наступні 1000 ітерацій лежить в межах від 1.8% до 2.1%.

ГА у поєднанні з методом гілок з відсіканням має таку ж залежність часу роботи від кількості ітерацій, як і інші розглянуті ГА.

4.2.3 Дослідження алгоритмів генерації початкової популяції

В ході експериментальних досліджень застосування гібридного підходу на основі ГА в комбінації з різними алгоритмами (рисунок 4.2) виявлено, що поєднання ГА з методом гілок з відсіканням знаходить найгірші рішення у порівнянні з іншими. Це пов’язано з тим, що даний метод генерує початкову популяцію без достатнього різноманіття та пропонує рішення з великою кількістю маршрутів. А при використанні методу гілок з відсіканням для вирішення ЗМТ з кількістю вершин більше 600 початковий набір рішень складається з маршрутів з відносно великою вартістю переміщень. При цьому генерація початкової популяції у даного алгоритму займає більше часу, ніж у інших алгоритмів.

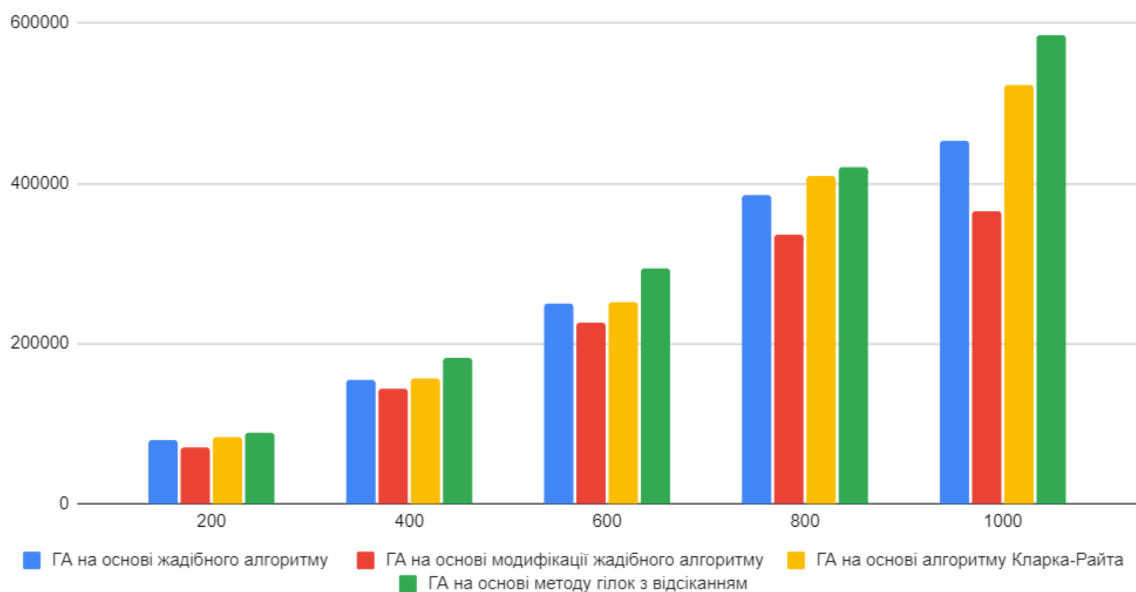

Рисунок 4.2 – Порівняння оцінок знайдених рішень ГА на основі різних класичних алгоритмів

Отриманий результат гібридизації ГА з алгоритмом Кларка-Райта пояснюється тим, що класичний алгоритм пристосований до симетричних матриць вартостей переміщення.

Результати комбінації ГА з жадібним алгоритмом забезпечуються різноманітністю початкової популяції, яка призводить до обстеження більшої області пошуку.

Найкращі рішення отримані за допомогою ГА на основі модифікації жадібного алгоритму, заснованої на максимізації завантаження транспортних засобів. Це пов’язано з тим, що початкова популяція складається з рішень, у яких кількість маршрутів наближена до мінімальної, при цьому вартість переміщення по ним може бути відносно вищою, ніж у інших розглянутих підходів. Це обумовлено тим, що для мінімізації кількості маршрутів при схрещуванні або мутації необхідно здійснити велику кількість тільки таких перестановок вершин, що приведуть до зменшення кількості маршрутів. Для того, щоб зменшити вартість переміщень достатньо однієї або декількох змін з вершинами, тому ГА вирішує проблему з вартістю переїздів (рисунок 4.3) після достатньої кількості ітерацій та знаходить рішення, що є наближеними до найкращого.

Рисунок 4.3 – Зміна середньої вартості переміщень при використанні ГА на основі модифікації жадібного алгоритму на графі з 600 вершинами

Якщо порівнювати час роботи, то найкращі результати мають ГА на основі модифікації жадібного алгоритму та методу Кларка-Райта. Найгірші результати має ГА у поєднанні з методом гілок та меж.

4.3 Дослідження роботи мурашиного алгоритму

Розроблене програмне забезпечення надає можливість налаштовувати велику кількість параметрів мурашиних алгоритмів, що потребує підбору параметрів експериментальним шляхом. В даній роботі мурашині алгоритми налаштовані наступним чином:

- початкова кількість феромону на кожному переміщенні 1;
- коефіцієнт випаровування 0.2;
- константи α та β мають значення 0.9 та 1.1 відповідно;
- модифікатор близькості 300;
- модифікатор розпилення феромонів 50000;
- кількість мурах 40.

Результати фіксувалися при досягненні одного з критеріїв зупинки: алгоритм виконав 1000 ітерацій або 200 без покращення поточного найкращого рішення.

4.3.1 Порівняння різних типів мурашиного алгоритму

В роботі досліджувалося чотири різних типи мурашиних алгоритмів (таблиця 4.6). Класичний мурашиний алгоритм, який передбачає використання мурах, що мають стандартну логіку. Елітарний мурашиний алгоритм, в якому частина мурах є елітними (36 звичайних мурах та 4 елітних). Ранговий мурашиний алгоритм, в якому кожному знайденому рішенню присвоюється ранг, а потім в залежності від рангу кожна мураха розпилює феромон. Лінивий мурашиний алгоритм, в якому частина мурах є лінивими (20 звичайних мурах та 20 лінивих).

Таблиця 4.6 – Порівняння результатів застосування різних типів мурашиного алгоритму на графах з різною кількістю вершин

Тип алгоритму	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
Класичний	Час, с	69	238.16	519.25	928.42	1387.6
	Оцінка	85485	165315	287722	424528	569349
Елітарний	Час, с	64.56	227.04	485.97	850.64	1321.8
	Оцінка	79715	159713	262981	386853	520672
Ранговий	Час, с	124.96	487.45	1069.73	1883.1	3039.3
	Оцінка	94831	191631	316867	537859	731510
Лінивий	Час, с	57.81	194.86	404.14	695.19	1080.4
	Оцінка	87389	164678	284551	446222	557637

Найбільший час роботи спостерігається у рангового мурашиного алгоритму. Це пояснюється тим, що ранговий алгоритм виконує додаткові дії для визначення рангу кожного рішення, а також збільшується кількість обчислень при розпиленні феромонів мурахою.

Найменший час роботи при будь-якій кількості вершин ЗМТ має лінивий мурашиний алгоритм. Такий результат досягається за рахунок лінивих мурах, які зменшують кількість можливих варіантів при виборі наступної вершини для переходу.

Час роботи класичного та елітарного алгоритмів пояснюється тим, що елітні мурахи не здійснюють процедуру випадкового вибору наступної вершини на основі розрахованих ймовірностей, а обирають вершину з найбільшою кількістю феромону.

Рішення з найменшим значенням фітнес-функції у порівнянні з іншими алгоритмами при будь-якій кількості вершин ЗМТ знаходить елітарний мурашиний алгоритм (рисунок 4.4). Рішення з найбільшим значенням функції пристосованості має ранговий мурашиний алгоритм.

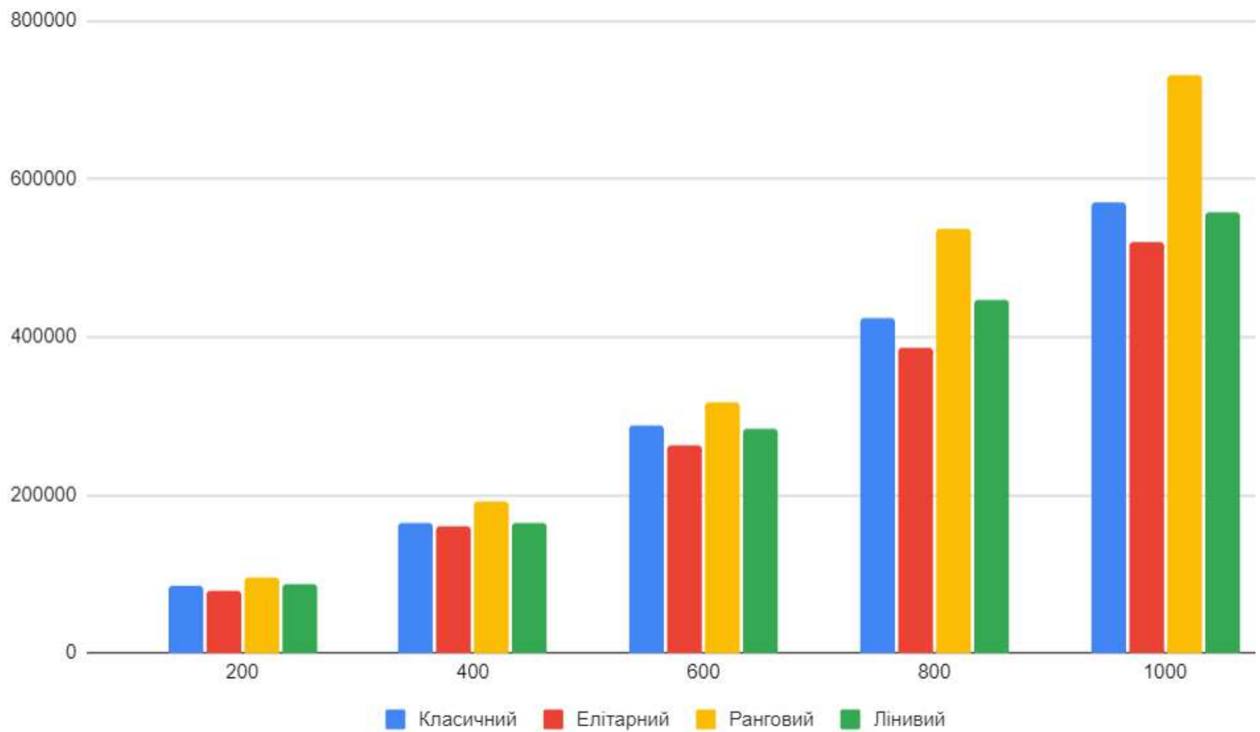

Рисунок 4.4 – Порівняння оцінок знайдених рішень різними типів мурашиних алгоритмів

Значення фітнес-функції рішень, що знайдені лінивим алгоритмом, відрізняються від значень рішень класичного алгоритму не більше, ніж на 4.8%. При цьому лінивий алгоритм дозволяє прискорити роботу алгоритму до 25%, при умові, якщо половина з наявних мурах буде лінивими.

Таким чином, можна зробити висновок, що для вирішення ЗМТ з додатковими обмеженнями доцільно використовувати елітарний та лінивий алгоритми, або гібридний варіант заснований на їх поєднанні. У випадку їх гібридизації кількість елітних та лінивих мурах необхідно підбирати експериментальним шляхом.

4.3.2 Дослідження впливу кількості елітарних мурах

Досліджувався вплив кількості елітарних мурах в інтервалі від 1 до 16 на ефективність роботи мурашиного алгоритму (таблиця 4.7).

Таблиця 4.7 – Порівняння результатів застосування мурашиного алгоритму з різною кількістю елітарних мурах

Кількість мурах	Критерії ефективності	Кількість вершин				
		200	400	600	800	1000
1	Час, с	68.01	233.56	509.23	914.73	1378.42
	Оцінка	82457	163391	271434	397134	542301
2	Час, с	66.87	230.32	497.41	883.12	1346.1
	Оцінка	81923	160765	263244	388982	523916
4	Час, с	64.56	227.04	485.97	850.64	1321.84
	Оцінка	79715	159713	262981	386853	520672
8	Час, с	59.48	216.76	455.06	786.11	1265.5
	Оцінка	79803	158931	264001	390427	523563
16	Час, с	51.32	205.23	429.31	732.42	1227.66
	Оцінка	83458	164129	271301	39512	538712

Найкращі результати спостерігаються при кількості елітних мурах від 2 до 8, при цьому незалежно від кількості вершин. Це пояснюється тим, що при великій кількості елітних мурах збільшується ймовірність застрягти в локальному екстремумі. Час роботи алгоритму зменшується із збільшенням кількості елітних мурах.

Проведено загальний аналіз результатів дослідження гібридного підходу для рішення ЗМТ, заснованого на мурашиних алгоритмах, а також поєднанні ГА та класичних алгоритмів. Найбільшу точність та швидкодію, у порівнянні з іншими алгоритмами, забезпечує гібридизація ГА та модифікації жадібного алгоритму. Мурашині алгоритми потребують в декілька разів більше часу для вирішення ЗМТ, ніж ГА, що робить недоцільним їх використання.

ВИСНОВКИ

В роботі запропонований гібридний підхід до рішення ЗМТ з урахуванням обмеження вантажопідйомності та часових вікон, заснований на поєднанні ГА, мурашиного та класичних алгоритмів. Обрані алгоритми адаптовано під наявні умови задачі та модифіковано з метою збільшення швидкодії та покращення точності.

Проведено дослідження та аналіз результатів роботи розроблених алгоритмів на різних наборах даних. В результаті досліджень виявлено, що найбільшу точність та швидкодію забезпечує гібридизація ГА та модифікації жадібного алгоритму, заснованої на максимізації завантаження транспорту у порівнянні з комбінаціями ГА та інших алгоритмів.

Для тестування та налагодження алгоритмів реалізовано програмний застосунок з графічним інтерфейсом, що дозволяє генерувати ЗМТ з заданими параметрами, проводити запуск наборів алгоритмів для вирішення сформованих наборів задач та переглядати отримані результати. При цьому програмна система має можливість відобразити детальну інформацію про роботу алгоритму.

В майбутньому доцільними є такі напрями розвитку розглянутого підходу:

- застосування декількох класичних алгоритмів одночасно для формування початкової популяції та забезпечення різноманіття;
- дослідження нових методів розв’язання ЗМТ;
- аналіз застосування різних видів кросинговеру, мутацій та інших стратегій відбору батьківських особин.

Результати роботи були опубліковані в журналі «Системи управління, навігації та зв’язку» [28] та представлені в рамках дев'ятої міжнародної науково-технічної конференції «Проблеми інформатизації» [29].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Rodrigue, J. P. The geography of transport systems [Текст] / J. P. Rodrigue, C. Comtois, B. Slack // Routledge. – London, 2013. – 432 p.

2. Goldberg, D. E. Alleles, loci and the traveling salesman problem [Текст] / D. E. Goldberg, R. Lingle // Proceedings of the First International Conference on Genetic Algorithms. – New Jersey, 1985. – P. 154–159.

3. Caceres-Cruz, J. Rich vehicle routing problem: Survey [Текст] / J. Caceres-Cruz, P. Arias // ACM Computing Surveys (CSUR). – 2015. – Vol. 47. – P. 32.

4. Dantzig, G. B. The Truck Dispatching Problem [Текст] / G. B. Dantzig, J. H. Ramser // Management Science. – 1959. – Vol. 6. – P. 80–91.

5. Clarke, G. Scheduling of vehicles from central depot to a number delivery points [Текст] / G. Clark, J. W. Wright // Operations Research. – 1964. – Vol. 12, No. 4 – P. 568–581.

6. Braekers, K. The vehicle routing problem: State of the art classification and review [Текст] / K. Braekers, K. Ramaekers, I. Nieuwenhuese // Computers & Industrial Engineering. – 2016. – Vol. 99. – P. 300–313.

7. Toth, P. The Vehicle Routing Problem [Текст] / P. Toth, D. Vigo // Discrete Mathematics and Applications. – 2002. – 358 p.

8. Golden, B. The Vehicle Routing Problem: Latest Advances and New Challenges [Текст] / B. Golden, S. Raghavan, E. Wasil // Operations Research. – 2008. – Vol. 43. – P. 29–48.

9. Ho, W. A hybrid genetic algorithm for the multi-depot vehicle routing problem [Текст] / W. Ho, G. T. Ho, P. Ji, H. C. Lau // Engineering Applications of Artificial Intelligence. – 2008. – Vol. 21. – No. 4. – P. 548–557.

10. Caramia, M. A heuristic approach for the truck and trailer routing problem [Текст] // M. Caramia, F. Guerriero // Journal of the Operational Research Society. – 2010. – Vol. 61. – P. 1168–1180.

11. Theurich, F. A branch-and-bound approach for a Vehicle Routing Problem with Customer Costs [Текст] / F. Theurich, A. Fischer, G. Scheithauer // EURO Journal on Computational Optimization. – 2021. – Vol. 9. – P. 29–40.

12. Baldacci, R. An Exact Algorithm for the Pickup and Delivery Problem with Time Windows [Текст] / R. Baldacci, E. Bartolini, A. Mingozzi // Operations Research. – 2010. – Vol. 59. – No. 2. – P. 189.

13. Pichpibul, T. A Heuristic Approach Based on Clarke-Wright Algorithm for Open Vehicle Routing Problem [Текст] / T. Pichpibul, R. Kawtummachai // The Scientific World Journal. – 2013. – Vol. 2013. – P. 11.

14. Renaud, J. An improved petal heuristic for the vehicle routing problem [Текст] / J. Renaud, F. F. Bostor, G. Laporte // Journal of Operational Research Society. – 1996. – №47. – P. 329–336.

15. Salhi, S. Improvements to vehicle routing heuristics [Текст] / S. Salhi, G. Rand // Journal of the Operational Research. – 1987. – №38. – P. 293–295.

16. Alba, E. Solving the vehicle routing problem by using cellular genetic algorithms [Текст] / E. Alba, B. Dorronsoro // European Conference on Evolutionary Computation in Combinatorial Optimization. – 2004. – Vol. 3004. – P. 11–20.

17. Gendreau, M. A tabu search heuristic for the vehicle routing problem [Текст] / M. Gendreau, A. Hertz, G. Laporte // Management Science. – 1997. – Vol. 40, No. 10. – P. 1276–1290.

18. Schmitt, L. J. An evaluation of a genetic algorithmic approach to the vehicle routing problem [Текст] / L. J. Schmitt // Working paper, Department of Information Technology Management. – Memphis, 1995. – P. 214–231.

19. Chang, Y. Solve the vehicle routing problem with time windows via a genetic algorithm [Текст] / Y. Chang, L. Chen // Discrete and Continuous Dynamical Systems. – 2007. – P. 240–249.

20. Bullnheimer, B. An improved ant system for the vehicle routing problem [Текст] / B. Bullheimer, R. F. Hartl, C. Strauss // Annals of Operations Research. – 1998. – Vol. 89. – P. 319–328.

21. Fuellerer, G. Ant Colony optimization for the two-dimensional loading vehicle routing problem [Текст] / G. Fuellerer, K. Doerner, R. Hartl, M. Iori // Computers & Operations Research. – 2009. – Vol. 36. – No. 3. – P. 655–673.

22. Aksen, D. Open vehicle routing problem with driver nodes and time deadlines [Текст] / D. Aksen, Z. Zyurt, N. Aras // Journal of the Operational Research Society. – 2006. – Vol. 58. – P. 106–114.

23. Dror, M. A vehicle routing improvement algorithm. Comparison of a Greedy and a Matching implementation for inventory routing [Текст] / M. Dror, L. Levy // Computers & Operations Research. – 1986. – №13. – P. 33–45.

24. Pichpibul, T. An improved Clarke and Wright savings algorithm for the capacitated vehicle routing problem [Текст] / T. Pichpibul, R. Kawtummachai // ScienceAsia. – 2012. – Vol. 38. – P. 307–318.

25. Mitchell, J. E. Branch-and-Cut Algorithms for Combinatorial Optimization Problems [Текст] / J. E. Mitchell // Mathematical Sciences Rensselaer Polytechnic Institute. – Troy, 1999. – P. 19.

26. Blanton, J. Multiple vehicle routing with time and capacity constraints using genetic algorithms [Текст] / J. Blanton, R. Wainwright // Proceedings of the Fifth International Conference on Genetic Algorithms. – Oklahoma, 1993. – P. 452–459.

27. White, T. Revisiting elitism in ant colony optimization [Текст] / T. White, S. Kaegi, T. Oda // In Genetic and Evolutionary Computation Conference. – Berlin, 2003. – P. 122–123.

28. Іващенко, Г. С. Гібридний метод рішення задачі маршрутизації транспорту з урахуванням додаткових обмежень / Г. С. Іващенко, А. С. Скляров, О. Ю. Барковська // Системи управління, навігації та зв’язку. – 2023. – Т. 1. – №71. – С. 31–35.

29. Іващенко, Г. С. Гібридні методи рішення транспортної задачі [Текст] / Г. С. Іващенко, А. С. Скляров // Проблеми інформатизації. – Харків, 2021. – С. 98.