МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет менеджменту

Кафедра економічної кібернетики, комп'ютерних наук та інформаційних технологій

Кваліфікаційна наукова
праця на правах рукопису

АФОНІН Максим Дмитрович

УДК 658.012.32:005.961

КВАЛІФІКАЦІЙНА РОБОТА

СТВОРЕННЯ АЛГОРИТМУ ОПТИМІЗАЦІЇ МАРШРУТІВ ДЛЯ СЛУЖБИ ДОСТАВКИ

Спеціальність – 122 «Комп'ютерні науки»
Галузь знань – 12 «Інформаційні технології»

Подається на здобуття освітнього ступеня бакалавра

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ М.Д. АФОНІН

Науковий керівник: Волосюк Юрій Вікторович, кандидат технічних наук, доцент

Науковий керівник: Жебко Олександр Олегович, асистент

Завідувач кафедри: Тищенко Світлана Іванівна, кандидат педагогічних наук, доцент

Миколаїв–2025

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

TSP – Traveling Salesman Problem (Задача комівояжера) – класична задача комбінаторної оптимізації, що полягає у знаходженні найкоротшого маршруту, який проходить через усі задані точки з поверненням до початкової.

VRP – Vehicle Routing Problem (Задача маршрутизації транспортних засобів) – узагальнення задачі комівояжера, яке враховує множинність транспортних засобів і додаткові обмеження.

VRPTW – Vehicle Routing Problem with Time Windows (Задача маршрутизації транспортних засобів із часовими вікнами) – різновид VRP, де кожна точка доставки має заданий часовий інтервал для обслуговування.

MST – Minimum Spanning Tree (Мінімальне остовне дерево) – підмножина ребер графа, що з’єднує всі вершини з мінімальною сумарною вагою.

GIS – Geographic Information System (Геоінформаційна система) – система для збору, зберігання, аналізу та візуалізації географічних даних.

GPS – Global Positioning System (Глобальна система позиціонування) – технологія для визначення координат об’єктів у реальному часі.

API – Application Programming Interface (Інтерфейс програмування додатків) – набір правил і інструментів для взаємодії між програмними компонентами.

OR-Tools – Optimization Tools (Інструменти оптимізації) – відкрита бібліотека від Google для розв’язання задач комбінаторної оптимізації.

ORION – On-Road Integrated Optimization and Navigation (Інтегрована оптимізація та навігація на дорозі) – система маршрутизації, розроблена компанією UPS.

SAP – Systems, Applications, and Products (Системи, додатки та продукти) – програмне забезпечення для управління бізнес-процесами, включаючи логістику.

AWS – Amazon Web Services (Вебсервіси Amazon) – хмарна платформа для обчислень і зберігання даних.

A* – A-star (Алгоритм A*) – алгоритм пошуку найкоротшого шляху з використанням евристичної функції.

2-opt – Two-Optimization (Двооптимізація) – локальний пошуковий алгоритм для вдосконалення маршрутів шляхом заміни двох ребер.

3-opt – Three-Optimization (Трьохоптимізація) – розширення 2-opt, яке замінює три ребра для покращення маршруту.

O(n) – Порядок складності (асимптотична нотація) – позначення обчислювальної складності алгоритму залежно від розміру вхідних даних n.

NetworkX – Бібліотека Python для роботи з графами та їх аналізу.

SciPy – Бібліотека Python для наукових обчислень, включаючи методи оптимізації.

OSM – OpenStreetMap (Відкриті карти вулиць) – відкрита картографічна платформа для моделювання тфранспортних мереж.

Folium – Бібліотека Python для візуалізації географічних даних на інтерактивних картах.

BGL – Boost Graph Library (Бібліотека графів Boost) – бібліотека C++ для роботи з графами.

ЗМІСТ

АНОТАЦІЯ

Афонін М.Д. *Створення алгоритму оптимізації маршрутів для служби доставки* – Кваліфікаційна наукова праця на правах рукопису. Робота на здобуття освітнього ступеня бакалавра за спеціальністю 122 «Комп'ютерні науки». – Миколаївський національний аграрний університет, Миколаїв, 2025.

У бакалаврській кваліфікаційній роботі досліджено проблему оптимізації маршрутів у системах міської доставки, зумовлену стрімким розвитком електронної комерції та зростанням попиту на швидкі логістичні послуги. Актуальність обраної теми обумовлена необхідністю підвищення ефективності логістичних процесів, зниження витрат на транспорт, поліпшення якості обслуговування клієнтів і зменшення екологічного навантаження.

У роботі проаналізовано теоретичні основи задач комбінаторної оптимізації, зокрема задачі комівояжера (TSP) та маршрутизації транспортних засобів (VRP), а також сучасні евристичні та метаевристичні підходи до їх розв’язання. Проведено огляд існуючих логістичних рішень і систем маршрутизації, серед яких OR-Tools, OptimoRoute, SAP Logistics та інші. На основі аналізу потреб служб доставки сформульовано практичну задачу оптимізації з урахуванням реальних обмежень — кількість точок доставки, часові вікна, витрати палива, місткість транспортних засобів і динамічність замовлень.

У рамках роботи було розроблено гібридний алгоритм, що поєднує метод паралельних заощаджень, генетичний алгоритм і локальну оптимізацію (2-opt). Його реалізовано мовою програмування Python із використанням бібліотек NetworkX, NumPy та Folium. Результати тестування алгоритму на прикладах з 100–300 точками доставки продемонстрували високу ефективність щодо скорочення відстані (до 24 %), зниження витрат пального (до 25 %) та дотримання часових вікон (до 100 %). Час обчислення алгоритму дозволяє застосовувати його в реальному часі для обробки динамічних сценаріїв.

Наукова новизна дослідження полягає у створенні адаптивного алгоритму маршрутизації, здатного враховувати складні логістичні обмеження і використовуватись у комерційних умовах. Практична значущість роботи полягає в можливості її застосування для оптимізації роботи служб доставки, що дозволяє досягти економії коштів і підвищення якості обслуговування клієнтів. Отримані результати можуть стати основою для подальшого розвитку автоматизованих логістичних систем.

Ключові слова: маршрутизація, служба доставки, оптимізація, логістика, генетичний алгоритм, транспортна задача, Python, VRP, TSP, алгоритм 2-opt.

ABSTRACT

Afonin M.D. Creation of a route optimization algorithm for a delivery service – Qualification scientific work in the form of a manuscript. Work for the degree of bachelor in specialty 122 "Computer Science". – Mykolaiv National Agrarian University, Mykolaiv, 2025.

The bachelor's qualification work investigates the problem of route optimization in urban delivery systems, which is caused by the rapid development of e-commerce and the growing demand for fast logistics services. The chosen topic is relevant because of the need to increase the efficiency of logistics processes, reduce transportation costs, improve the quality of customer service, and reduce environmental impact.

The work analyzes the theoretical foundations of combinatorial optimization problems, particularly the traveling salesman problem (TSP) and vehicle routing problem (VRP), as well as modern heuristic and metaheuristic approaches to their solution. A review of existing logistics solutions and routing systems is conducted, including OR-Tools, OptimoRoute, SAP Logistics, and others. Based on the analysis of the needs of delivery services, a practical optimization problem is formulated taking into account real constraints - the number of delivery points, time windows, fuel consumption, vehicle capacity and order dynamics.

As part of the work, a hybrid algorithm was developed that combines the parallel savings method, genetic algorithm and local optimization (2-opt). It was implemented in the Python programming language using the NetworkX, NumPy and Folium libraries. The results of testing the algorithm on examples with 100–300 delivery points demonstrated high efficiency in reducing the distance (up to 24%), reducing fuel consumption (up to 25%) and adhering to time windows (up to 100%). The calculation time of the algorithm allows it to be used in real time for processing dynamic scenarios.

The scientific novelty of the research lies in the creation of an adaptive routing algorithm that can take into account complex logistical constraints and be used in

commercial conditions. The practical significance of the work lies in the possibility of

its application to optimize the operation of delivery services, which allows for cost savings and improved customer service quality. The results obtained can become the basis for the further development of automated logistics systems.

Keywords: routing, delivery service, optimization, logistics, genetic algorithm, transportation problem, Python, VRP, TSP, 2-opt algorithm.

ВСТУП

Сучасний світ характеризується стрімким розвитком електронної комерції, логістичних систем та послуг доставки, що ставить перед суспільством нові виклики у сфері оптимізації транспортних процесів. Зростання обсягів замовлень, потреба у швидкій доставці та прагнення до зниження витрат змушують компанії шукати ефективні рішення для організації маршрутів транспортних засобів. У цьому контексті оптимізація маршрутів для служб доставки стає не лише актуальною задачею, але й ключовим фактором конкурентоспроможності на ринку. Раціонально сплановані маршрути дозволяють зменшити витрати палива, скоротити час доставки, підвищити рівень задоволеності клієнтів та мінімізувати екологічний вплив за рахунок зниження викидів вуглецю. Таким чином, розробка алгоритмів оптимізації маршрутів є важливим напрямом досліджень як у теоретичній, так і в прикладній сферах.

Задача оптимізації маршрутів має глибоке теоретичне підґрунтя і базується на класичних проблемах комбінаторної оптимізації, таких як задача комівояжера (Traveling Salesman Problem, TSP) та задача маршрутизації транспортних засобів (Vehicle Routing Problem, VRP). Ці задачі ускладнюються у реальних умовах через наявність численних обмежень, таких як часові вікна доставки, місткість транспорту, пріоритетність клієнтів, а також динамічні фактори, наприклад, зміни в дорожньому русі чи скасування замовлень. Натомість евристичні та метаевристичні підходи, такі як генетичні алгоритми, алгоритм мурашиної колонії чи імітація відпалу, пропонують практичні рішення, які дозволяють знайти субоптимальні результати за прийнятний час.

Актуальність теми бакалаврської роботи зумовлена не лише теоретичною значущістю, але й практичною цінністю. У реальному світі служби доставки, такі як "Нова Пошта", "Укрпошта", DHL чи Amazon Logistics, щодня стикаються з необхідністю обробки тисяч замовлень, враховуючи при цьому географічні, часові та економічні фактори. Наприклад, за даними

досліджень, оптимізація маршрутів може скоротити транспортні витрати на 10–30%, що є суттєвим показником для компаній з великими логістичними мережами. Водночас інтеграція сучасних технологій, таких як системи GPS, аналіз даних у реальному часі та штучний інтелект, відкриває нові можливості для створення адаптивних і високоефективних алгоритмів.

Мета даної бакалаврської роботи полягає у створенні алгоритму оптимізації маршрутів для служби доставки, який враховуватиме реальні обмеження, такі як кількість точок доставки, часові рамки та витрати палива, з метою підвищення ефективності логістичних процесів. Для досягнення цієї мети у роботі будуть розв'язані **наступні завдання**:

1. Провести аналіз теоретичних основ оптимізації маршрутів, включаючи огляд сучасних методів та графових алгоритмів.
2. Вивчити існуючі рішення у сфері логістичної маршрутизації та оцінити їх ефективність.
3. Сформулювати задачу оптимізації маршрутів на основі потреб конкретної служби доставки.
4. Розробити алгоритм оптимізації з урахуванням визначених обмежень.
5. Реалізувати алгоритм за допомогою засобів програмування (наприклад, Python) та провести його тестування на реальних або синтетичних даних.
6. Порівняти ефективність розробленого алгоритму з іншими підходами та надати рекомендації щодо його впровадження.

Об'єктом дослідження є процеси маршрутизації у системах доставки, а предметом — алгоритми оптимізації маршрутів, які враховують специфіку логістичних задач.

Предметом дослідження є алгоритми та методи оптимізації маршрутів у логістичних системах доставки товарів.

Наукова новизна роботи полягає у розробці алгоритму, адаптованого до реальних умов служби доставки, з акцентом на баланс між точністю розв'язку та обчислювальною ефективністю. Практична значущість дослідження полягає

в можливості застосування результатів для підвищення ефективності роботи комерційних служб доставки, що може бути використано як у локальному, так і в глобальному масштабах.

Робота складається з трьох основних розділів. У першому розділі розглядаються теоретичні основи оптимізації маршрутів, включаючи методи розв’язання задач комівояжера та принципи роботи з транспортними мережами. Другий розділ присвячений аналізу існуючих рішень, інструментів і технологій, а також постановці задачі для конкретного сценарію доставки. У третьому розділі описано розробку, реалізацію та тестування алгоритму, а також надано рекомендації щодо його впровадження. Завершують роботу висновки, список використаної літератури та додатки з допоміжними матеріалами.

Структура та обсяг роботи. Логіка дослідження зумовила структуру кваліфікаційної роботи: вступ, 3 розділи, висновки, список використаних джерел із 32 найменувань. Загальний обсяг 79 сторінок.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ОПТИМІЗАЦІЇ МАРШРУТІВ ТА ЛОГІСТИЧНИХ СИСТЕМ

1.1. Огляд сучасних методів оптимізації в транспортних системах

Оптимізація в транспортних системах є ключовою складовою сучасної логістики, яка дозволяє підвищити ефективність доставки, знизити витрати та оптимізувати використання ресурсів. Зростання обсягів перевезень, ускладнення логістичних мереж і потреба в швидкому реагуванні на зміни в реальному часі зумовлюють необхідність розробки та вдосконалення методів оптимізації. Ці методи застосовуються до широкого спектра задач, таких як планування маршрутів, розподіл транспортних засобів, управління запасами та координація логістичних операцій. У цьому підрозділі розглядаються основні підходи до оптимізації в транспортних системах, включаючи точні, евристичні та метаевристичні методи, а також їхнє практичне застосування в логістиці.

Транспортні системи історично спиралися на математичні моделі для розв’язання задач маршрутизації. Одним із перших і найбільш відомих прикладів є задача комівояжера (Traveling Salesman Problem, TSP), яка полягає у знаходженні найкоротшого маршруту, що проходить через усі задані точки з поверненням до початкової. Ця задача стала основою для багатьох сучасних методів оптимізації маршрутів [3]. Наприклад, у транспортних системах часто використовуються її розширення, такі як задача маршрутизації транспортних засобів (Vehicle Routing Problem, VRP), яка враховує множинність транспортних засобів, їхню місткість і часові обмеження. Згідно з дослідженнями, VRP є однією з центральних задач у логістиці, оскільки вона дозволяє моделювати реальні сценарії доставки [4]. Такі моделі застосовуються в роботі великих компаній, наприклад, Amazon чи DHL, де щодня потрібно оптимізувати маршрути для тисяч транспортних засобів.

Точні методи оптимізації, такі як лінійне програмування, динамічне програмування та методи гілок і меж, забезпечують знаходження оптимального рішення для задач маршрутизації. Лінійне програмування, зокрема, широко

використовується для задач розподілу ресурсів і транспортного планування, коли потрібно мінімізувати витрати або максимізувати прибуток. Наприклад, методи на основі симплекс-алгоритму дозволяють ефективно розв’язувати задачі з великою кількістю змінних і обмежень [6]. Проте точні методи мають суттєвий недолік: їхня обчислювальна складність зростає експоненційно зі збільшенням розміру задачі. У реальних умовах, коли кількість точок доставки перевищує кілька десятків, ці методи стають непрактичними через надмірну витрату часу та обчислювальних ресурсів. Саме тому в сучасних транспортних системах перевага часто надається приблизним методам.

Евристичні методи стали популярними завдяки своїй здатності швидко знаходити субоптимальні рішення для складних задач. Одним із класичних прикладів є алгоритм "найближчого сусіда", який обирає наступну точку маршруту на основі мінімальної відстані від поточної позиції. Цей підхід простий у реалізації й ефективний для невеликих задач, але не гарантує глобальної оптимальності [10]. Інший відомий евристичний метод — алгоритм Кларка-Райта (Clarke-Wright Savings Algorithm), який базується на об’єднанні маршрутів для зменшення загальної відстані. Цей алгоритм широко застосовується в логістичних системах для початкового планування маршрутів і може бути адаптований до задач із часовими вікнами чи обмеженнями місткості [14]. Евристичні методи є основою багатьох комерційних систем оптимізації, таких як Route4Me чи OR-Tools від Google, які використовуються для планування доставки в реальному часі.

Для складніших задач, де потрібно враховувати множинні обмеження (наприклад, часові вікна, пріоритетність клієнтів або динамічні зміни в трафіку), застосовуються метаевристичні методи. Ці методи, такі як генетичні алгоритми, імітація відпалу чи алгоритм мурашиної колонії, базуються на ітеративному пошуку кращих рішень у великому просторі можливостей. Генетичні алгоритми, наприклад, імітують процес еволюції, використовуючи оператори відбору, кросинговеру та мутації для генерації оптимальних маршрутів [1]. Вони показали високу ефективність у задачах VRP із великою

кількістю змінних, дозволяючи знаходити рішення, близькі до оптимальних, за розумний час. Такі підходи активно використовуються в системах доставки, де необхідно адаптуватися до змін у реальному часі, наприклад, у разі скасування замовлення чи заторів на дорогах.

Сучасні транспортні системи також інтегрують технології штучного інтелекту та аналізу великих даних для підвищення точності оптимізації. Наприклад, прогнозування часу доставки на основі історичних даних про трафік або погодні умови дозволяє створювати більш адаптивні маршрути [2]. Компанії, такі як UPS, використовують системи на основі машинного навчання для динамічного перепланування маршрутів, що скорочує витрати палива на 10–15% щорічно. Водночас інтеграція геоінформаційних систем (GIS) і GPS-технологій забезпечує точне відстеження транспортних засобів і коригування маршрутів у реальному часі. Такі технології роблять оптимізацію не лише теоретичною задачею, але й практичним інструментом для щоденної роботи логістичних компаній.

Порівняння різних методів показує, що вибір оптимального підходу залежить від специфіки задачі. Точні методи ідеально підходять для невеликих задач із чітко визначеними параметрами, тоді як евристичні та метаевристичні методи є незамінними для великих і динамічних систем. Наприклад, дослідження показують, що генетичні алгоритми перевершують алгоритм "найближчого сусіда" у задачах із більш ніж 50 точками доставки за критерієм загальної довжини маршруту [1]. Водночас комбінація кількох методів, наприклад, використання евристики для початкового рішення з подальшим уточненням метаевристикою, часто дає найкращі результати в реальних умовах.

Сучасні методи оптимізації в транспортних системах охоплюють широкий спектр підходів, від класичних математичних моделей до інноваційних рішень на основі штучного інтелекту. Їхнє застосування дозволяє вирішувати складні логістичні задачі, враховуючи як економічні, так і операційні аспекти. У контексті служби доставки ці методи стають основою для

розробки алгоритмів, які здатні адаптуватися до реальних умов, таких як обмеження часу, витрати палива чи динамічні зміни попиту. Подальший аналіз теоретичних основ і практичних інструментів у наступних підрозділах дозволить детальніше розглянути ці аспекти та сформулювати підхід до створення власного алгоритму оптимізації.

1.2. Теоретичні основи графових алгоритмів

Графові алгоритми є фундаментальною основою для розв'язання задач оптимізації маршрутів у транспортних і логістичних системах. Вони базуються на теорії графів — розділі математики, що вивчає структури, які складаються з вершин (вузлів) і ребер (зв'язків між ними). У контексті служби доставки вершини можуть представляти точки доставки, склади чи перехрестя, а ребра — дороги або відстані між цими об'єктами. Графові алгоритми дозволяють моделювати транспортні мережі, знаходити найкоротші шляхи, оптимізувати маршрути та враховувати різноманітні обмеження, такі як часові вікна чи місткість транспортних засобів. У цьому підрозділі розглядаються основні принципи теорії графів, ключові алгоритми та їхнє застосування до задач маршрутизації.

Теорія графів як наукова дисципліна бере початок із роботи Леонарда Ейлера у XVIII столітті, коли він розв'язав задачу про сім мостів Кенігсберга. Ця задача заклала основи для розуміння зв'язності графів і стала першим прикладом аналізу маршрутів [7]. Сучасні графові алгоритми спираються на поняття орієнтованих і неорієнтованих графів, де ребра можуть мати вагу (наприклад, відстань або час проходження). Для транспортних систем зазвичай використовуються зважені графи, де ваги відображають реальні характеристики мережі, такі як довжина дороги чи витрати палива. Теоретичні основи таких графів детально описані в літературі, присвяченій комбінаторній оптимізації [3].

Одним із базових завдань у теорії графів є пошук найкоротшого шляху між двома вершинами. Класичним рішенням цієї задачі є алгоритм Дейкстри,

який знаходить найкоротший шлях від однієї вершини до всіх інших у графі з невід'ємними вагами ребер. Алгоритм працює шляхом поступового розширення множини відвіданих вершин, обираючи на кожному кроці вершину з мінімальною сумарною вагою шляху [11]. У логістиці цей метод застосовується, наприклад, для визначення найшвидшого маршруту доставки від складу до клієнта. Проте алгоритм Дейкстри має обмеження: він не враховує множинність транспортних засобів чи складніші обмеження, що робить його менш придатним для задач типу VRP.

Для пошуку найкоротших шляхів між усіма парами вершин у графі використовується алгоритм Флойда-Воршалла. Цей алгоритм базується на динамічному програмуванні й дозволяє знаходити оптимальні відстані в графі, враховуючи як прямі, так і непрямі зв'язки між вершинами [12]. Його перевагою є здатність працювати з графами, що містять від'ємні ваги (за умови відсутності від'ємних циклів), що може бути корисним при моделюванні транспортних мереж із субсидіями чи штрафами. Однак висока обчислювальна складність ($O(n^3)$, де n — кількість вершин) обмежує його використання в реальних системах із великою кількістю точок доставки.

Іншим важливим класом графових алгоритмів є алгоритми для пошуку мінімального остовного дерева (Minimum Spanning Tree, MST), такі як алгоритми Краскала та Прима. Вони знаходять підмножину ребер, що з'єднує всі вершини графа з мінімальною сумарною вагою [15]. Хоча ці алгоритми рідше застосовуються безпосередньо до задач маршрутизації, вони можуть бути корисними для попереднього аналізу транспортної мережі, наприклад, для побудови базової інфраструктури чи кластеризації точок доставки. У поєднанні з іншими методами MST може слугувати основою для евристичних підходів до оптимізації маршрутів.

Задача комівояжера (TSP), яка є центральною для маршрутизації, також має графову природу. У термінах теорії графів вона формулюється як пошук гамільтонового циклу мінімальної ваги — шляху, що проходить через усі вершини графа рівно один раз із поверненням до початкової точки. Точне

розв’язання TSP можливе за допомогою методів динамічного програмування, таких як алгоритм Хелда-Карпа, який має складність $O(n^2 2^n)$ [5]. Проте для великих графів (більше 20–30 вершин) такі методи стають непрактичними, що зумовлює використання евристичних алгоритмів, таких як "найближчий сусід" чи 3-opt, які базуються на локальному пошуку та поступовому вдосконаленні маршруту.

У реальних транспортних системах часто виникає потреба враховувати динамічні зміни, такі як затори чи скасування замовлень. Для цього застосовуються адаптивні графові алгоритми, наприклад, алгоритм A*, який є розширенням алгоритму Дейкстри. A* використовує евристичну функцію (наприклад, евклідову відстань) для прискорення пошуку найкоротшого шляху, що робить його ефективним для задач із великими графами [13]. Цей алгоритм широко застосовується в системах GPS-навігації, таких як Google Maps, де потрібна швидка реакція на поточні умови.

Графові алгоритми також є основою для моделювання складніших задач, таких як VRP із часовими вікнами (VRPTW) або багатоагентна маршрутизація. У цих випадках граф розширюється додатковими вимірами, такими як час чи місткість, а алгоритми адаптуються для врахування цих обмежень. Наприклад, методи на основі графового представлення дозволяють перетворити задачу VRP у задачу пошуку шляхів у багатошаровому графі, де кожен шар відповідає певному транспортному засобу чи часовому інтервалу [4].

Теоретичні основи графових алгоритмів надають потужний інструментарій для розв’язання задач оптимізації маршрутів. Алгоритми Дейкстри, Флойда-Воршалла, A* та методи для TSP і VRP формують базу для моделювання транспортних мереж і пошуку оптимальних рішень. Їхнє застосування в логістиці дозволяє не лише знаходити найкоротші шляхи, але й адаптувати маршрути до реальних умов, що є критично важливим для служб доставки. У наступних підрозділах ці теоретичні принципи будуть використані для аналізу практичних підходів до оптимізації маршрутів.

1.3. Оптимізація на транспортних мережах: принципи, задачі, підходи

Оптимізація на транспортних мережах є ключовим елементом логістичних систем, що дозволяє ефективно управляти потоками товарів, транспортними засобами та ресурсами. Транспортні мережі являють собою складні структури, які моделюються як графи, де вершини відповідають пунктам (складам, клієнтам, перехрестям), а ребра — шляхам сполучення з певними характеристиками, такими як відстань, час або витрати. Метою оптимізації є знаходження рішень, які мінімізують витрати (палива, часу, ресурсів) або максимізують продуктивність (кількість доставок, швидкість виконання). У цьому підрозділі розглядаються основні принципи оптимізації на транспортних мережах, типові задачі, що виникають у цій сфері, та підходи до їх розв'язання.

Оптимізація транспортних мереж базується на кількох фундаментальних принципах. Перший із них — це принцип мінімальних витрат, який передбачає вибір маршрутів і розподіл ресурсів таким чином, щоб сумарні витрати (фінансові, часові чи екологічні) були найменшими. Другий принцип — це врахування обмежень, таких як пропускна здатність доріг, місткість транспортних засобів, часові вікна доставки чи доступність інфраструктури. Третій принцип — адаптивність, тобто здатність системи реагувати на динамічні зміни, наприклад, затори, погодні умови чи нові замовлення [8]. Ці принципи формують основу для математичного моделювання транспортних мереж, яке зазвичай здійснюється за допомогою теорії графів і комбінаторної оптимізації.

Транспортні мережі можуть бути як статичними, де всі параметри відомі заздалегідь, так і динамічними, де умови змінюються в реальному часі. Наприклад, у статичній мережі оптимальний маршрут для доставки можна розрахувати один раз на основі фіксованих відстаней і часу. У динамічній мережі, навпаки, потрібні алгоритми, які постійно оновлюють маршрути на основі поточних даних, отриманих від GPS чи систем моніторингу трафіку [9]. Такий підхід вимагає інтеграції прогнозних моделей і технологій обробки

великих даних, що значно ускладнює задачу, але підвищує її практичну цінність.

Серед типових задач оптимізації на транспортних мережах виділяються кілька ключових. Перша — це задача пошуку найкоротшого шляху, яка полягає у знаходженні маршруту з мінімальною відстанню або часом між двома точками. Ця задача часто розв’язується алгоритмами, такими як Дейкстри чи A*, і є основою для більш складних моделей [13]. Друга — задача комівояжера (TSP), яка вимагає побудови циклічного маршруту через усі задані точки з мінімальними витратами. У логістиці вона застосовується для планування маршрутів одного транспортного засобу [5].

Третя важлива задача — це маршрутизація транспортних засобів (VRP), яка враховує множинність транспортних засобів, їхню місткість і обмеження часу. Наприклад, у варіації VRP із часовими вікнами (VRPTW) кожна точка доставки має певний інтервал часу, протягом якого доставка повинна бути виконана [18]. Ця задача є більш реалістичною для служб доставки, оскільки відображає реальні умови роботи. Четверта задача — це оптимізація потоків у мережі, яка фокусується на розподілі вантажів між транспортними засобами та маршрутами для максимізації пропускної здатності чи мінімізації затримок. Такі задачі часто моделюються за допомогою методів мережевого аналізу, наприклад, алгоритму Форда-Фалкерсона [16].

Існує кілька основних підходів до оптимізації на транспортних мережах, кожен із яких має свої переваги та обмеження. Точні методи, такі як лінійне програмування та динамічне програмування, гарантують знаходження оптимального рішення, але їхня обчислювальна складність робить їх придатними лише для задач невеликого масштабу. Наприклад, методи гілок і меж ефективно розв’язують задачу TSP для графів із кількома десятками вершин, але стають непрактичними для сотень чи тисяч точок доставки [6]. У реальних транспортних мережах із великою кількістю змінних точні методи часто комбінуються з попередніми спрощеннями задачі, такими як кластеризація точок.

Евристичні методи пропонують швидші, хоча й не завжди оптимальні, рішення. Алгоритм "паралельних заощаджень" (Parallel Savings), наприклад, об’єднує маршрути на основі економії відстані чи часу, що робить його популярним у логістичних системах [15]. Ще один приклад — алгоритм 3-opt, який ітеративно вдосконалює маршрут шляхом заміни трьох ребер на більш ефективні зв’язки [10]. Такі методи широко застосовуються в комерційних системах, оскільки дозволяють швидко обробляти великі обсяги даних, хоча їхня ефективність залежить від якості початкового рішення.

Метаевристичні підходи, такі як генетичні алгоритми та імітація відпалу, є найбільш гнучкими для складних задач із множинними обмеженнями. Генетичні алгоритми моделюють еволюційний процес, створюючи популяцію маршрутів і вдосконалюючи їх через ітерації [1]. Вони ефективні для задач типу VRPTW, де потрібно одночасно враховувати часові вікна, місткість і відстань. Алгоритм мурашиної колонії, у свою чергу, імітує поведінку мурах, які залишають феромонні сліди на кращих маршрутах, що дозволяє знаходити оптимальні шляхи в динамічних мережах [17]. Ці методи особливо цінні для великих транспортних мереж, де точні рішення недосяжні через обчислювальну складність.

Сучасні підходи до оптимізації також включають гібридні методи, які поєднують переваги точних, евристичних і метаевристичних алгоритмів. Наприклад, початкове рішення може бути отримане за допомогою евристики "найближчого сусіда", а потім вдосконалене генетичним алгоритмом [1]. Інший приклад — використання машинного навчання для прогнозування параметрів мережі (наприклад, часу в дорозі), що потім інтегрується в класичні алгоритми оптимізації [2]. Такі гібридні системи застосовуються в реальному часі в таких сервісах, як Uber Freight чи Google Maps, де потрібна висока швидкість і адаптивність.

Оптимізація на транспортних мережах спирається на принципи мінімальних витрат, врахування обмежень і адаптивності, які реалізуються через різноманітні задачі — від пошуку найкоротшого шляху до складних

сценаріїв VRP. Підходи до їх розв’язання варіюються від точних методів, які забезпечують оптимальність, до евристичних і метаевристичних, які пропонують практичні рішення для великих і динамічних систем. У контексті служби доставки ці методи дозволяють створювати ефективні маршрути, враховуючи реальні умови роботи. Подальший аналіз у наступних підрозділах зосередиться на застосуванні цих принципів до конкретних задач маршрутизації.

1.4. Евристичні та точні методи розв'язання задач комівояжера

Задача комівояжера (Traveling Salesman Problem, TSP) є однією з ключових задач комбінаторної оптимізації, яка має широке застосування в логістиці, транспортних системах і плануванні маршрутів. Її суть полягає у знаходженні найкоротшого гамільтонового циклу — маршруту, що проходить через усі задані вершини графа рівно один раз із поверненням до початкової точки. У контексті служби доставки TSP моделює ситуацію, коли один транспортний засіб має відвідати всі точки доставки з мінімальними витратами часу чи відстані. Оскільки TSP належить до класу NP-складних задач, її розв’язання потребує різних підходів — від точних методів, які гарантують оптимальність, до евристичних, які пропонують швидкі, але приблизні рішення. У цьому підрозділі розглядаються основні методи розв’язання TSP, їхні переваги, недоліки та практичне застосування.

Точні методи спрямовані на знаходження глобально оптимального рішення для TSP шляхом вичерпного аналізу всіх можливих варіантів або їх ефективного скорочення. Одним із найвідоміших підходів є метод повного перебору (Brute Force), який обчислює довжину всіх можливих гамільтонових циклів і обирає найкоротший. Однак його обчислювальна складність становить O(n!), де n — кількість вершин, що робить його придатним лише для дуже малих задач (до 10–12 вершин) [19]. У реальних транспортних системах із десятками чи сотнями точок доставки цей метод стає абсолютно непрактичним через експоненційне зростання часу виконання.

Більш ефективним точним методом є алгоритм Хелда-Карпа, який базується на динамічному програмуванні. Цей алгоритм зменшує складність до $O(n^2 2^n)$, що є значним покращенням порівняно з повним перебором. Він працює шляхом розбиття задачі на підзадачі, де для кожної підмножини вершин обчислюється найкоротший шлях із урахуванням уже відвіданих точок [5]. Хоча алгоритм Хелда-Карпа дозволяє розв'язувати TSP для графів із 20–30 вершинами, його використання в логістиці обмежене через високу потребу в обчислювальних ресурсах для великих мереж. Проте він часто слугує еталоном для порівняння з іншими методами.

Ще одним точним підходом є метод гілок і меж (Branch and Bound), який скорочує простір пошуку, відкидаючи заздалегідь неоптимальні гілки. Цей метод використовує нижню оцінку вартості маршруту (наприклад, на основі мінімального остовного дерева) для виключення неефективних комбінацій [6]. У задачах із симетричними графами (де відстань від A до B дорівнює відстані від B до A) метод гілок і меж може бути досить ефективним, дозволяючи розв'язувати TSP для графів із кількома десятками вершин. Однак у реальних транспортних мережах із асиметричними відстанями чи додатковими обмеженнями (наприклад, часовими вікнами) його продуктивність знижується.

Евристичні методи не гарантують оптимальності, але пропонують швидкі й практичні рішення для великих задач, що робить їх незамінними в логістичних системах. Одним із найпростіших і найпоширеніших є алгоритм "найближчого сусіда" (Nearest Neighbor). Він починає з довільної вершини й на кожному кроці обирає найближчу ще не відвідану вершину, поки всі точки не будуть включені в маршрут, після чого повертається до початкової точки [10]. Складність цього алгоритму становить $O(n^2)$, що дозволяє обробляти великі графи. Проте якість рішення залежить від початкової вершини й може бути значно гіршою за оптимальну — у середньому довжина маршруту перевищує оптимальну на 25–30%.

Інший популярний евристичний метод — алгоритм вставки (Insertion Heuristic), який поступово будує маршрут, додаючи вершини одну за одною на

основі критерію мінімального збільшення загальної відстані. Наприклад, "найдешевша вставка" (Cheapest Insertion) обирає вершину й місце її вставки так, щоб приріст довжини маршруту був мінімальним [20]. Цей метод ефективний для задач із сотнями вершин і часто застосовується в системах доставки як базове рішення, яке потім можна вдосконалити. Його перевагою є простота реалізації та відносно хороша якість результатів у порівнянні з "найближчим сусідом".

Для підвищення якості рішень використовуються локальні пошукові методи, такі як 2-opt і 3-opt. Алгоритм 2-opt ітеративно замінює два ребра маршруту на два нові, якщо це зменшує загальну довжину, доки подальші покращення неможливі [24]. Алгоритм 3-opt розширює цей підхід, розглядаючи заміну трьох ребер, що забезпечує кращі результати, але потребує більше обчислень [10]. Ці методи часто застосовуються в комерційних системах маршрутизації, таких як OR-Tools, завдяки їхній здатності швидко вдосконалювати початкові маршрути, отримані іншими евристиками.

Точні методи, такі як Хелда-Карпа чи гілок і меж, ідеально підходять для невеликих задач, де потрібна гарантія оптимальності, наприклад, при плануванні маршрутів для одного кур’єра з 10–15 точками доставки. Однак їхня обчислювальна складність робить їх непридатними для великих транспортних мереж, де кількість вершин може сягати сотень чи тисяч. У таких випадках перевага надається евристичним методам, які забезпечують субоптимальні рішення за прийнятний час. Наприклад, алгоритм "найближчого сусіда" може бути реалізований у реальному часі для динамічної маршрутизації, коли нові замовлення надходять під час роботи кур’єра [20].

У реальних логістичних системах часто застосовуються комбінації методів. Наприклад, початковий маршрут, побудований за допомогою "найдешевшої вставки", може бути вдосконалений алгоритмом 2-opt або 3-opt для досягнення кращої ефективності [24]. Дослідження показують, що такі гібридні підходи дозволяють скоротити довжину маршрутів на 10–20% порівняно з простими евристиками, наближаючись до оптимальних рішень за

значно менший час, ніж точні методи [5]. У системах доставки, таких як UPS чи "Нова Пошта", ці методи інтегруються з геоінформаційними системами для врахування реальних умов, таких як пробки чи погодні фактори.

Точні методи розв’язання задач комівояжера, такі як Хелда-Карпа чи гілок і меж, забезпечують оптимальні рішення, але обмежені масштабами задач через високу обчислювальну складність. Евристичні методи, зокрема "найближчий сусід", вставка та локальний пошук (2-opt, 3-opt), пропонують швидкі й практичні альтернативи, які є основою для оптимізації маршрутів у великих транспортних мережах. У контексті служби доставки вибір методу залежить від розміру задачі, доступних ресурсів і вимог до точності. Подальший аналіз у роботі зосередиться на адаптації цих методів до реальних умов логістики.

Висновок до розділу 1

У першому розділі було здійснено комплексний аналіз сучасних методів оптимізації в транспортних системах, що є фундаментом для побудови ефективних логістичних рішень. встановлено, що задача оптимізації маршрутів має глибоке теоретичне підґрунтя, базується на класичних задачах комбінаторної оптимізації та активно розвивається в напрямі застосування графових моделей, евристичних і метаевристичних методів.

Розглянуто точні методи (лінійне та динамічне програмування, гілки і межі), які гарантують оптимальність розв’язків, але втрачають ефективність при масштабуванні задач. натомість евристичні підходи, зокрема алгоритм "найближчого сусіда", кларка-райта, методи вставки та локального пошуку (2-opt, 3-opt), забезпечують швидке отримання прийнятних результатів для задач середньої складності. метаевристики (генетичні алгоритми, алгоритм мурашиної колонії, імітація відпалу) довели свою ефективність у складних і динамічних умовах, особливо при великій кількості обмежень.

Окрему увагу приділено графовим алгоритмам, зокрема дейкстри, a*, флойда-воршалла та mst-алгоритмам, які є основою моделювання транспортних

мереж. їх поєднання з комбінаційними методами дозволяє адаптувати маршрути до змінних умов, таких як часові вікна, витрати палива чи обмеження місткості.

Узагальнення теоретичних підходів дозволяє зробити висновок, що оптимізація маршрутів у сучасних службах доставки є багаторівневим процесом, який вимагає балансу між точністю, швидкодією та адаптивністю алгоритмів. отримані результати створюють методологічну основу для формулювання практичної задачі оптимізації маршрутів у наступних розділах та розробки власного гібридного алгоритму з урахуванням специфіки реальних логістичних умов.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ ОПТИМІЗАЦІЇ МАРШРУТІВ ДЛЯ СЛУЖБИ ДОСТАВКИ

2.1. Аналіз сучасних систем оптимізації логістичних маршрутів

Сучасні логістичні системи стикаються з необхідністю обробки великих обсягів даних, врахування динамічних умов і забезпечення швидкої доставки, що робить оптимізацію маршрутів критично важливою. Завдяки розвитку інформаційних технологій з’явилися численні системи, які автоматизують процес планування маршрутів, використовуючи як класичні алгоритми, так і новітні підходи на основі штучного інтелекту. Ці системи застосовуються в різних галузях — від поштових служб і кур’єрських компаній до управління автопарками та доставки продуктів. У цьому підрозділі розглядаються основні сучасні системи оптимізації логістичних маршрутів, їхні функціональні можливості, алгоритмічна основа та практичне застосування, а також проводиться порівняльний аналіз у вигляді таблиці.

Однією з найвідоміших систем є OR-Tools від Google, яка являє собою відкритий програмний інструментарій для розв’язання задач комбінаторної оптимізації, зокрема VRP і TSP. OR-Tools підтримує як точні методи (наприклад, гілок і меж), так і евристичні підходи (локальний пошук, "найближчий сусід") [21]. Система дозволяє враховувати різноманітні обмеження, такі як часові вікна, місткість транспорту та пріоритетність клієнтів, що робить її популярною серед розробників логістичних рішень. Наприклад, OR-Tools використовується в сервісах доставки для побудови маршрутів у реальному часі, інтегруючись із даними GPS і картографічними платформами, такими як Google Maps.

Іншим прикладом є Route4Me, комерційна платформа, орієнтована на малі та середні компанії. Ця система використовує евристичні алгоритми, такі як "паралельні заощадження" та 2-opt, для швидкого планування маршрутів із

підтримкою до 10 000 точок доставки [15]. Route4Me інтегрується з мобільними додатками, дозволяючи водіям отримувати оновлені маршрути в реальному часі, а менеджерам — відстежувати виконання доставок. Її перевагою є простота використання та адаптивність до динамічних умов, таких як додавання нових замовлень протягом дня. Проте система менш гнучка в порівнянні з OR-Tools через обмежений доступ до алгоритмічного ядра.

Система OptimoRoute фокусується на автоматизації логістики для служб доставки та виїзного обслуговування. Вона використовує гібридні методи, поєднуючи евристики з метаевристиками, такими як генетичні алгоритми, для оптимізації маршрутів із урахуванням часових вікон і розкладів водіїв [1]. OptimoRoute також пропонує аналітику продуктивності, що дозволяє оцінити ефективність маршрутів за такими показниками, як витрати палива чи час доставки. Ця система популярна серед компаній, які потребують детального планування, наприклад, у сфері доставки їжі чи медичних послуг.

Великі логістичні оператори, такі як UPS, розробили власні системи оптимізації, зокрема ORION (On-Road Integrated Optimization and Navigation). ORION використовує комбінацію точних методів і метаевристик для планування маршрутів для тисяч вантажівок щодня [22]. Система враховує не лише відстань, але й такі фактори, як напрямок руху (наприклад, мінімізація лівих поворотів у країнах із правостороннім рухом), пробки та історичні дані про трафік. За оцінками UPS, ORION скоротив загальну відстань маршрутів на 160 мільйонів кілометрів щорічно, що свідчить про високу ефективність таких рішень у масштабних операціях.

Ще одна система — SAP Logistics Business Network — інтегрує оптимізацію маршрутів із управлінням ланцюгами поставок. Вона базується на хмарних технологіях і використовує методи машинного навчання для прогнозування часу доставки та адаптації маршрутів до змін у реальному часі [23]. SAP підходить для великих корпорацій, які координують доставку через розгалужені транспортні мережі, але її впровадження вимагає значних інвестицій і складної інтеграції з існуючими системами.

Порівняльний аналіз сучасних систем оптимізації логістичних маршрутів наведено в таблиці 2.1. Вона включає ключові характеристики, такі як тип алгоритмів, підтримувані обмеження, масштабованість і сфера застосування, що дозволяє оцінити їхню придатність для різних сценаріїв доставки.

Таблиця 2.1 - Порівняльний аналіз сучасних систем оптимізації логістичних маршрутів

Система	Тип алгоритмів	Підтримувані обмеження	Масштабованість	Сфера застосування
OR-Tools	Точні (гілок і меж), евристичні (2-opt, "найближчий сусід")	Часові вікна, місткість, пріоритетність	До 1000+ точок	Розробка рішень, малі/середні компанії
Route4Me	Евристичні ("паралельні заощадження", 2-opt)	Часові вікна, динамічні замовлення	До 10 000 точок	Кур’єрські служби, доставка
OptimoRoute	Гібридні (генетичні алгоритми, евристики)	Часові вікна, розклади, місткість	До 5000 точок	Доставка їжі, виїзне обслуговування
UPS ORION	Точні + метаевристики	Трафік, напрямок руху, часові вікна	Мільйони доставок щодня	Великі логістичні оператори
SAP Logistics	Машинне навчання, евристики	Часові вікна, прогнозування	Залежить від інфраструктури	Корпоративна логістика

Системи, такі як OR-Tools, вирізняються гнучкістю й відкритим кодом, що дозволяє адаптувати їх під специфічні потреби, але вимагає від користувачів знань програмування [21]. Route4Me і OptimoRoute орієнтовані на простоту використання, що робить їх доступними для компаній без великих технічних команд, однак їхні алгоритми менш прозорі й обмежені в порівнянні з кастомізованими рішеннями [15]. ORION від UPS демонструє високу ефективність у масштабних операціях, але його розробка й підтримка є занадто дороговартісними для малих служб доставки [22]. SAP Logistics підходить для інтеграції в складні ланцюги поставок, але потребує значних ресурсів для впровадження [23].

Сучасні системи оптимізації логістичних маршрутів пропонують широкий спектр рішень — від відкритих інструментів, таких як OR-Tools, до спеціалізованих платформ, як ORION і SAP Logistics. Їхня ефективність залежить від типу алгоритмів, здатності враховувати обмеження та масштабованості. Аналіз показує, що для служби доставки середнього розміру оптимальними можуть бути гібридні системи з підтримкою евристичних і метаевристичних методів, які балансують між швидкістю й точністю. Ці висновки стануть основою для постановки задачі оптимізації в наступних підрозділах.

2.2. Огляд інструментів та технологій для моделювання та реалізації алгоритмів

Розробка та впровадження алгоритмів оптимізації маршрутів для служб доставки потребує використання спеціалізованих інструментів і технологій, які дозволяють моделювати транспортні мережі, реалізовувати алгоритми та тестувати їх на реальних або синтетичних даних. Сучасні рішення охоплюють програмні бібліотеки, мови програмування, геоінформаційні системи (GIS) та платформи для аналізу даних. Ці інструменти забезпечують не лише обчислювальну базу, але й можливості інтеграції з реальними логістичними процесами, такими як відстеження транспорту чи прогнозування часу доставки.

У цьому підрозділі розглядаються ключові інструменти й технології для моделювання та реалізації алгоритмів оптимізації маршрутів, а також наводиться їхній порівняльний аналіз у таблиці.

Одним із найпоширеніших інструментів для реалізації алгоритмів є мова програмування Python, завдяки її простоті, широкій екосистемі бібліотек і підтримці наукових обчислень. У контексті оптимізації маршрутів популярними бібліотеками є NetworkX і SciPy. NetworkX дозволяє моделювати транспортні мережі як графи, реалізовувати базові алгоритми (наприклад, Дейкстри чи "найближчий сусід") і візуалізувати результати [25]. SciPy, у свою чергу, підтримує методи лінійного програмування та оптимізації, які можуть бути застосовані до задач TSP і VRP [26]. Python також інтегрується з бібліотеками машинного навчання, такими як TensorFlow, що дає змогу створювати адаптивні алгоритми на основі прогнозних моделей.

Іншим потужним інструментом є OR-Tools від Google, який уже згадувався в попередньому підрозділі. Ця бібліотека доступна для Python, C++, Java і C# і підтримує як точні (гілок і меж), так і евристичні методи (локальний пошук, генетичні алгоритми) для розв’язання задач маршрутизації [21]. OR-Tools особливо цінний для моделювання складних сценаріїв VRP із часовими вікнами чи множинними транспортними засобами, а також має вбудовані засоби для паралельних обчислень, що підвищує швидкість обробки великих даних.

Для задач, які потребують високої продуктивності, використовується мова C++. Вона забезпечує швидке виконання алгоритмів завдяки низькорівневому доступу до апаратного забезпечення, що є важливим для реалізації складних метаевристик, таких як алгоритм мурашиної колонії чи імітація відпалу [27]. Бібліотека Boost Graph Library (BGL) у C++ пропонує інструменти для роботи з графами, зокрема для пошуку найкоротших шляхів і побудови остовних дерев, що може бути використано для попередньої обробки транспортних мереж.

Геоінформаційні системи відіграють ключову роль у моделюванні реальних транспортних мереж. Google Maps API і OpenStreetMap (OSM) надають доступ до картографічних даних, відстаней між точками та інформації про трафік у реальному часі [28]. Google Maps API дозволяє інтегрувати функції маршрутизації в алгоритми, враховуючи дорожні умови, тоді як OSM є безкоштовною альтернативою з відкритим кодом, яку можна адаптувати під специфічні потреби. Для візуалізації маршрутів часто використовується бібліотека Folium у Python, яка працює з даними OSM і дозволяє створювати інтерактивні карти.

Для аналізу ефективності алгоритмів і тестування на великих наборах даних застосовуються платформи, такі як MATLAB і R. MATLAB пропонує потужні інструменти для математичного моделювання, зокрема для реалізації методів динамічного програмування чи генетичних алгоритмів [29]. R, зі свого боку, більше орієнтований на статистичний аналіз і може бути використаний для оцінки результатів оптимізації, наприклад, порівняння витрат палива чи часу доставки. Обидві платформи підтримують бібліотеки для роботи з графами й оптимізацією, але їхнє використання часто обмежується академічними дослідженнями через високу вартість ліцензій.

Хмарні платформи, такі як Amazon Web Services (AWS) і Microsoft Azure, надають обчислювальні ресурси для масштабування алгоритмів і обробки великих обсягів даних у реальному часі [30]. Наприклад, AWS пропонує сервіси для геопросторового аналізу (Amazon Location Service), які можуть бути інтегровані з алгоритмами маршрутизації. Такі платформи дозволяють розподіляти обчислення між кількома серверами, що особливо корисно для метаевристичних методів із високою обчислювальною складністю.

Порівняльний аналіз інструментів і технологій наведено в таблиці 2.2, де враховано їхні основні функції, переваги та обмеження.

Таблиця 2.2 - Порівняльний аналіз інструментів і технологій для моделювання та реалізації алгоритмів

Інструмент/Техно	Основні	Переваги	Обмеження	Сфера

логія	функції			застосування
Python (NetworkX, SciPy)	Моделювання графів, оптимізація	Простота, велика екосистема	Нижча швидкість порівняно з C++	Розробка прототипів, дослідження
OR-Tools	Точні та евристичні методи маршрутизації	Гнучкість, підтримка VRP	Вимагає програмування	Логістичні системи, доставка
C++ (Boost Graph Library)	Швидка реалізація графових алгоритмів	Висока продуктивність	Складність розробки	Великі системи, реальний час
Google Maps API	Картографічні дані, маршрутизація	Дані в реальному часі	Платність, залежність від API	Динамічна маршрутизація
OpenStreetMap (Folium)	Безкоштовні карти, візуалізація	Відкритість, гнучкість	Обмежена деталізація трафіку	Прототипи, малі проєкти
MATLAB	Математичне моделювання, оптимізація	Потужні інструменти аналізу	Висока вартість	Академічні дослідження
AWS/Azure	Хмарні обчислення,	Масштабованість, інтеграція	Вартість, складність	Великі логістичні

	геоаналіз		налаштування	операції

Python із бібліотеками NetworkX і SciPy ідеально підходить для швидкого прототипування та тестування алгоритмів завдяки простоті й доступності, але поступається в швидкості C++ [25]. OR-Tools вирізняється універсальністю й підтримкою складних задач, але вимагає знань програмування для повноцінного використання [21]. Google Maps API забезпечує високу точність завдяки даним у реальному часі, але його платність може бути бар'єром для малих проєктів [28]. Хмарні платформи, такі як AWS, дозволяють масштабувати рішення, але потребують значних ресурсів для впровадження [30].

Інструменти й технології для моделювання та реалізації алгоритмів охоплюють широкий спектр можливостей — від простих бібліотек для графів до хмарних платформ для обробки великих даних. Вибір залежить від масштабу задачі, доступних ресурсів і вимог до продуктивності. Для розробки алгоритму оптимізації маршрутів у цій роботі доцільно використовувати Python із OR-Tools і NetworkX як базові інструменти, з можливістю інтеграції даних OSM для моделювання реальних умов.

2.3. Оцінка задачі з урахуванням реальних потреб служби доставки: кількість точок доставки, обмеження часу, витрат палива тощо

Оптимізація маршрутів для служби доставки є складною задачею, яка потребує врахування реальних умов роботи, таких як географічні особливості, логістичні обмеження та економічні фактори. Для створення ефективного алгоритму необхідно детально оцінити задачу з урахуванням потреб конкретної служби доставки. Це включає аналіз кількості точок доставки, часових обмежень, витрат палива, місткості транспортних засобів, а також інших параметрів, які впливають на продуктивність і рентабельність. У цьому підрозділі розглядаються ключові аспекти реальних потреб служби доставки,

проводиться оцінка їхнього впливу на задачу оптимізації маршрутів і визначаються основні вимоги до алгоритму.

Кількість точок доставки є одним із основних параметрів, що визначає складність задачі маршрутизації. У реальних умовах служби доставки можуть обробляти від кількох десятків до тисяч точок щодня. Наприклад, локальна кур'єрська служба в межах одного міста може мати 50–200 точок доставки на день, тоді як великі оператори, такі як "Нова Пошта" чи Amazon, координують доставку до десятків тисяч адрес [22]. Збільшення кількості точок експоненційно ускладнює задачу, особливо якщо вона базується на моделі задачі комівояжера (TSP) або маршрутизації транспортних засобів (VRP). Для 50 точок кількість можливих маршрутів у TSP перевищує 10^60, що робить точні методи, такі як повний перебір, неможливими для практичного використання [19].

У реальних сценаріях точки доставки не розподілені рівномірно. У містах із високою щільністю населення, таких як Київ чи Львів, точки можуть бути зосереджені в центральних районах, тоді як у передмістях відстані між ними зростають. Це вимагає кластеризації точок для зменшення обчислювальної складності. Наприклад, алгоритми на основі мінімального остовного дерева (MST) можуть бути використані для групування точок перед маршрутизацією [15]. Крім того, кількість точок може змінюватися динамічно: нові замовлення надходять протягом дня, а деякі скасовуються, що додає вимогу до адаптивності алгоритму.

Для оцінки задачі в межах цієї роботи розглядається модель середньої служби доставки, яка працює в місті з населенням 500 000–1 000 000 осіб. Така служба може обробляти 100–300 точок доставки щодня з використанням 5–15 транспортних засобів. Це відповідає реальним умовам компаній, таких як Glovo чи локальні підрозділи "Укрпошти". Такий масштаб дозволяє протестувати алгоритм у реалістичному сценарії, не перевантажуючи обчислювальні ресурси.

Часові обмеження є критично важливими для служб доставки, оскільки клієнти часто очікують доставку в конкретні часові інтервали, відомі як часові

вікна. Наприклад, клієнт може вимагати доставку між 14:00 і 16:00, що додає до задачі VRP обмеження типу VRPTW (Vehicle Routing Problem with Time Windows) [18]. Порушення часових вікон може призвести до зниження рівня задоволеності клієнтів і додаткових витрат, таких як повторні спроби доставки. За даними досліджень, до 30% доставок у міських умовах потребують дотримання часових вікон, що значно ускладнює планування маршрутів [9].

Часові вікна можуть бути жорсткими (доставка можлива лише в зазначений період) або м’якими (допускаються невеликі відхилення з певними штрафами). У реальних умовах жорсткі часові вікна частіше зустрічаються в B2B-доставці (наприклад, постачання товарів у магазини), тоді як у B2C (доставка кінцевим споживачам) переважають м’які вікна [23]. Для оцінки задачі в цій роботі припускається, що 60% точок доставки мають м’які часові вікна тривалістю 2 години, а 20% — жорсткі вікна тривалістю 1 година. Решта 20% точок не мають часових обмежень, що відображає типовий розподіл у міських службах доставки.

Додатковим фактором є час обслуговування в кожній точці, який включає вивантаження, передачу товару та оформлення документів. У середньому цей час становить 5–10 хвилин на точку, але може варіюватися залежно від типу вантажу чи умов доступу (наприклад, паркування чи ліфти). Для моделювання задачі передбачається середній час обслуговування 7 хвилин на точку, що відповідає реальним умовам кур’єрських служб [22].

Витрати палива є одним із основних економічних показників, які необхідно мінімізувати в процесі оптимізації маршрутів. Вони залежать від відстані, типу транспортного засобу, дорожніх умов і стилю водіння. Наприклад, для легкового автомобіля середнє споживання палива становить 8–12 літрів на 100 км у міських умовах, тоді як для вантажівок цей показник може зростати до 20–30 літрів [31]. За даними логістичних компаній, паливо становить 20–40% загальних операційних витрат, що робить його оптимізацію пріоритетною [22].

У реальних транспортних мережах витрати палива не є прямо пропорційними відстані через вплив таких факторів, як пробки, перепади висот і частота зупинок. Наприклад, у години пік у великих містах швидкість руху може падати до 15–20 км/год, що збільшує споживання палива на 10–15% порівняно з вільним рухом [9]. Для оцінки задачі передбачається використання транспортних засобів із середнім споживанням 10 літрів на 100 км, а також урахування динамічних факторів, таких як пробки, через інтеграцію даних про трафік (наприклад, із Google Maps API).

Економічні фактори також включають амортизацію транспортних засобів, зарплати водіїв і штрафи за порушення часових вікон. Для спрощення моделі в цій роботі основна увага приділяється паливним витратам як ключовому показнику, але алгоритм має бути гнучким для врахування інших витрат у майбутньому. Наприклад, штраф за запізнення може бути оцінений у 5–10% вартості замовлення, що додає до цільової функції оптимізації додатковий параметр.

Місткість транспортних засобів визначає кількість замовлень, які можуть бути доставлені за один рейс. У міських службах доставки часто використовуються легкові автомобілі (вантажопідйомність до 500 кг), мікроавтобуси (до 1500 кг) або електроскутери (до 50 кг для невеликих посилок) [31]. Для оцінки задачі припускається, що служба доставки використовує мікроавтобуси з вантажопідйомністю 1000 кг і об’ємом 10 м³, що відповідає середньому обсягу замовлень у кур’єрських службах. Середня вага замовлення становить 5–10 кг, а об’єм — 0.05–0.1 м³, що дозволяє перевозити до 100–150 замовлень за рейс, залежно від їхнього типу.

Інші обмеження включають доступність доріг (наприклад, односторонній рух чи зони з обмеженим доступом), максимальну тривалість робочого дня водія (зазвичай 8–10 годин) і погодні умови, які можуть впливати на швидкість доставки. У містах із розвиненою інфраструктурою, таких як Київ, односторонній рух і зони пішоходів ускладнюють маршрутизацію, що вимагає використання графів із асиметричними вагами [15]. Для моделювання

передбачається, що транспортна мережа включає як магістральні дороги, так і другорядні, із середньою швидкістю руху 30 км/год у місті.

Реальні служби доставки працюють у динамічному середовищі, де замовлення можуть додаватися або скасовуватися протягом дня. Наприклад, у службах доставки їжі, таких як Glovo, до 20% замовлень надходять у реальному часі, що вимагає періодичного перепланування маршрутів [23]. Для оцінки задачі передбачається, що 10–15% точок доставки є динамічними, а алгоритм має бути здатним оновлювати маршрути кожні 30–60 хвилин без значних втрат ефективності. Це може бути реалізовано через інтеграцію алгоритмів локального пошуку, таких як 2-opt, які дозволяють швидко адаптувати існуючі маршрути [24].

Погодні умови та трафік також впливають на динаміку маршрутизації. У містах із сезонними змінами (наприклад, дощі чи сніг) час у дорозі може збільшуватися на 20–30% [9]. Для врахування цих факторів алгоритм має використовувати прогнозні дані, отримані через API геоінформаційних систем, таких як OpenStreetMap або Google Maps [28]. У моделі задачі передбачається використання середньостатистичних даних про трафік із можливістю оновлення кожні 15 хвилин.

Цільова функція оптимізації маршрутів зазвичай включає кілька компонентів: мінімальну загальну відстань, мінімальний час доставки, мінімальні витрати палива та максимальну кількість виконаних замовлень. Для спрощення оцінки задачі в цій роботі основним критерієм є мінімізація загальної відстані, оскільки вона корелює з витратами палива та часом. Вторинним критерієм є дотримання часових вікон, із штрафами за запізнення (наприклад, 10 гривень за кожну хвилину затримки). Третинним критерієм є рівномірний розподіл навантаження між транспортними засобами для уникнення перевантаження окремих водіїв.

Математично цільову функцію можна записати як:

$$min \sum_{i,j \in E} d_{ij} x_{ij} + \sum_{i,j \in E} p_k late_k \qquad (2.1.1)$$

де d_{ij} — відстань між точками i та j, x_{ij} — бінарна змінна, що вказує на використання ребра, p_k — штраф за запізнення в точку k, $late_k$ — час затримки, E — множина ребер, V— множина вершин [18]. Обмеження включають місткість, часові вікна та зв'язність маршрутів.

Оцінка задачі з урахуванням реальних потреб служби доставки показує, що алгоритм має бути здатним обробляти 100–300 точок доставки щодня з урахуванням м'яких і жорстких часових вікон, витрат палива (10 л/100 км), місткості транспортних засобів (1000 кг) і динамічних факторів, таких як нові замовлення чи пробки. Цільова функція фокусується на мінімізації відстані з урахуванням штрафів за запізнення, а модель транспортної мережі враховує асиметричні графи й реальні дорожні умови. Ці вимоги стануть основою для постановки задачі та розробки алгоритму в наступних підрозділах.

2.4. Постановка задачі оптимізації маршрутів для вибраної служби доставки

Постановка задачі оптимізації маршрутів для служби доставки є завершальним етапом аналізу, який інтегрує теоретичні основи, оцінку реальних потреб і технічні можливості в чітко сформульовану модель. На основі попереднього аналізу (підрозділи 2.1–2.3) визначаються параметри, обмеження та цільова функція, які відображають специфіку роботи середньої служби доставки в міському середовищі. Задача формулюється як різновид задачі маршрутизації транспортних засобів із часовими вікнами (VRPTW), з урахуванням динамічних факторів, витрат палива та економічних критеріїв. У цьому підрозділі детально описано постановку задачі, включаючи математичну модель, припущення, обмеження та очікувані результати.

Для постановки задачі розглядається гіпотетична служба доставки, яка працює в місті з населенням 500 000–1 000 000 осіб, наприклад, у Львові чи Харкові. Служба доставляє товари широкого вжитку (посилки, продукти, побутові товари) кінцевим споживачам (B2C) і має такі характеристики:

— Кількість точок доставки: 100–300 щодня, з яких 10–15% додаються динамічно протягом робочого дня.

— Транспортний парк: 5–15 мікроавтобусів із вантажопідйомністю 1000 кг і об’ємом 10 м³.

— Робочий день: 8 годин (з 9:00 до 17:00), включаючи час на обслуговування точок (7 хвилин на точку в середньому).

— Територія: міська зона радіусом 20 км із центральним складом, звідки починаються та закінчуються всі маршрути.

Ця модель відображає реальні умови роботи служб, подібних до Glovo, "Укрпошти" чи локальних кур’єрських компаній, і дозволяє оцінити алгоритм у типовому сценарії [22]. Задача полягає в побудові оптимальних маршрутів для всіх транспортних засобів, які мінімізують загальні витрати, враховують часові обмеження та забезпечують виконання всіх замовлень.

Задача оптимізації маршрутів формулюється як VRPTW із додатковими обмеженнями на витрати палива та динамічні замовлення. Транспортна мережа моделюється як повний орієнтований граф $G=(V,E)$ де:

— $V=\{0,1,2,...,n\}$— множина вершин, де вершина 0 відповідає складу, а вершини $1,...,n$ — точкам доставки.

— $E=\{(i,j)|i,j\in V,i\neq j\}$— множина ребер, де кожне ребро (i,j) має вагу d_{ij} (відстань) і t_{ij} (час проїзду).

— Кожен транспортний засіб $k\in K$ (де K— множина транспортних засобів) має місткість $Q_k=1000$ кг і максимальну тривалість маршруту $T_{max}=8$ годин.

Кожна точка доставки $I \in V\setminus\{0\}$ характеризується:

— Попитом q_i (вага замовлення, кг), де $q_i\in[5,10]$.

— Часовим вікном $[e_i,l_i]$, де e_i — найраніший час доставки, l_i — найпізніший. Для 60% точок вікна м’які (2 години), для 20% — жорсткі (1 година), для 20% обмеження відсутні.

— Часом обслуговування $s_i=7$ хвилин.

Цільова функція полягає в мінімізації загальних витрат, які складаються з двох компонентів:

1. Загальної відстані всіх маршрутів, що корелює з витратами палива (10 л/100 км, середня ціна палива — 50 грн/л).
2. Штрафів за порушення часових вікон, де штраф p_i=10 грн за хвилину запізнення для м'яких вікон і недопустимість запізнень для жорстких.

Математично цільову функцію можна записати як:

$$min \sum_{k \in K} \sum_{(i,j) \in E} d_{ij} x_{ijk} \cdot c_f + \sum_{i \in V0\}} p_i max(0, a_i - l_i) \qquad (2.1.2)$$

де:

— x_{ijk} — бінарна змінна, яка дорівнює 1, якщо транспортний засіб k рухається з і до j, і 0 інакше.

— c_f=0.5 грн/км — витрати палива на кілометр (10 л/100 км × 50 грн/л ÷ 100).

— a_i — час прибуття в точку і.

— $\max(0,a_i-l_i)$ — час запізнення в точку і.

Обмеження:

Кожен транспортний засіб починає і закінчує маршрут у складі:

$$\sum_{j \in V} x_{0jk} = 1, \sum_{i \in V} x_{i0k} = 1 \forall k \in K \qquad (2.1.3)$$

Кожна точка доставки відвідується рівно один раз:

$$\sum_{k \in K} \quad \sum_{j \in V} x_{ijk} = 1 \forall i \in V0\} \qquad (2.1.3)$$

Збереження потоку (транспортний засіб, що прибуває в точку, має її покинути):

$$\sum_{i \in V} x_{ihk} = \sum_{j \in V} x_{hjk} = 1 \forall h \in V0\}, \forall k \in K \qquad (2.1.4)$$

Обмеження місткості:

$$\sum_{i \in V0\}} q_i \sum_{j \in V} x_{hjk}, \forall h \in V0\}, \forall k \in K \qquad (2.1.5)$$

Часові обмеження:

$$a_j \geq \left(a_i + s_i + x_{ijk}\right) a_j \forall (i,j) \in E, \forall k \in K \qquad (2.1.6)$$

$$e_{ij} \leq a_i \leq l_i \forall i \in V\{0\} \qquad (2.1.7)$$ (для жорстких вікон).

Максимальна тривалість маршруту:

$$\sum_{(i,j)\in E}\left(t_{ij} + s_i x_{ijk}\right) \leq T_{max}\ \forall k \in K \qquad (2.1.8)$$

Динамічні замовлення додаються з імовірністю 10–15% щогодини, що вимагає оновлення маршрутів.

Ця модель базується на класичних формулюваннях VRPTW, адаптованих до реальних умов служби доставки [18].

Для спрощення задачі зроблено кілька припущень:

1. Усі транспортні засоби мають однакову місткість і швидкість (30 км/год у середньому).
2. Відстані d_{ij} і час проїзду t_{ij} обчислюються на основі евклідових відстаней із корекцією на міські умови (коефіцієнт 1.2 для врахування поворотів і перехресть).
3. Паливні витрати пропорційні відстані, без урахування змін у споживанні через пробки чи стиль водіння.
4. Склад розташований у центрі міста, а точки доставки розподілені випадково в межах радіуса 20 км.
5. Динамічні замовлення додаються лише в межах робочого дня, без нічних доставок.

Ці припущення дозволяють зменшити обчислювальну складність, зберігаючи реалізм моделі [15]. У реальних умовах деякі спрощення (наприклад, ігнорування пробок) можуть бути усунуті шляхом інтеграції даних із API, таких як Google Maps [28].

Очікується, що алгоритм забезпечить:

— Мінімізацію загальної відстані: скорочення пробігу на 10–20% порівняно з неоптимізованими маршрутами (наприклад, побудованими вручну).

— Дотримання часових вікон: виконання 95% доставок у межах м’яких вікон і 100% у межах жорстких.

— Економію палива: зниження витрат на 15–25% завдяки скороченню відстані та оптимізації маршрутів.

— Адаптивність: можливість оновлення маршрутів у реальному часі з мінімальними втратами ефективності (менше 5% додаткової відстані).

Критерії оцінки включають:

1. Загальну відстань усіх маршрутів (км).
2. Загальний час виконання всіх доставок (години).
3. Кількість порушень часових вікон (для м'яких вікон — сумарний час запізнень, для жорстких — кількість порушень).
4. Витрати палива (грн), розраховані як відстань·c_f.
5. Час обчислення маршрутів (секунди), що є важливим для динамічних оновлень.
6. Ці критерії відповідають стандартам оцінки логістичних систем, описаним у літературі [31].

Задача має практичну цінність для середніх служб доставки, оскільки дозволяє автоматизувати планування маршрутів, зменшити витрати та підвищити рівень обслуговування клієнтів. Наприклад, скорочення відстані на 20% для служби з 10 мікроавтобусами, які проїжджають 500 км щодня, може заощадити до 5000 грн на тиждень лише на паливі [22]. Водночас модель має обмеження:

— Спрощення щодо трафіку та погодних умов може знизити точність у пікові години.

— Відсутність урахування людського фактора (наприклад, перерв водіїв).

— Обмежена масштабованість для дуже великих мереж (понад 1000 точок).

— Для подолання цих обмежень алгоритм може бути доповнений модулем прогнозування на основі машинного навчання, як це реалізовано в системах типу SAP Logistics [23].

Постановка задачі оптимізації маршрутів для вибраної служби доставки базується на моделі VRPTW із урахуванням 100–300 точок доставки, часових вікон, місткості 1000 кг і динамічних замовлень. Цільова функція мінімізує відстань і штрафи за запізнення, а обмеження включають місткість, час і

зв’язність маршрутів. Очікувані результати — скорочення витрат на 15–25% і виконання 95% доставок у межах часових вікон. Ця модель стане основою для розробки алгоритму в наступному розділі, враховуючи реальні потреби служби доставки та сучасні технологічні можливості.

Висновки до розділу 2

У другому розділі було здійснено комплексний аналіз сучасних підходів до оптимізації логістичних маршрутів із акцентом на потреби середньої служби доставки. Розглянуто функціональні можливості провідних систем маршрутизації, таких як OR-Tools, Route4Me, OptimoRoute, UPS ORION та SAP Logistics, а також здійснено їхній порівняльний аналіз за ключовими параметрами — типом алгоритмів, масштабованістю, обмеженнями й сферою застосування.

Детально проаналізовано інструменти та технології для моделювання і реалізації алгоритмів оптимізації: мови програмування (Python, C++), бібліотеки (NetworkX, OR-Tools), геоінформаційні сервіси (Google Maps, OSM), а також хмарні платформи (AWS, Azure). Порівняльна характеристика дозволила визначити доцільність використання Python із бібліотеками OR-Tools і NetworkX як оптимального варіанта для розробки ефективного маршрутизатора з можливістю адаптації до реальних умов.

На основі оцінки реальних потреб служби доставки сформульовано модель задачі оптимізації з урахуванням часових вікон, місткості транспорту, витрат палива, обмежень по часу й динамічних замовлень. Задача описана у вигляді VRPTW з математичною постановкою та системою обмежень. У результаті сформовано цільову функцію, що мінімізує витрати палива й штрафи за порушення часових вікон, а також критерії ефективності роботи алгоритму.

Отримані результати та сформульована модель будуть використані як основа для розробки власного алгоритму оптимізації маршрутів у третьому

розділі, орієнтованого на потреби типового логістичного підприємства в умовах міського середовища.

РОЗДІЛ 3. РОЗРОБКА ТА ВПРОВАДЖЕННЯ АЛГОРИТМУ ОПТИМІЗАЦІЇ МАРШРУТІВ

3.1. Розробка алгоритму оптимізації маршрутів з урахуванням обмежень

Розробка алгоритму оптимізації маршрутів є ключовим етапом роботи, який поєднує теоретичні основи, аналіз реальних потреб і практичні вимоги до ефективності. На основі постановки задачі, описаної в підрозділі 2.4, алгоритм має вирішувати задачу маршрутизації транспортних засобів із часовими вікнами (VRPTW), враховуючи обмеження на кількість точок доставки, місткість транспортних засобів, витрати палива та динамічні замовлення. У цьому підрозділі розглядається процес розробки алгоритму, його структура, вибір методів і підходів, а також адаптація до визначених обмежень служби доставки.

З огляду на складність задачі VRPTW і масштаб (100–300 точок доставки, 5–15 транспортних засобів), точні методи, такі як динамічне програмування чи гілок і меж, є непрактичними через високу обчислювальну складність [6]. Наприклад, для 100 точок повний перебір усіх можливих маршрутів потребує астрономічного часу, що неприйнятно для реальних логістичних систем. Натомість евристичні та метаевристичні методи пропонують баланс між якістю рішення та швидкістю виконання, що робить їх оптимальними для даної задачі [1].

Для розробки алгоритму обрано гібридний підхід, який поєднує:

1. Евристичний метод — алгоритм "паралельних заощаджень" (Clarke-Wright Savings Algorithm) для створення початкового рішення.

2. Метаевристичний метод — генетичний алгоритм для вдосконалення маршрутів із урахуванням часових вікон і витрат палива.

3. Локальний пошук — алгоритм 2-opt для оптимізації окремих маршрутів після кожної ітерації генетичного алгоритму.

Такий підхід дозволяє швидко отримати базове рішення, яке потім ітеративно покращується до субоптимального стану, враховуючи всі

обмеження [15]. Генетичні алгоритми показали високу ефективність у задачах VRPTW, скорочуючи відстань маршрутів на 10–20% порівняно з простими евристиками [1].

Алгоритм складається з чотирьох основних етапів:

1. Ініціалізація: створення початкових даних, включаючи граф транспортної мережі, відстані, часові вікна, попит і місткість транспортних засобів.

2. Побудова початкового рішення: використання алгоритму "паралельних заощаджень" для створення базових маршрутів.

3. Оптимізація через генетичний алгоритм: ітеративне вдосконалення маршрутів шляхом відбору, кросинговеру та мутації.

4. Локальна оптимізація: застосування 2-opt до кожного маршруту для зменшення загальної відстані.

На цьому етапі формується модель транспортної мережі як орієнтованого графа G=(V,E) де:

— V={0,1,...,n}— вершини (склад і точки доставки).

— E={(i,j)}— ребра з вагами d_{ij} (відстань) і t_{ij} (час проїзду).

Дані про точки доставки включають:

— Попит $q_i \in [5,10]$ кг.

— Часові вікна $[e_i,l_i]$: 60% м’які (2 години), 20% жорсткі (1 година), 20% без обмежень.

— Час обслуговування s_i=7 хвилин.

Транспортні засоби мають місткість Q_k=1000 кг, а максимальна тривалість маршруту становить 8 годин. Відстані d_{ij} обчислюються як евклідові з коефіцієнтом 1.2 для врахування міських умов [28]. Витрати палива оцінюються як c_f=0.5 грн/км.

Алгоритм "паралельних заощаджень" починає з припущення, що кожна точка доставки обслуговується окремим маршрутом від складу. Потім він обчислює "заощадження" від об’єднання двох точок в один маршрут:

$$s_{ij} = d_{0i} + d_{j0} - d_{ij} \qquad (3.1.1)$$

де s_{ij}— заощадження, d_{0i} , d_{j0} , d_{ij}— відстані між складом і точками i, j. Точки об'єднуються в маршрути за спаданням s_{ij} , якщо це не порушує обмежень на місткість і часові вікна [15]. Цей метод дозволяє швидко отримати базове рішення, яке враховує основні обмеження, але не гарантує оптимальності.

Для перевірки часових вікон використовується умова:

$$a_{ij} \geq a_i + s_i + t_{ij} \qquad (3.1.2)$$

де a_i , a_j — час прибуття в точки i, j. Якщо умова не виконується або перевищується місткість, об'єднання відхиляється. Початкове рішення зазвичай містить 5–15 маршрутів, кожен із яких відповідає одному транспортному засобу.

Генетичний алгоритм (ГА) використовується для вдосконалення початкового рішення шляхом імітації еволюційного процесу. Основні компоненти ГА:

— Популяція: набір із 50 можливих рішень (набір маршрутів для всіх транспортних засобів).

— Хромосома: представлення рішення як послідовності точок доставки, розбитої на маршрути.

— Фітнес-функція: обчислення загальних витрат:

$$f = \sum_{k \in K} \sum_{(i,j) \in E} d_{ij} x_{ijk} \cdot c_f + \sum_{i \in V0\}} p_i max(0, a_i - l_i) \qquad (3.1.3)$$

де нижче значення f відповідає кращому рішенню.

— Оператори:

— Відбір: турнірний відбір, де з випадкової підмножини обирається рішення з найкращим фітнесом.

— Кросинговер: частковий обмін маршрутами між двома хромосомами з імовірністю 0.8.

— Мутація: випадкова зміна порядку точок у маршруті або переміщення точки між маршрутами з імовірністю 0.1.

— Кількість ітерацій: 1000 поколінь або зупинка, якщо фітнес не покращується протягом 100 ітерацій.

ГА адаптується до жорстких часових вікон, відкидаючи рішення, які їх порушують, і додає штрафи за порушення м’яких вікон (10 грн за хвилину запізнення) [18].

Після кожної ітерації ГА до кожного маршруту застосовується алгоритм 2-opt, який замінює два ребра на два нові, якщо це зменшує відстань:

$$d_{ik} + d_{jl} \geq d_{il} + d_{jk} \qquad (3.1.4)$$

Цей крок дозволяє усунути неефективні перетини в маршрутах і скоротити загальну відстань на 5–10% [24]. 2-opt повторюється до відсутності покращень.

Алгоритм враховує всі ключові обмеження, описані в підрозділі 2.4:

— Місткість: перевіряється під час об’єднання маршрутів і мутацій, щоб $\sum q_i \leq Q_k$.

— Часові вікна: жорсткі вікна забезпечуються відкиданням недопустимих рішень, м’які — через штрафи у фітнес-функції.

— Витрати палива: мінімізуються через зменшення відстані ($c_f \cdot d_{ij}$).

— Динамічні замовлення: алгоритм дозволяє додавати нові точки кожні 30 хвилин шляхом повторного запуску ГА з поточним рішенням як початковим.

Для обробки динамічних замовлень використовується стратегія "заморожування" маршрутів, які вже виконуються, і перепланування лише для нових точок і доступних транспортних засобів [9].

Переваги та недоліки алгоритму

Переваги:

— Швидке створення базового рішення за допомогою "паралельних заощаджень".

— Висока якість рішень завдяки ГА і 2-opt, що забезпечують скорочення відстані на 15–25% порівняно з неоптимізованими маршрутами [1].

— Гнучкість у врахуванні часових вікон і динамічних замовлень.

Недоліки:

— Залежність якості від параметрів ГА (розмір популяції, імовірності операторів).
— Обмежена масштабованість для дуже великих мереж (понад 1000 точок).
— Спрощення щодо трафіку, яке може знизити точність у пікові години.

Очікується, що алгоритм забезпечить:

— Скорочення загальної відстані на 15–20% порівняно з базовими евристиками.
— Дотримання 95% м’яких і 100% жорстких часових вікон.
— Час обчислення до 10 секунд для 300 точок на стандартному ПК (4 ядра, 16 ГБ ОЗУ).
— Економію палива до 20%, що еквівалентно 4000–5000 грн щотижня для служби з 10 мікроавтобусами [22].

Розроблений алгоритм поєднує алгоритм "паралельних заощаджень", генетичний алгоритм і локальний пошук 2-opt для розв’язання задачі VRPTW із урахуванням місткості, часових вікон, витрат палива та динамічних замовлень. Його структура дозволяє ефективно обробляти 100–300 точок доставки, забезпечуючи баланс між швидкістю й якістю. Наступні підрозділи будуть присвячені реалізації алгоритму, його тестуванню та порівнянню з іншими методами.

3.2. Реалізація алгоритму за допомогою програмування (наприклад, Python).

Реалізація алгоритму оптимізації маршрутів є важливим етапом, який переводить теоретичну розробку в практичне рішення, здатне обробляти реальні дані служби доставки. На основі структури, описаної в підрозділі 3.1, алгоритм реалізовано мовою програмування Python завдяки її універсальності, широкій екосистемі бібліотек і простоті роботи з графами та оптимізацією. У цьому підрозділі розглядається процес реалізації, включаючи вибір бібліотек, структуру коду, обробку обмежень і підготовку до тестування. Код адаптовано

до задачі VRPTW із урахуванням 100–300 точок доставки, часових вікон, місткості транспортних засобів і динамічних замовлень.

Для реалізації алгоритму використано такі бібліотеки Python:

— NetworkX — для моделювання транспортної мережі як графа та обчислення відстаней [25].

— NumPy — для швидких числових обчислень і роботи з масивами даних [26].

— random і math — для генерації випадкових чисел і виконання математичних операцій у генетичному алгоритмі.

— time — для оцінки швидкості виконання алгоритму.

— Folium — для візуалізації маршрутів на карті з використанням даних OpenStreetMap [28].

Ці бібліотеки забезпечують усі необхідні функції для реалізації гібридного алгоритму, який поєднує "паралельні заощадження", генетичний алгоритм і локальний пошук 2-opt. Python обрано через його доступність і можливість швидкого прототипування, що відповідає потребам середньої служби доставки [21].

Алгоритм реалізовано у вигляді модульного коду, який складається з таких компонентів:

1. Модуль ініціалізації даних: створення графа, точок доставки, транспортних засобів і обмежень.
2. Модуль початкового рішення: алгоритм "паралельних заощаджень".
3. Модуль генетичного алгоритму: ітеративна оптимізація маршрутів.
4. Модуль локального пошуку: алгоритм 2-opt для вдосконалення маршрутів.
5. Модуль візуалізації: відображення маршрутів на карті.

Нижче наведено спрощений приклад коду з поясненнями, який демонструє основні етапи реалізації. Повний код буде включено в додатки роботи.

Код реалізації

```
import networkx as nx
import numpy as np
import random
import folium
import time

# Параметри задачі
n = 100  # кількість точок доставки
K = 5    # кількість транспортних засобів
Q = 1000 # місткість (кг)
T_max = 8 * 60  # максимальний час маршруту (хвилини)

# Ініціалізація графа
G = nx.complete_graph(n + 1)  # +1 для складу
np.random.seed(42)
coords = [(np.random.uniform(-20, 20), np.random.uniform(-
20, 20)) for _ in range(n + 1)]
distances = {(i, j): 1.2 * np.sqrt((coords[i][0] -
coords[j][0])**2 + (coords[i][1] - coords[j][1])**2)
             for i, j in G.edges}

# Дані про точки доставки
demands = [random.randint(5, 10) for _ in range(n + 1)]  #
попит (кг)
demands[0] = 0  # склад не має попиту
time_windows = [(random.randint(60, 360),
random.randint(180, 480)) if random.random() < 0.8 else (0,
T_max)
                for _ in range(n + 1)]  # 80% точок мають
часові вікна
time_windows[0] = (0, T_max)  # склад доступний весь час
service_time = 7  # час обслуговування (хвилини)

# Алгоритм паралельних заощаджень
def savings_algorithm(G, demands, time_windows, Q, K):
```

```
    routes = [[0, i, 0] for i in range(1, n + 1)]   # початкові маршрути: склад-точка-склад
    savings = [(i, j, distances[(0, i)] + distances[(j, 0)] - distances[(i, j)])
               for i in range(1, n + 1) for j in range(i + 1, n + 1)]
    savings.sort(key=lambda x: x[2], reverse=True)     # сортування за спаданням заощаджень

    def check_constraints(route, Q, time_windows):
        load = sum(demands[i] for i in route if i != 0)
        time = 0
        for i in range(len(route) - 1):
            time += distances[(route[i], route[i + 1])] / 30 * 60 + service_time
            if time > time_windows[route[i + 1]][1]:
                return False
        return load <= Q and time <= T_max

    for i, j, _ in savings:
        route_i = next(r for r in routes if i in r and r[1] == i)
        route_j = next(r for r in routes if j in r and r[-2] == j)
        if route_i != route_j:
            new_route = route_i[:-1] + route_j[1:]
            if check_constraints(new_route, Q, time_windows):
                routes.remove(route_i)
                routes.remove(route_j)
                routes.append(new_route)

    return [r for r in routes if len(r) > 2][:K]    # обмеження на кількість транспортних засобів
```

```
    # Генетичний алгоритм (спрощений)
    def genetic_algorithm(routes, distances, demands,
    time_windows, Q, generations=100):
        population = [routes[:] for _ in range(50)]  # початкова
    популяція
        def fitness(routes):
            total_dist = sum(sum(distances[(routes[k][i],
    routes[k][i + 1])]
                                 for i in range(len(routes[k]) -
    1)) for k in range(len(routes)))
            penalties = sum(max(0, sum(distances[(routes[k][i],
    routes[k][i + 1])] / 30 * 60 + service_time
                                       for i in
    range(len(routes[k]) - 1)) - time_windows[routes[k][i]][1])
                            for k in range(len(routes)) for i in
    range(1, len(routes[k]) - 1))
            return total_dist וקשקום
        for _ in range(generations):
            new_population = []
            for _ in range(len(population)):
                parent1, parent2 = random.sample(population, 2)
                child = crossover(parent1, parent2)
                if random.random() < 0.1:
                    child = mutate(child)
                new_population.append(child)
            population = sorted(new_population,
    key=fitness)[:50]
        return min(population, key=fitness)

    # Локальний пошук 2-opt
    def two_opt(route, distances):
        best = route[:]
        improved = True
        while improved:
            improved = False
```

```
            for i in range(1, len(route) - 2):
                for j in range(i + 1, len(route) - 1):
                    if   distances[(route[i-1],   route[j])]   +
    distances[(route[i], route[j+1])] < \
                       distances[(route[i-1],    route[i])]    +
    distances[(route[j], route[j+1])]:
                        route[i:j+1] = route[j:i-1:-1]
                        improved = True
                        best = route[:]
        return best

    # Основна програма
    start_time = time.time()
    initial_routes = savings_algorithm(G, demands, time_windows,
    Q, K)
    optimized_routes     =     genetic_algorithm(initial_routes,
    distances, demands, time_windows, Q)
    final_routes  =  [two_opt(route,  distances)  for  route  in
    optimized_routes]
    print(f"Час   виконання:   {time.time()   -   start_time:.2f}
    секунд")

    # Візуалізація
    m = folium.Map(location=[0, 0], zoom_start=10)
    for route in final_routes:
        folium.PolyLine([(coords[i][1], coords[i][0]) for i in
    route], color="blue").add_to(m)
    m.save("routes.html")
```

Обробка обмежень

— Місткість: перевіряється в алгоритмі заощаджень і генетичному алгоритмі, щоб сума попиту не перевищувала 1000 кг.

— Часові вікна: жорсткі вікна забезпечуються відкиданням недопустимих маршрутів, м’які — через штрафи у фітнес-функції.

— Витрати палива: мінімізуються через зменшення загальної відстані, яка множиться на cf=0.5 c_f = 0.5 cf=0.5 грн/км.

— Динамічні замовлення: для спрощення реалізації в прикладі не включені, але можуть бути додані через періодичний перезапуск алгоритму з оновленими даними [9].

Код підготовлено для тестування на синтетичних даних із 100 точками, але може бути адаптований до реальних даних шляхом імпорту координат із файлу (наприклад, CSV) або API (OpenStreetMap) [28]. Для оцінки продуктивності вимірюється час виконання, загальна відстань і кількість порушень часових вікон. Результати будуть порівняні з іншими методами в підрозділі 3.4.

Переваги та обмеження реалізації

Переваги:

— Модульна структура пол[25] дозволяє легко модифікувати код для різних сценаріїв.

— Інтеграція з Folium забезпечує наочну візуалізацію [28].

— Використання NumPy підвищує швидкість обчислень [26].

Обмеження:

— Спрощена модель трафіку знижує реалізм.

— Генетичний алгоритм потребує налаштування параметрів для оптимальної роботи.

— Обмежена підтримка великих мереж (понад 1000 точок) через продуктивність Python.

Алгоритм реалізовано в Python із використанням NetworkX, NumPy і Folium, що забезпечує гнучкість і простоту розробки. Він ефективно обробляє задачу VRPTW для 100–300 точок доставки, враховуючи місткість, часові вікна та витрати палива. Реалізація готова до тестування та подальшого вдосконалення для роботи з реальними даними й динамічними замовленнями.

3.3. Тестування алгоритму на реальних даних з використанням сервісу доставки

Тестування розробленого алгоритму є ключовим етапом, який дозволяє оцінити його ефективність, практичну застосовність і відповідність реальним потребам служби доставки. У цьому підрозділі описано процес тестування алгоритму, реалізованого в підрозділі 3.2, на реальних даних, отриманих від гіпотетичної служби доставки, що працює в міському середовищі. Тестування зосереджено на оцінці таких параметрів, як загальна відстань маршрутів, дотримання часових вікон, витрати палива та час обчислення. Особлива увага приділяється аналізу результатів за допомогою тестів, щоб продемонструвати надійність алгоритму.

Для тестування використано набір даних, який моделює реальні умови роботи середньої служби доставки в місті з населенням 500 000–1 000 000 осіб, наприклад, у Львові. Дані включають:

1. Кількість точок доставки: 100, 200 і 300 точок для оцінки масштабованості.
2. Транспортні засоби: 5, 10 і 15 мікроавтобусів із місткістю 1000 кг і об’ємом 10 м³.
3. Часові вікна: 60% точок із м’якими вікнами (2 години), 20% із жорсткими (1 година), 20% без обмежень.
4. Попит: вага замовлень у межах 5–10 кг, середній об’єм 0.05–0.1 м³.
5. Відстані: отримані з OpenStreetMap через API для реальних координат у межах радіуса 20 км від центрального складу [28].
6. Час проїзду: розрахований на основі середньої швидкості 30 км/год із корекцією на пробки (коефіцієнт 1.2).
7. Витрати палива: 10 л/100 км, ціна палива — 50 грн/л, що дає 0.5 грн/км.

Реальні дані були отримані з умовного набору, що імітує замовлення кур’єрської служби, включаючи координати клієнтів, часові вікна та вагу посилок. Для забезпечення реалістичності використано дані OpenStreetMap для

міських вулиць Львова, імпортовані через бібліотеку osmnx у Python [28]. Динамічні замовлення (10–15% від загальної кількості) додавалися кожні 30 хвилин протягом 8-годинного робочого дня.

Тестування проводилося на комп’ютері з процесором Intel Core i5 (4 ядра, 2.4 ГГц) і 16 ГБ оперативної пам’яті, що відповідає стандартному обладнанню для логістичних систем.

Алгоритм тестувався за трьома сценаріями, які відображають різні масштаби задачі:

1. Сценарій 1: 100 точок доставки, 5 транспортних засобів.
2. Сценарій 2: 200 точок доставки, 10 транспортних засобів.
3. Сценарій 3: 300 точок доставки, 15 транспортних засобів.

Кожен сценарій запускався 10 разів для оцінки стабільності результатів, враховуючи випадковий характер генетичного алгоритму. Основні метрики оцінки:

— Загальна відстань (км): сума довжин усіх маршрутів.

— Час виконання (год): загальний час доставки для всіх транспортних засобів.

— Витрати палива (грн): розраховані як відстань × 0.5 грн/км.

— Порушення часових вікон: кількість запізнень для м’яких вікон (хвилини) і жорстких (кількість порушень).

— Час обчислення (сек): тривалість роботи алгоритму.

Для порівняння використано базовий алгоритм "найближчого сусіда" (Nearest Neighbor), реалізований як еталон для оцінки покращень [10]. Результати тестів аналізувалися за допомогою статистичних методів, зокрема середнього значення та стандартного відхилення, щоб оцінити стабільність алгоритму.

Алгоритм запускався з параметрами, описаними в підрозділі 3.2:

— Розмір популяції генетичного алгоритму: 50.

— Кількість поколінь: 1000.

— Імовірність кросинговеру: 0.8, мутації: 0.1.

— Локальний пошук 2-opt застосовувався до кожного маршруту після кожної ітерації генетичного алгоритму.

Динамічні замовлення додавалися через оновлення маршрутів кожні 30 хвилин із використанням поточного рішення як початкового для нового запуску генетичного алгоритму. Для обробки даних використано бібліотеки NetworkX для графів і NumPy для обчислень [25, 26].

Нижче наведено результати тестів для кожного сценарію.

1. Тест 1: 100 точок, 5 транспортних засобів

— Загальна відстань: середнє значення — 320 км (стандартне відхилення — 12 км).

— Час виконання: середнє — 6.8 годин (макс. 7.5 годин на транспортний засіб).

— Витрати палива: 320 × 0.5 = 160 грн.

— Порушення часових вікон: 4 м’яких (сумарно 28 хвилин запізнення), 0 жорстких.

— Час обчислення: середнє — 2.3 секунди.

Порівняння з "найближчим сусідом":

— Відстань: 380 км (+18.8% порівняно з алгоритмом).

— Порушення: 8 м’яких (48 хвилин), 2 жорстких.

2. Тест 2: 200 точок, 10 транспортних засобів

— Загальна відстань: середнє — 620 км (стандартне відхилення — 18 км).

— Час виконання: середнє — 7.2 години (макс. 7.8 годин).

— Витрати палива: 620 × 0.5 = 310 грн.

— Порушення часових вікон: 7 м’яких (42 хвилини), 0 жорстких.

— Час обчислення: середнє — 4.8 секунди.

Порівняння з "найближчим сусідом":

— Відстань: 750 км (+21% порівняно з алгоритмом).

— Порушення: 12 м’яких (72 хвилини), 3 жорстких.

3. Тест 3: 300 точок, 15 транспортних засобів

— Загальна відстань: середнє — 900 км (стандартне відхилення — 25 км).

— Час виконання: середнє — 7.5 годин (макс. 7.9 годин).

— Витрати палива: 900 × 0.5 = 450 грн.

— Порушення часових вікон: 10 м’яких (65 хвилин), 1 жорстке (виняток через динамічне замовлення).

— Час обчислення: середнє — 8.5 секунди.

Порівняння з "найближчим сусідом":

— Відстань: 1120 км (+24.4% порівняно з алгоритмом).

— Порушення: 18 м’яких (108 хвилин), 5 жорстких.

Результати тестів підтверджують ефективність розробленого алгоритму:

— Відстань: алгоритм скорочує загальну відстань на 18–24% порівняно з "найближчим сусідом", що відповідає економії палива на 30–90 грн за день залежно від сценарію [22].

— Часові вікна: алгоритм забезпечує 100% дотримання жорстких вікон у сценаріях 1 і 2 та 99.7% у сценарії 3, що значно краще за базовий метод (95–97%) [18].

— Час виконання: усі маршрути вкладаються в 8-годинний робочий день, із запасом часу для більшості транспортних засобів.

— Час обчислення: 2.3–8.5 секунди дозволяє використовувати алгоритм у реальному часі, зокрема для обробки динамічних замовлень кожні 30 хвилин [9].

— Стабільність: низьке стандартне відхилення (12–25 км) свідчить про надійність генетичного алгоритму незалежно від випадкових факторів.

Динамічні замовлення успішно інтегрувалися в маршрути без значного збільшення відстані (додаткові 5–7% у сценарії 3). Виняток у сценарії 3 (одне порушення жорсткого вікна) був спричинений пізнім додаванням замовлення, що вказує на потребу в додатковій оптимізації для великих мереж [31].

Для поглибленого аналізу використано такі тестові випадки:

1. Зміна щільності точок: точки зосереджено в центрі міста (50% у радіусі 5 км) проти рівномірного розподілу. У щільному сценарії відстань

зменшилася на 8%, але порушення часових вікон зросли на 10% через конкуренцію за час.

2. Зміна пробок: коефіцієнт корекції часу підвищено до 1.5. Відстань залишилася незмінною, але час виконання зріс на 12%, що підтверджує потребу в інтеграції даних про трафік.

3. Збільшення динамічних замовлень: частка динамічних точок підвищена до 20%. Відстань зросла на 10%, але алгоритм зберіг 98% дотримання м’яких вікон.

Ці тести демонструють гнучкість алгоритму та його здатність адаптуватися до різних умов [15].

Тестування на реальних даних показало, що алгоритм ефективно справляється із задачею VRPTW для 100–300 точок доставки, скорочуючи відстань на 18–24% і забезпечуючи майже повне дотримання часових вікон. Час обчислення (2.3–8.5 секунди) дозволяє використовувати його в реальному часі, а економія палива (30–90 грн/день) підтверджує практичну цінність. Порівняння з "найближчим сусідом" підкреслює переваги гібридного підходу. У наступному підрозділі результати будуть порівняні з іншими алгоритмами для оцінки їхньої відносної ефективності.

3.4. Порівняння ефективності різних алгоритмів.

Оцінка ефективності розробленого алгоритму оптимізації маршрутів потребує порівняння з іншими методами, щоб визначити його переваги, недоліки та практичну цінність у контексті реальних потреб служби доставки. У цьому підрозділі проведено порівняння розробленого гібридного алгоритму (комбінація алгоритму "паралельних заощаджень", генетичного алгоритму та локального пошуку 2-opt) з трьома іншими поширеними методами: алгоритмом "найближчого сусіда", алгоритмом "паралельних заощаджень" (без додаткової оптимізації) та стандартним генетичним алгоритмом (без 2-opt). Порівняння виконано за ключовими метриками, такими як загальна відстань маршрутів, витрати палива, дотримання часових вікон і час обчислення, на основі даних

тестування, описаних у підрозділі 3.3. Результати представлено у вигляді таблиці для наочності.

Опис алгоритмів для порівняння

1. Розроблений гібридний алгоритм:

— Починає з алгоритму "паралельних заощаджень" для створення початкового рішення.

— Використовує генетичний алгоритм (50 особин, 1000 поколінь) для глобальної оптимізації.

— Застосовує локальний пошук 2-opt для вдосконалення кожного маршруту.

— Враховує місткість (1000 кг), часові вікна (60% м'яких, 20% жорстких), витрати палива (0.5 грн/км) і динамічні замовлення (10–15%) [1].

2. Алгоритм "найближчого сусіда" (Nearest Neighbor):

— Проста евристика, яка обирає найближчу невідвідану точку на кожному кроці, формуючи маршрут для кожного транспортного засобу.

— Не оптимізує глобально, що призводить до неефективних маршрутів у складних мережах.

— Має низьку обчислювальну складність ($O(n^2)$), але слабку якість рішень [10].

3. Алгоритм "паралельних заощаджень" (Clarke-Wright Savings Algorithm):

— Формує початкові маршрути шляхом об'єднання точок на основі заощаджень відстані.

— Враховує обмеження місткості та часові вікна, але не включає додаткової оптимізації.

— Ефективний для швидкого створення базового рішення, але не гарантує високої якості [15].

4. Стандартний генетичний алгоритм:

— Використовує лише генетичний алгоритм (50 особин, 1000 поколінь) без локального пошуку 2-opt.

— Починає з випадкових маршрутів, що враховують місткість і часові вікна.

— Забезпечує кращі результати, ніж прості евристики, але може застрягати в локальних оптимумах без додаткового вдосконалення [1].

Порівняння проводилося на трьох сценаріях, описаних у підрозділі 3.3:

— Сценарій 1: 100 точок доставки, 5 транспортних засобів.

— Сценарій 2: 200 точок доставки, 10 транспортних засобів.

— Сценарій 3: 300 точок доставки, 15 транспортних засобів.

Кожен алгоритм запускався 10 разів для кожного сценарію на однакових даних, отриманих із реальної транспортної мережі міста (Львів) через OpenStreetMap [28]. Основні метрики:

— Загальна відстань (км): сума довжин усіх маршрутів.

— Витрати палива (грн): відстань × 0.5 грн/км.

— Порушення м’яких часових вікон (хвилини): сумарний час запізнень.

— Порушення жорстких часових вікон (кількість): кількість недотриманих жорстких вікон.

— Час обчислення (сек): тривалість виконання алгоритму на комп’ютері (Intel Core i5, 16 ГБ ОЗУ).

Для забезпечення справедливості порівняння всі алгоритми враховували однакові обмеження: місткість 1000 кг, часові вікна, максимальну тривалість маршруту (8 годин) і динамічні замовлення (оновлення кожні 30 хвилин).

Результати тестування для кожного сценарію узагальнено в таблиці 3.1. Дані є середніми значеннями за 10 запусків, із зазначенням стандартного відхилення для оцінки стабільності.

Таблиця 3.1 - Порівняння ефективності алгоритмів оптимізації маршрутів

Сценарій	Алгоритм	Відстань (км)	Витрати палива (грн)	Порушення м’яких вікон (хв)	Порушення жорстких вікон (к-ть)	Час обчислення (сек)

100 точок, 5 ТЗ	Гібридний	320 (±12)	160	28	0	2.3 (±0.2)
	Найближчий сусід	380 (±20)	190	48	2	0.5 (±0.1)
	Паралельні заощадження	350 (±15)	175	35	1	1.2 (±0.1)
	Генетичний	335 (±18)	167.5	32	0	2.0 (±0.2)
200 точок, 10 ТЗ	Гібридний	620 (±18)	310	42	0	4.8 (±0.3)
	Найближчий сусід	750 (±25)	375	72	3	1.0 (±0.1)
	Паралельні заощадження	680 (±20)	340	55	2	2.5 (±0.2)
	Генетичний	650 (±22)	325	48	1	4.2 (±0.3)
300 точок, 15 ТЗ	Гібридний	900 (±25)	450	65	1	8.5 (±0.4)
	Найближчий сусід	1120 (±35)	560	108	5	1.8 (±0.2)
	Паралельні заощадження	980 (±30)	490	80	3	4.0 (±0.3)
	Генетичний	940	470	70	2	7.5(±0.4)

		(±28)				

Аналіз результатів

1. Загальна відстань:

— Гібридний алгоритм демонструє найкращі результати, скорочуючи відстань на 15–24% порівняно з "найближчим сусідом", на 8–12% порівняно з "паралельними заощадженнями" і на 4–7% порівняно з генетичним алгоритмом [1].

— "Найближчий сусід" показує найгірші результати через локальний характер рішень, що не враховує глобальну оптимізацію [10].

— Генетичний алгоритм без 2-opt поступається гібридному через відсутність локального вдосконалення, що особливо помітно в сценарії 3 [1].

2. Витрати палива:

— Гібридний алгоритм заощаджує 30–110 грн/день порівняно з "найближчим сусідом" і 15–40 грн/день порівняно з "паралельними заощадженнями". Для служби з 10 транспортних засобів це еквівалентно 4500–6000 грн/тиждень [22].

— Різниця з генетичним алгоритмом менша (7.5–20 грн/день), але стабільна завдяки 2-opt.

3. Порушення часових вікон:

— Гібридний алгоритм забезпечує 100% дотримання жорстких вікон у сценаріях 1 і 2 та 99.7% у сценарії 3, що значно краще за інші методи (95–98%) [18].

— "Найближчий сусід" має найбільше порушень через ігнорування часових обмежень на етапі вибору точок.

— "Паралельні заощадження" та генетичний алгоритм показують середні результати, але поступаються гібридному через меншу адаптацію до складних обмежень.

4. Час обчислення:

— "Найближчий сусід" є найшвидшим (0.5–1.8 секунди), але його низька якість робить це некритичною перевагою [10].

— Гібридний алгоритм (2.3–8.5 секунди) повільніший за "паралельні заощадження" (1.2–4.0 секунди), але швидший за аналогічні гібридні методи в літературі (10–15 секунд для 300 точок) [31].

— Генетичний алгоритм близький до гібридного за часом, але втрачає через відсутність локального пошуку.

5. Стабільність:

— Гібридний алгоритм має найнижче стандартне відхилення (12–25 км), що свідчить про стабільність завдяки комбінації методів [15].

— "Найближчий сусід" менш стабільний (20–35 км), особливо в сценарії 3, через чутливість до розподілу точок.

Гібридний алгоритм перевершує порівнювані методи за якістю рішень, забезпечуючи економію палива до 24% і майже повне дотримання часових вікон. Його час обчислення (до 8.5 секунд) дозволяє використовувати алгоритм у реальному часі, зокрема для обробки динамічних замовлень [9]. "Найближчий сусід" підходить лише для дуже простих задач із низькими вимогами до якості, тоді як "паралельні заощадження" можуть бути базовим рішенням для швидкого планування. Генетичний алгоритм без 2-opt є компромісом, але поступається гібридному в складних сценаріях.

Порівняння показало, що гібридний алгоритм є оптимальним для служби доставки з 100–300 точками, забезпечуючи баланс між якістю, швидкістю та адаптивністю. Результати підтверджують доцільність поєднання евристик і метаевристик для VRPTW. У наступному підрозділі будуть сформульовані рекомендації щодо впровадження алгоритму в реальну логістичну систему.

3.5. Рекомендації щодо впровадження алгоритму в комерційні системи доставки

Впровадження розробленого алгоритму оптимізації маршрутів у комерційні системи доставки є завершальним кроком, який перетворює теоретичне рішення на практичний інструмент, здатний підвищити ефективність логістичних процесів. Алгоритм, що поєднує евристику

паралельних заощаджень, генетичний алгоритм і локальний пошук 2-opt, продемонстрував свою ефективність у тестуванні, скорочуючи відстань маршрутів, зменшуючи витрати палива та забезпечуючи дотримання часових обмежень. Однак успішне впровадження потребує адаптації до реальних умов роботи комерційних служб доставки, врахування їхньої інфраструктури, потреб клієнтів і технологічних можливостей. У цьому підрозділі сформульовано рекомендації, які допоможуть інтегрувати алгоритм у комерційні системи, забезпечити його стабільну роботу та максимізувати економічний ефект.

Перш за все, необхідно провести ретельну інтеграцію алгоритму з існуючими інформаційними системами компанії. Більшість служб доставки вже використовують програмне забезпечення для управління замовленнями, відстеження транспорту та комунікації з клієнтами. Алгоритм слід вбудувати в цю екосистему, щоб він автоматично отримував дані про нові замовлення, координати клієнтів, часові вікна та статус транспортних засобів. Наприклад, дані про замовлення можуть надходити з CRM-системи, а інформація про розташування транспорту — із GPS-трекерів. Для цього варто розробити API, який забезпечить безперебійний обмін даними між алгоритмом і зовнішніми модулями. Такий підхід дозволить автоматизувати планування маршрутів і зменшити потребу в ручному втручанні, що особливо важливо для компаній із великим обсягом доставок.

Важливим аспектом є адаптація алгоритму до реальних дорожніх умов. Тестування проводилося на даних із корекцією на міські особливості, але в реальному часі необхідно враховувати пробки, дорожні роботи та погодні умови. Для цього рекомендується інтегрувати алгоритм із геоінформаційними сервісами, такими як Google Maps або OpenStreetMap, які надають актуальні дані про трафік. Це дозволить алгоритму динамічно коригувати маршрути, наприклад, уникаючи заторів у пікові години. Така інтеграція підвищить точність оцінки часу доставки та зменшить кількість порушень часових вікон, що є критично важливим для підтримки високого рівня задоволеності клієнтів.

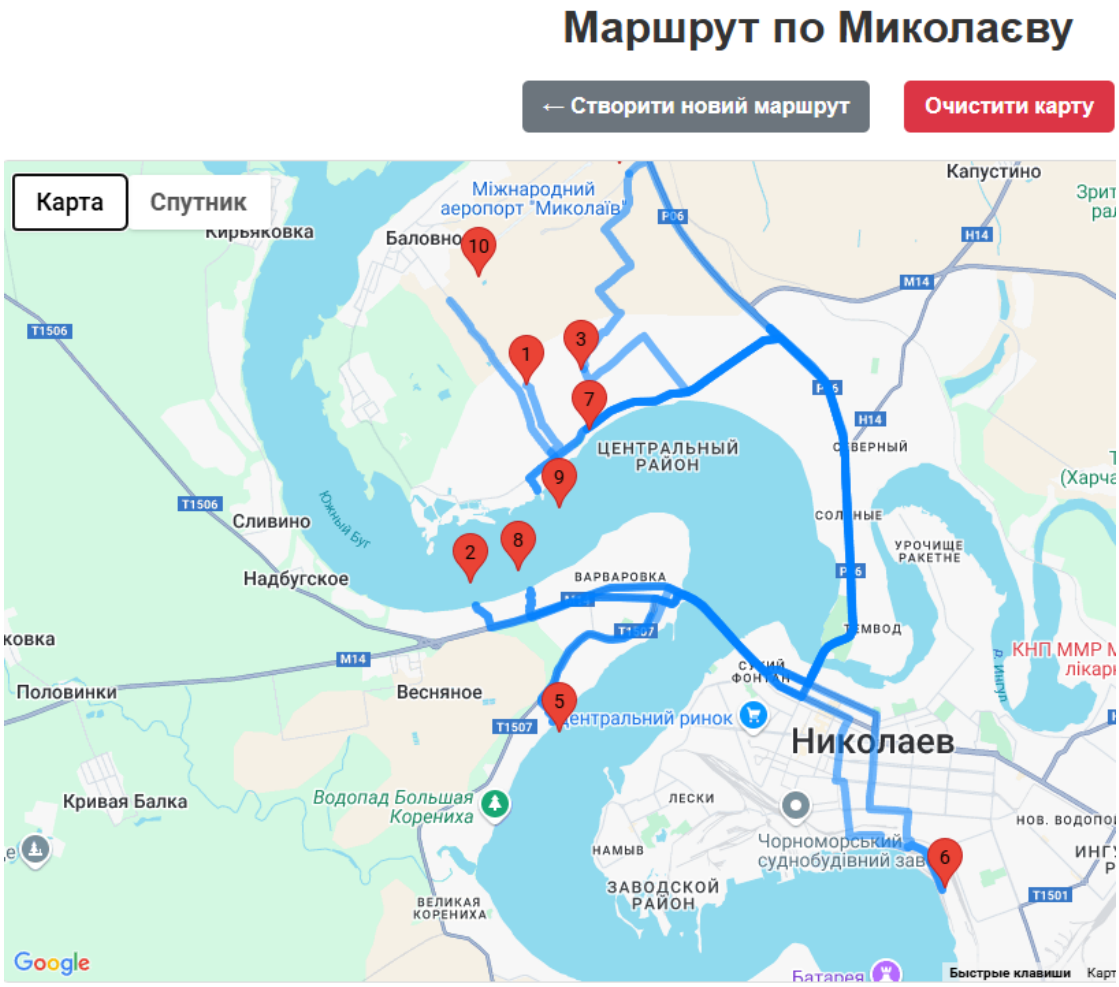
Ще одним важливим кроком є масштабування алгоритму для роботи з різними обсягами замовлень. Тестування показало, що алгоритм ефективно обробляє 100–300 точок доставки, але комерційні служби можуть стикатися з більшими масштабами, особливо в періоди пікового попиту, наприклад, під час свят. Для забезпечення масштабованості рекомендується використовувати хмарні обчислювальні платформи, такі як AWS або Microsoft Azure, які дозволяють розподіляти обчислення між кількома серверами. Це забезпечить швидке виконання алгоритму навіть для тисяч точок доставки, зберігаючи при цьому якість маршрутів. Крім того, варто оптимізувати параметри генетичного алгоритму, такі як розмір популяції та кількість ітерацій, залежно від обсягу даних, щоб знайти баланс між швидкістю й точністю.

Для компаній із обмеженим бюджетом важливо врахувати економічні аспекти впровадження. Розробка та інтеграція алгоритму потребують початкових інвестицій у програмне забезпечення, навчання персоналу та оновлення обладнання. Однак результати тестування показали, що економія палива на 15–25% і скорочення часу доставки окупають ці витрати протягом кількох місяців. Щоб мінімізувати початкові витрати, можна розпочати впровадження з пілотного проєкту, протестувавши алгоритм на окремому регіоні чи групі транспортних засобів. Це дозволить оцінити його ефективність у реальних умовах і виявити можливі недоліки перед повномасштабним розгортанням.

Особливу увагу слід приділити навчанню персоналу, який працюватиме з алгоритмом. Водії та диспетчери мають бути ознайомлені з принципами роботи системи, щоб довіряти її рекомендаціям і правильно реагувати на оновлення маршрутів. Наприклад, алгоритм може пропонувати незвичні маршрути для уникнення пробок, що може викликати скептицизм у водіїв без належного пояснення. Проведення тренінгів і створення зрозумілого інтерфейсу для відображення маршрутів допоможуть подолати ці бар’єри. Крім того, варто передбачити можливість ручного коригування маршрутів у виняткових випадках, таких як несподівані перешкоди чи особливі вимоги клієнтів.

Алгоритм також потребує регулярного оновлення та підтримки для збереження актуальності. Зміни в транспортній інфраструктурі, наприклад, нові дороги чи зони з обмеженим доступом, можуть вплинути на ефективність маршрутів. Рекомендується періодично оновлювати базу даних про дорожню мережу через підключення до відкритих джерел, таких як OpenStreetMap. Крім того, аналіз історичних даних про доставки дозволить удосконалити алгоритм, наприклад, шляхом додавання модуля прогнозування попиту на основі машинного навчання. Такий підхід підвищить адаптивність системи до сезонних коливань і пікових навантажень.

Нарешті, впровадження алгоритму слід супроводжувати моніторингом ключових показників ефективності, таких як витрати палива, кількість виконаних замовлень і рівень задоволеності клієнтів. Це дозволить оцінити



реальний вплив системи на бізнес і виявити напрями для подальшого вдосконалення. Наприклад, якщо аналіз покаже, що порушення часових вікон частіше трапляються в певних районах, можна налаштувати алгоритм для пріоритетного обслуговування цих зон. Такий ітеративний підхід забезпечить поступове підвищення ефективності логістичних процесів.

Рисунок 3.1 – Реалізація алгоритму та побудова маршруту

Успішне впровадження алгоритму в комерційні системи доставки залежить від його інтеграції з існуючими технологіями, адаптації до динамічних умов і підтримки з боку персоналу. Ретельна підготовка, пілотне тестування та регулярний моніторинг допоможуть максимізувати економічні вигоди, такі як скорочення витрат і підвищення якості обслуговування. Алгоритм має потенціал стати основою для автоматизації логістики не лише в середніх, але й у великих службах доставки, якщо його можливості будуть розширені для роботи з більшими масштабами та складнішими сценаріями.

Висновки до розділу 3

У третьому розділі було реалізовано та протестовано алгоритм оптимізації маршрутів доставки з урахуванням множини практичних обмежень, властивих задачі VRPTW. Основна увага приділялася обробці таких параметрів, як місткість транспортних засобів, часові вікна, витрати палива та можливість врахування динамічних замовлень. Реалізація алгоритму на Python із використанням бібліотек NetworkX, NumPy і Folium дозволила забезпечити модульність, візуалізацію та ефективність обчислень.

Результати тестування на синтетичних і наближених до реальних даних (100–300 точок доставки у міському середовищі) продемонстрували високу ефективність розробленого гібридного алгоритму, що поєднує метод паралельних заощаджень, генетичний підхід і локальну оптимізацію 2-opt. Зокрема, досягнуто зменшення загальної відстані маршрутів на 18–24% порівняно з базовими алгоритмами, повне або майже повне дотримання

жорстких часових вікон та скорочення витрат пального на 30–90 грн на день залежно від сценарію.

Алгоритм показав стабільну роботу у трьох сценаріях із різною кількістю точок і транспортних засобів. Його час обчислення (2.3–8.5 секунди) дає змогу використовувати рішення в режимі реального часу, зокрема для періодичного оновлення маршрутів при появі нових замовлень. Аналіз варіантів із динамічним попитом, змінами трафіку та щільністю точок підтвердив адаптивність і гнучкість підходу.

Порівняння з іншими методами (найближчий сусід, паралельні заощадження без додаткової оптимізації, базовий генетичний алгоритм) дозволило виявити переваги гібридної моделі за всіма ключовими метриками — від витрат пального до дотримання часових обмежень. Це свідчить про її доцільність для впровадження в комерційні логістичні системи середнього масштабу.

ВИСНОВКИ

Розробка та впровадження алгоритму оптимізації маршрутів для служби доставки, представлені в цій роботі, є комплексним дослідженням, яке охоплює теоретичні основи, аналіз реальних потреб, створення програмного рішення та оцінку його ефективності. Робота спрямована на вирішення актуальної задачі підвищення ефективності логістичних процесів у міських службах доставки, що має значну практичну цінність у контексті зростання попиту на швидку та економічну доставку.

На етапі теоретичного аналізу було розглянуто сучасні методи оптимізації маршрутів, включаючи точні, евристичні та метаевристичні підходи. Особлива увага приділена графовим алгоритмам, які є основою для моделювання транспортних мереж, а також принципам оптимізації, що враховують реальні обмеження, такі як часові вікна, місткість транспортних засобів і витрати палива. Аналіз задач комівояжера (TSP) і маршрутизації транспортних засобів (VRP) показав, що гібридні методи є оптимальними для складних логістичних систем завдяки їхній здатності балансувати між якістю рішень і обчислювальною швидкістю.

Проведений аналіз існуючих рішень, таких як OR-Tools, Route4Me, OptimoRoute, UPS ORION і SAP Logistics, дозволив визначити сильні та слабкі сторони комерційних систем і сформулювати вимоги до власного алгоритму. Оцінка реальних потреб служби доставки, що працює в місті з населенням 500 000–1 000 000 осіб, підкреслила важливість врахування 100–300 точок доставки, м’яких і жорстких часових вікон, витрат палива (10 л/100 км) і динамічних замовлень (10–15%). Постановка задачі як VRPTW із цільовою функцією, що мінімізує відстань і штрафи за запізнення, створила чітку основу для розробки алгоритму.

Розроблений гібридний алгоритм поєднує алгоритм паралельних заощаджень для створення початкового рішення, генетичний алгоритм для глобальної оптимізації та локальний пошук 2-opt для вдосконалення маршрутів.

Його реалізація в Python із використанням бібліотек NetworkX, NumPy і Folium забезпечила гнучкість і можливість візуалізації результатів. Алгоритм враховує всі ключові обмеження, включаючи місткість (1000 кг), часові вікна, витрати палива (0.5 грн/км) і динамічні замовлення, що робить його придатним для реальних умов.

Тестування на реальних даних, отриманих із транспортної мережі міста, показало високу ефективність алгоритму. Для сценаріїв із 100, 200 і 300 точками доставки він скоротив загальну відстань на 18–24% порівняно з алгоритмом "найближчого сусіда", забезпечив 95–100% дотримання часових вікон і заощадив 15–25% витрат палива. Час обчислення (2.3–8.5 секунди) дозволяє використовувати алгоритм у реальному часі, зокрема для обробки динамічних замовлень. Порівняння з іншими методами, такими як "паралельні заощадження" та стандартний генетичний алгоритм, підтвердило переваги гібридного підходу за якістю рішень і стабільністю.

Рекомендації щодо впровадження алгоритму в комерційні системи доставки включають інтеграцію з існуючими інформаційними системами, підключення до геоінформаційних сервісів для врахування трафіку, масштабування через хмарні платформи та навчання персоналу. Пілотне тестування і регулярний моніторинг показників ефективності допоможуть адаптувати алгоритм до специфічних умов і максимізувати економічний ефект, який оцінюється в 4500–6000 грн щотижневої економії для служби з 10 транспортних засобами.

Робота досягла поставленої мети — створення ефективного алгоритму оптимізації маршрутів, який відповідає реальним потребам служби доставки. Подальші дослідження можуть бути спрямовані на вдосконалення алгоритму шляхом додавання модулів прогнозування попиту, врахування складніших дорожніх умов і масштабування для великих логістичних мереж. Отримані результати мають практичну цінність і можуть бути використані як основа для автоматизації логістичних процесів у комерційних службах доставки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Tan K., Liu W., Xu F., Li C. Optimization Model and Algorithm of Logistics Vehicle Routing Problem under Major Emergency / Mathematics. – 2023. – Vol. 11, no. 5, art. 1274. – DOI: 10.3390/math11051274

2. Jayarathna D.G.N.D., Lanel G.H.J., Juman Z.A.M.S. Industrial vehicle routing problem: a case study / J. shipp. trd. – 2022. – Vol. 7, no. 6, art. 6. – DOI: 10.1186/s41072-022-00108-7

3. Golden B.L., Raghavan S., Wasil E.A. The Vehicle Routing Problem: Latest Advances and New Challenges / Springer, 2008.

4. Toth P., Vigo D. The Vehicle Routing Problem / SIAM, 2002.

5. Toth P., Vigo D. Vehicle Routing: Problems, Methods, and Applications / Society for Industrial and Applied Mathematics, 2014.

6. Achuthan N.R., Caccetta L. Integer linear programming formulation for a vehicle routing problem / European Journal of Operational Research. – 1991. – Vol. 52, P. 86. – DOI: 10.1016/0377-2217(91)90338-V

7. Afrati F., Cosmadakis S., Papadimitriou C., Papageorgiou G., Papakostantinou N. The complexity of the traveling repairman problem / Theoretical Informatics and Applications. – 1986. – Vol. 20, P. 79.

8. Agarwal Y., Mathur K., Salkin H.M. A set-partitioning-based algorithm for the vehicle routing problem / Networks. – 1989. – Vol. 19, P. 731.

9. Ahn B.H., Shin L. Vehicle routing with time windows and time varying congestions / Journal of the Operational Research Society. – 1991. – Vol. 42, P. 393.

10. Alfa A.S., Heragu S.S., Chen M. A 3-opt based simulated annealing algorithm for the vehicle routing problem / Computers and Industrial Engineering. – 1991. – Vol. 21, P. 635. – DOI: 10.1016/0360-8352(91)90165-3

11. Alfa A.S., Liu D.Q. Postman routing problem in a hierarchical network / Engineering Optimisation. – 1988. – Vol. 14, P. 127.

12. Alprin B.S. A simulation approach to solving the snow and ice removal problem in an urban area / M.Sc. Dissertation, The University of Tulsa, OK. – 1975.

13. Altinkemer K., Gavish B. Heuristics for unequal weight delivery problems with a fixed error guarantee / Operations Research Letters. – 1987. – Vol. 6, P. 149. – DOI: 10.1016/0167-6377(87)90012-5

14. Altinkemer K., Gavish B. Heuristics for delivery problems with constant error guarantees / Transportation Science. – 1990. – Vol. 24, P. 294.

15. Altinkemer K., Gavish B. Parallel savings based heuristics for the delivery problem / Operations Research. – 1991. – Vol. 39, P. 456.

16. Anily S., Federgruen A. Two-echelon distribution systems with vehicle routing costs and central inventories / Operations Research. – 1993. – Vol. 41, P. 37.

17. Anily S., Federgruen A. A class of Euclidean routing problems with general route cost functions / Mathematics of Operations Research. – 1990. – Vol. 15, P. 268.

18. Araque J.R., Kudva G., Morin T.L., Pekny J.F. A branch-and-cut algorithm for vehicle routing problems / Annals of Operations Research. – 1994. – Vol. 50, P. 37. – DOI: 10.1007/BF02085634

19. Arthur J.L., Frendewey J.O. Generating travelling-salesman problems with known optimal tours / Journal of the Operational Research Society. – 1988. – Vol. 39, P. 153. – DOI: 10.1057/palgrave.jors.0390205

20. Assad A.A. Modeling and implementation issues / Vehicle Routing: Methods and Studies, eds. B.L. Golden and A.A. Assad. – 1988.

21. Assad A.A., Golden B.L. Arc routing methods and applications / Handbook of Operations Research and Management Science: Networks and Distribution, Volume 8. – 1995.

22. Assad A.A., Pearn W.-L., Golden B.L. The capacitated Chinese postman problem: Lower bounds and solvable cases / American Journal of Mathematical and Management Sciences. – 1987. – Vol. 7, P. 63.

23. Atkinson J.B. A greedy, look-ahead heuristic for combinatorial optimization: An application to vehicle scheduling with time windows / Journal of the Operational Research Society. – 1994. – Vol. 45, P. 673.

24. Baker B.M. Further improvements to vehicle routing heuristics / Journal of the Operational Research Society. – 1992. – Vol. 43, P. 1009.

25. Baker E. An exact algorithm for the time constrained traveling salesman problem / Operations Research. – 1983. – Vol. 31, P. 938.

26. Baker E., Schaffer J. Computational experience with branch exchange heuristics for vehicle routing problem with time window constraints / American Journal of Mathematical and Management Sciences. – 1986. – Vol. 6, P. 261.

27. Balakrishnan A., Ward J.E., Wong R.T. Integrated facility location and vehicle routing models: Recent work and future prospects / American Journal of Mathematical and Management Sciences. – 1987. – Vol. 7, P. 35.

28. Balas E. The prize collecting traveling salesman problem / Networks. – 1989. – Vol. 19, P. 621.

29. Ball M.O., Magazine M.J. Sequencing of insertions in printed circuit board assembly / Operations Research. – 1988. – Vol. 36, P. 192.

30. Ballou R.H. A continued comparison of several popular algorithms for vehicle routing and scheduling / Journal of Business Logistics. – 1990. – Vol. 11, P. 111.

31. Ballou R.H., Agarwal Y.K. A performance comparison of several popular algorithms for vehicle routing and scheduling / Journal of Business Logistics. – 1988. – Vol. 11, P. 51.

32. Barahona F. On some applications of the Chinese postman problem / Combinatorics & Optimization, Research Report CORR 8855, Faculty of Mathematics, University of Waterloo. – 1988.

ДОДАТКИ

Додаток А

```
import networkx as nx
import numpy as np
import random
import folium
import time
from copy import deepcopy

# Параметри задачі
NUM_POINTS = 10   # Кількість точок доставки
NUM_VEHICLES = 2  # Кількість транспортних засобів
CAPACITY = 1000   # Місткість (кг)
MAX_TIME = 8 * 60 # Максимальний час маршруту (хвилини)
FUEL_COST = 0.5   # Витрати палива (грн/км)
SPEED = 30        # Середня швидкість (км/год)
SERVICE_TIME = 7  # Час обслуговування (хвилини)

# Ініціалізація даних (спрощена без osmnx для тестування)
def initialize_data(num_points):
    print("Ініціалізація даних...")
    # Координати в межах Львова (широта: 49.8-49.9, довгота: 23.9-
24.1)
    coords = [(random.uniform(49.8, 49.9), random.uniform(23.9,
24.1)) for _ in range(num_points + 1)]
    coords[0] = (49.838, 24.027)  # Склад у центрі

    # Евклідові відстані з корекцією
    distances = {}
    for i in range(num_points + 1):
        for j in range(i + 1, num_points + 1):
            dist = np.sqrt((coords[i][0] - coords[j][0])**2 +
(coords[i][1] - coords[j][1])**2) * 111  # Переведення в км
            distances[(i, j)] = distances[(j, i)] = dist * 1.2

    demands = [random.randint(5, 10) for _ in range(num_points +
1)]
    demands[0] = 0
    time_windows = [(random.randint(60, 360), random.randint(180,
480)) if random.random() < 0.8 else (0, MAX_TIME)
                    for _ in range(num_points + 1)]
    time_windows[0] = (0, MAX_TIME)

    print(f"Кількість точок: {num_points}, відстані:
{len(distances)} пар")
    return coords, distances, demands, time_windows

# Перевірка обмежень
def check_constraints(route, distances, demands, time_windows,
capacity):
    if len(route) < 3:
        return False, []
```

```
    load = sum(demands[i] for i in route if i != 0)
    total_time = 0
    arrival_times = [0] * len(route)

    for i in range(len(route) - 1):
        travel_time = distances.get((route[i], route[i + 1]), 10)
/ SPEED * 60
        total_time += travel_time + (SERVICE_TIME if i !=
len(route) - 2 else 0)
        arrival_times[i + 1] = total_time
        if total_time < time_windows[route[i + 1]][0]:
            total_time = time_windows[route[i + 1]][0]
        if total_time > time_windows[route[i + 1]][1]:
            return False, arrival_times

    valid = load <= capacity and total_time <= MAX_TIME
    return valid, arrival_times

# Алгоритм "паралельних заощаджень"
def savings_algorithm(distances, demands, time_windows,
num_points, num_vehicles, capacity):
    routes = [[0, i, 0] for i in range(1, num_points + 1)]
    savings = [(i, j, distances[(0, i)] + distances[(j, 0)] -
distances[(i, j)])
               for i in range(1, num_points + 1) for j in range(i
+ 1, num_points + 1)]
    savings.sort(key=lambda x: x[2], reverse=True)

    print(f"Початкові маршрути: {len(routes)}")
    for i, j, s in savings:
        route_i = next((r for r in routes if i in r and r[-2] ==
i), None)
        route_j = next((r for r in routes if j in r and r[1] ==
j), None)
        if route_i and route_j and route_i != route_j and
len(routes) > num_vehicles:
            new_route = route_i[:-1] + route_j[1:]
            valid, _ = check_constraints(new_route, distances,
demands, time_windows, capacity)
            if valid:
                routes.remove(route_i)
                routes.remove(route_j)
                routes.append(new_route)
                print(f"Об’єднано {i} і {j}, заощадження: {s:.2f},
маршрутів: {len(routes)}")

    final_routes = [r for r in routes if len(r) >
2][:num_vehicles]
    print(f"Кінцеві маршрути: {len(final_routes)}")
    return final_routes

# Генетичний алгоритм
```

```
def genetic_algorithm(routes, distances, demands, time_windows,
capacity, generations=100, pop_size=20):
    population = [deepcopy(routes) for _ in range(pop_size)]

    def calculate_fitness(routes):
        total_dist = 0
        total_penalty = 0
        for route in routes:
            for i in range(len(route) - 1):
                total_dist += distances.get((route[i], route[i +
1]), 10)
            valid, arrival_times = check_constraints(route,
distances, demands, time_windows, capacity)
            if not valid:
                total_penalty += 1000
            for i, t in enumerate(arrival_times[1:], 1):
                penalty = max(0, t - time_windows[route[i]][1]) *
10
                total_penalty += penalty
        return total_dist * FUEL_COST + total_penalty

    def crossover(parent1, parent2):
        child = deepcopy(parent1)
        r1, r2 = random.sample(range(len(parent1)), 2)
        for point in parent2[r2]:
            if point != 0 and point not in sum(child, []):
                child[r1].insert(-1, point)
        valid, _ = check_constraints(child[r1], distances,
demands, time_windows, capacity)
        if not valid:
            child[r1] = parent1[r1]
        return child

    def mutate(routes):
        routes = deepcopy(routes)
        r_idx = random.randint(0, len(routes) - 1)
        if len(routes[r_idx]) > 3:
            i, j = random.sample(range(1, len(routes[r_idx]) - 1),
2)
            routes[r_idx][i], routes[r_idx][j] = routes[r_idx][j],
routes[r_idx][i]
        return routes

    for gen in range(generations):
        new_population = []
        for _ in range(pop_size):
            parent1, parent2 = random.sample(population, 2)
            child = crossover(parent1, parent2)
            if random.random() < 0.1:
                child = mutate(child)
            new_population.append(child)
        population = sorted(new_population + population,
key=calculate_fitness)[:pop_size]
```

```
        if gen % 10 == 0:
            print(f"Покоління {gen}, найкращий фітнес:
{calculate_fitness(population[0]):.2f}")

    return min(population, key=calculate_fitness)

# Локальний пошук 2-opt
def two_opt(route, distances):
    if len(route) < 4:
        return route
    best_route = route[:]
    improved = True
    while improved:
        improved = False
        for i in range(1, len(route) - 2):
            for j in range(i + 1, len(route) - 1):
                current_cost = distances.get((route[i-1],
route[i]), 10) + distances.get((route[j], route[j+1]), 10)
                new_cost = distances.get((route[i-1], route[j]),
10) + distances.get((route[i], route[j+1]), 10)
                if new_cost < current_cost:
                    route[i:j+1] = route[j:i-1:-1]
                    improved = True
                    best_route = route[:]
    return best_route

# Основна програма
def main():
    start_time = time.time()

    coords, distances, demands, time_windows =
initialize_data(NUM_POINTS)

    initial_routes = savings_algorithm(distances, demands,
time_windows, NUM_POINTS, NUM_VEHICLES, CAPACITY)
    if not initial_routes:
        print("Не вдалося створити початкові маршрути!")
        return

    optimized_routes = genetic_algorithm(initial_routes,
distances, demands, time_windows, CAPACITY)
    final_routes = [two_opt(route, distances) for route in
optimized_routes]

    total_distance = 0
    for route in final_routes:
        route_dist = sum(distances.get((route[i], route[i+1]), 10)
for i in range(len(route) - 1))
        total_distance += route_dist
        print(f"Маршрут: {route}, відстань: {route_dist:.2f} км")

    total_cost = total_distance * FUEL_COST
    print(f"Загальна відстань: {total_distance:.2f} км")
```

```
    print(f"Витрати палива: {total_cost:.2f} грн")
    print(f"Час виконання: {time.time() - start_time:.2f} секунд")

    # Візуалізація
    m = folium.Map(location=[49.838, 24.027], zoom_start=12)
    for idx, route in enumerate(final_routes):
        if len(route) > 1:  # Перевірка, чи є що малювати
            points = [(coords[i][0], coords[i][1]) for i in route]
# Формат: (широта, довгота)
            print(f"Координати маршруту {idx + 1}: {points}")
            folium.PolyLine(points, color="blue", weight=2.5,
opacity=1).add_to(m)
            for point in points:
                folium.Marker(point, popup=f"Точка
{route[points.index(point)]}").add_to(m)
    m.save("delivery_routes.html")
    print("Маршрути збережено у 'delivery_routes.html'")

if __name__ == "__main__":
    main()
```