Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Комп'ютерного моделювання та інтелектуальних технологій
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Метод побудови альтернативних маршрутів для негабаритних перевезень на основі модифікованого алгоритму Дейкстри та контекстного аналізу транспортних обмежень
(тема)

Виконав:
здобувач II року навчання,
групи СПРм-24-1
Серветник Дмитро Сергійович
(власне ім'я, прізвище)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Системне проєктування
(повна назва освітньої програми)
Керівник: доц. Петрова Р. В.
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ Гребеннік І. В.
(підпис) (власне ім'я, прізвище)

2026 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Комп'ютерного моделювання та інтелектуальних технологій

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

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

Освітня програма Системне проєктування
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____» _____ 2026 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Серветнику Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Метод побудови альтернативних маршрутів для негабаритних перевезень перевезень на основі модифікованого алгоритму Дейкстри та контекстного аналізу транспортних обмежень

затверджена наказом по університету від 6 квітня 2026 р. № 268 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 20 травня 2026 р.

3. Вихідні дані до роботи Розробити метод побудови альтернативних маршрутів для негабаритних перевезень на основі модифікованого алгоритму Дейкстри та алгоритму Єна з урахуванням транспортних обмежень. Використовувати мову програмування Python, фреймворк Flask та графову модель транспортної мережі.

4. Перелік питань, що потрібно опрацювати у роботі 4.1 Вступ, 4.2 Аналіз предметної області та існуючих методів маршрутизації, 4.2.1 Особливості негабаритних перевезень, 4.2.2 Транспортні обмеження та їх класифікація, 4.2.3 Аналіз сучасних методів маршрутизації, 4.2.4 Алгоритми пошуку найкоротшого шляху, 4.2.5 Аналіз алгоритму Дейкстри, 4.2.6 Проблеми побудови маршрутів з обмеженнями, 4.2.7 Постановка задачі, 4.3 Моделювання транспортної мережі, 4.3.1 Представлення транспортної мережі у вигляді графа, 4.3.2 Вершини та ребра графа, 4.3.3 Агрегована модель транспортної мережі, 4.3.4 Дискретизована геометрична модель дороги, 4.3.5 Зважений граф та функції вартості, 4.3.6 Моделювання транспортних обмежень, 4.4 Розробка методу побудови маршрутів, 4.4.1 Модифікація алгоритму Дейкстри, 4.4.2 Врахування жорстких та м'яких обмежень, 4.4..3 Використання штрафних функцій, 4.4.4 Контекстний аналіз транспортних обмежень, 4.4.5 Побудова альтернативних маршрутів, 4.4.6 Алгоритм k-найкоротших шляхів (алгоритм Єна), 4.4.7 Загальна схема роботи методу, 4.5 Програмна реалізація, 4.5.1 Архітектура системи та вибір технологій, 4.5.2 Реалізація графа, 4.5.3 Реалізація модифікованого алгоритму Дейкстри, 4.5.4 Реалізація обмежень, 4.5.5 Реалізація альтернативних маршрутів,

4.5.6 Візуалізація маршрутів, 4.5.7 Інтерфейс користувача, 4.6 Аналіз результатів, 4.6.1 Опис тестових сценаріїв, 4.6.2 Аналіз маршрутів, 4.6.3 Аналіз альтернативних маршрутів, 4.6.4 Переваги та недоліки, 4.7 Висновок.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних страцій (п.5 включається до завдання за рішенням випускової кафедри) 5.1 Узагальнена блок-схема роботи алгоритму Дейкстри, 5.2 Приклад графового представлення транспортної мережі, 5.3 Приклад агрегованої моделі: прямі лінії між обласними центрами, 5.4 Приклад дискретизованої моделі: ламана лінія, що повторює реальну трасу, 5.5 Схема роботи модифікованого алгоритму Дейкстри, 5.6 Схема перевірки жорстких транспортних обмежень, 5.7 Схема використання штрафних функцій, 5.8 Схема контекстного аналізу транспортних обмежень, 5.9 Схема побудови альтернативних маршрутів, 5.10 Схема роботи алгоритму Єна, 5.11 Загальна схема роботи методу побудови маршрутів,

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

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Розділи спеціальної частини	Петрова Р. В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін використання етапів роботи	Примітка
	Отримання завдання	26.01.2026	Виконано
1	Аналіз предметної області та існуючих методів	27.01.2026 - 28.02.2026	Виконано
2	Аналіз алгоритму Дейкстри	29.02.2026 - 12.03.2026	Виконано
3	Моделювання транспортної мережі та представлення мережі у вигляді графа	13.03.2026 – 21.03.2026	Виконано
4	Аналіз агрегованої моделі транспортної мережі	22.03.2026 – 27.03.2026	Виконано
5	Аналіз дискретизованої геометричної модель	28.03.2026 – 05.04.2026	Виконано
6	Розробка методу побудови маршрутів	06.04.2026 – 12.04.2026	Виконано
7	Контекстний аналіз транспортних обмежень	13.04.2026 – 20.04.2026	Виконано
8	Реалізація модифікованого алгоритму Дейкстри	21.04.2026 – 25.04.2026	Виконано
9	Реалізація альтернативних маршрутів	26.04.2026 – 02.05.2026	Виконано
10	Аналіз маршрутів та альтернативних маршрутів	03.05.2026 – 09.05.2026	Виконано
11	Оформлювання пояснювальної записки	09.05.2026	Виконано
12	Попередній захист кваліфікаційної роботи	19.05.2026	Виконано
13	Представлення кваліфікаційної роботи до ЕК	20.06.2026	Виконано

Дата видачі завдання 26 січня 2026 р.

Здобувач _____
(підпис)

Керівник роботи _____ доц. Петрова Р. В.
(підпис) (посада, власне ім'я, прізвище)

Я, як студент ХНУРЕ, розумію і підтримую політику закладу із академічної доброчесності. Я не надавав і не одержував недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

«16» травня 2025 р. Серветник Д. С.

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.
Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено «19» травня 2026 р.

Керівник кваліфікаційної роботи доц. Петрова Р. В.

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи містить: 137 с., 45 рис., 8 табл., 2 дод., 22 джерела.

НЕГАБАРИТНІ ПЕРЕВЕЗЕННЯ, МАРШРУТИЗАЦІЯ, АЛГОРИТМ ДЕЙКСТРИ, АЛГОРИТМ ЄНА, ГРАФОВА МОДЕЛЬ, ТРАНСПОРТНІ ОБМЕЖЕННЯ, АЛЬТЕРНАТИВНІ МАРШРУТИ, WEB-ДОДАТОК, FLASK, PYTHON.

Об’єктом дослідження в роботі є процес побудови маршрутів для негабаритних перевезень з урахуванням транспортних обмежень дорожньої мережі.

Предметом дослідження є методи та програмні засоби побудови альтернативних маршрутів на основі модифікованого алгоритму Дейкстри та алгоритму k-найкоротших шляхів.

Метою роботи є розробка методу побудови альтернативних маршрутів для негабаритних перевезень із врахуванням жорстких та м’яких транспортних обмежень, а також створення програмної реалізації системи маршрутизації.

Методи дослідження – теорія графів, алгоритми пошуку найкоротших шляхів, методи структурного аналізу, об’єктно-орієнтоване проектування, моделювання транспортних мереж.

Сфера застосування – логістичні системи, транспортні компанії, системи підтримки прийняття рішень у сфері негабаритних перевезень та інтелектуальні транспортні системи.

ABSTRACT

The explanatory note to the qualification work contains: 137 pages, 45 figures, 8 tables, 2 appendices, 22 sources.

OVERSIZED TRANSPORTATION, ROUTING, DIJKSTRA'S ALGORITHM, YEN'S ALGORITHM, GRAPH MODEL, TRANSPORT CONSTRAINTS, ALTERNATIVE ROUTES, WEB APPLICATION, FLASK, PYTHON.

The object of research is the process of route construction for oversized transportation taking into account transport constraints of the road network.

The subject of research is methods and software tools for constructing alternative routes based on the modified Dijkstra's algorithm and the k-shortest paths algorithm.

The purpose of the work is to develop a method for constructing alternative routes for oversized transportation considering hard and soft transport constraints, as well as to create a software implementation of the routing system.

Research methods – graph theory, shortest path search algorithms, structural analysis methods, object-oriented design, transport network modeling.

Field of application – logistics systems, transport companies, decision support systems in the field of oversized transportation, and intelligent transport systems.

ЗМІСТ

ПЕРЕЛІК УМОВИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – (Application Programming Interface) – прикладний програмний інтерфейс.

HTTP – (Hypertext Transfer Protocol) – протокол передачі гіпертексту.

JSON – (JavaScript Object Notation) – текстовий формат обміну даними.

OSM – (OpenStreetMap) – некомерційний картографічний проєкт з відкритими даними.

АО – алгоритмічне обмеження (у контексті графових моделей).

ГМ – графова модель (транспортної мережі).

КНШ – k-найкоротших шляхів (відповідно до алгоритму Єна). ММ – математична модель.

ТМ – транспортна мережа.

ТО – транспортне обмеження (габаритні, вагові та інші ліміти).

ВСТУП

У сучасних умовах розвитку транспортної інфраструктури та логістичних систем питання ефективного планування маршрутів набуває особливої актуальності. Постійне зростання обсягів перевезень великогабаритних та великовагових вантажів зумовлює необхідність створення інтелектуальних систем маршрутизації, здатних враховувати специфічні транспортні обмеження дорожньої мережі. До таких перевезень належать транспортування промислового обладнання, будівельної техніки, енергетичних конструкцій, елементів інфраструктури та інших вантажів, габарити або маса яких перевищують допустимі нормативні значення.

На відміну від класичних задач маршрутизації, де основним критерієм є мінімізація відстані або часу руху, у задачах негабаритних перевезень ключовим фактором стає допустимість маршруту. Наявність мостів із обмеженням висоти, вузьких ділянок дороги, обмежень по ширині, масі чи радіусу повороту може зробити найкоротший шлях непридатним для використання. Крім того, частина транспортних обмежень має не абсолютний, а рекомендаційний характер, що потребує додаткового аналізу та врахування штрафних коефіцієнтів при побудові маршруту.

Сучасні навігаційні сервіси орієнтовані переважно на стандартний транспорт і не забезпечують повноцінної підтримки негабаритних перевезень. Хоча існують професійні логістичні системи, більшість із них є комерційними, закритими та потребують значних фінансових витрат. Тому актуальним є створення власних методів маршрутизації, які б поєднували ефективність класичних алгоритмів пошуку шляху з можливістю врахування транспортних обмежень.

У роботі розглядається задача побудови альтернативних маршрутів для негабаритних перевезень на основі модифікованого алгоритму Дейкстри та алгоритму k-найкоротших шляхів Єна. Запропонований підхід дозволяє враховувати як жорсткі обмеження, що повністю забороняють проїзд певними ділянками, так і м’які обмеження, які впливають на пріоритетність вибору

маршруту через систему штрафів.

Особливу увагу приділено моделюванню транспортної мережі у вигляді графа, де вершини відповідають транспортним вузлам, а ребра містять інформацію про довжину дороги, допустимі габарити та додаткові штрафи. На основі цієї моделі реалізовано програмний застосунок із використанням мови Python та фреймворку Flask, що дозволяє будувати декілька альтернативних маршрутів між заданими точками.

Актуальність теми полягає у необхідності автоматизації процесу планування негабаритних перевезень, зменшення ризику виникнення аварійних ситуацій, оптимізації логістичних витрат та підвищення ефективності транспортних операцій. Розробка подібних систем має важливе значення для транспортної галузі, оскільки дозволяє підвищити безпеку руху та забезпечити більш гнучке управління перевезеннями.

Об’єктом дослідження є процес побудови маршрутів для негабаритних перевезень у транспортних мережах.

Предметом дослідження є методи та програмні засоби побудови альтернативних маршрутів на основі модифікованого алгоритму Дейкстри та контекстного аналізу транспортних обмежень.

Метою роботи є розробка методу побудови альтернативних маршрутів для негабаритних перевезень із врахуванням транспортних обмежень та створення програмної реалізації системи маршрутизації.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– проаналізувати особливості негабаритних перевезень та існуючі методи маршрутизації;

– дослідити алгоритми пошуку найкоротших шляхів;

– розробити модель транспортної мережі у вигляді графа;

– модифікувати алгоритм Дейкстри для врахування транспортних обмежень;– реалізувати механізм врахування жорстких та м’яких обмежень;

– реалізувати побудову альтернативних маршрутів за допомогою алгоритму Єна;

– створити програмний застосунок для візуалізації маршрутів;

– провести тестування та аналіз отриманих результатів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ МЕТОДІВ МАРШРУТИЗАЦІЇ

1.1 Особливості негабаритних перевезень

Негабаритні перевезення – це транспортування вантажів, розміри або маса яких перевищують встановлені нормативні обмеження для звичайних транспортних засобів. До таких вантажів належать великогабаритне промислове обладнання, будівельна техніка, трансформатори, вітрові турбіни, мостові конструкції та інші об'єкти, що потребують спеціальних умов переміщення дорогами загального користування.

Головна відмінність негабаритних перевезень від стандартних логістичних задач полягає в тому, що основний критерій, мінімальна відстань або час, часто відходить на другий план. На перше місце виходить допустимість маршруту. Навіть якщо знайдено короткий шлях, він може виявитися непридатним через конкретні характеристики окремих ділянок дороги. Наприклад, міст із обмеженням висоти в 4 метри стає непереборною перешкодою для автопоїзда з вантажем висотою 4,5 метра, незалежно від того, наскільки цей шлях є оптимальним за відстанню.

Практика показує, що при плануванні таких перевезень потрібно враховувати цілий комплекс факторів. Найбільш значущими серед них є габаритні обмеження, максимально допустима висота, ширина та довжина транспортного засобу разом із вантажем. Вузькі тунелі, низькі мости, лінії електропередач, щільна міська забудова, кожен з цих об'єктів накладає жорсткі вимоги. Вагові обмеження теж критичні: міст може мати допустиме навантаження 25 тонн, і перевищення цього значення загрожує руйнуванням конструкції.

Також існують обмеження, пов'язані з радіусом повороту. Довгомірні автопоїзди не можуть виконувати різкі маневри на перехрестях або в тісних промислових зонах. Якщо дорога має гострий поворот, навіть відповідаючи

габаритам по висоті та ширині, вона може стати непридатною. Крім того, у багатьох містах діють часові заборони на рух великогабаритного транспорту в денний час або у визначені години пік. Для окремих перевезень потрібне отримання дозволів і навіть супровід патрульних машин, що теж впливає на вибір маршруту.

Окремо варто згадати про *м'які* обмеження, які не забороняють рух, але роблять його небажаним. Наприклад, проїзд через центральну частину міста може бути формально дозволеним, але пов'язаний із високим ризиком заторів, вузькими вулицями, підвищеною увагою до пішоходів. У таких випадках логісти планують об'їзні шляхи, навіть якщо вони довші, щоб зменшити ризики.

Із цих особливостей випливає, що класичні алгоритми пошуку найкоротшого шляху, які оперують лише довжиною або часом, не можуть безпосередньо застосовуватися для негабаритних перевезень. Потрібно модифікувати як модель транспортної мережі, так і сам алгоритм, щоб він перевіряв відповідність характеристик кожного ребра параметрам транспортного засобу. Це ставить задачу побудови маршрутів у розряд задач із жорсткими обмеженнями, а не лише оптимізаційних.

1.2 Транспортні обмеження та їх класифікація

Щоб будувати маршрути для негабаритних перевезень, потрібно чітко розуміти, які саме обмеження впливають на можливість проїзду тією чи іншою ділянкою дороги. На практиці всі транспортні обмеження можна поділити на дві великі групи: жорсткі та м'які.

Жорсткі обмеження – це ті, які фізично забороняють рух, якщо параметри транспортного засобу перевищують допустимі значення. Якщо хоча б одне таке обмеження не виконується, відповідна ділянка дороги стає непридатною, і маршрут, що її містить, не може бути реалізованим. До жорстких обмежень належать:

1. Габаритні обмеження: максимальна висота, ширина та довжина транспортного засобу разом із вантажем. Наприклад, тунель з висотою 4,2 м автоматично виключає автопоїзд висотою 4,5 м.

2. Вагові обмеження: максимальне навантаження на вісь або загальна маса, допустима на мосту чи ділянці дороги. Перевищення загрожує руйнуванням покриття або конструкцій.

3. Обмеження за радіусом повороту: для довгомірних транспортних засобів деякі перехрестя або серпантини можуть бути фізично непрохідними через недостатню ширину проїзної частини на поворотах.

Жорсткі обмеження враховуються шляхом прямої перевірки: для кожного ребра графа зберігаються значення, і якщо висота або ширина транспортного засобу перевищує ці значення, ребро просто виключається з пошуку.

М'які обмеження – це ті, які не забороняють рух, але роблять проїзд небажаним. Їх враховують через додаткові штрафи, які збільшують вартість використання ребра. Наприклад, проїзд через центральну частину великого міста часто є формально дозволеним для негабаритного транспорту, але пов'язаний із заторами, вузькими вулицями, великою кількістю пішоходів. Логістично вигідніше обрати об'їзну дорогу, навіть якщо вона довша. Також до м'яких обмежень можна віднести екологічні зони (підвищені вимоги до викидів), години пік (часові обмеження зі штрафами), дороги з платним проїздом, де вартість може впливати на вибір.

У коді м'які обмеження реалізовані через параметр penalty, який додається до довжини ребра при розрахунку загальної вартості маршруту. Таким чином, алгоритм обиратиме шлях із меншим штрафом, навіть якщо він трохи довший.

Зручно розрізняти ці два типи за принципом дії: жорсткі обмеження – це вмикачі-вимикачі (ребро або доступне, або ні), а м'які – це регулятори (вони впливають на пріоритет, але не виключають варіант). Такий поділ дозволяє гнучко налаштовувати алгоритм під різні задачі. Наприклад, для звичайного

вантажного автомобіля можна застосувати тільки м'які штрафи за в'їзд у центр міста, а для негабариту, додати жорсткі габаритні фільтри.

Варто зауважити, що частина обмежень може мати часову динаміку. Наприклад, заборона руху великогабаритного транспорту діє з 7:00 до 20:00, а вночі дозволена. Проте в рамках цієї роботи транспортна мережа розглядається як статична модель, де обмеження не змінюються під час роботи алгоритму. Це спрощення прийнятне для початкового етапу дослідження, але в реальних системах дані можуть оновлюватися перед кожним запуском.

Отже, класифікація обмежень на жорсткі та м'які дає чітку основу для модифікації алгоритму пошуку маршрутів. Далі потрібно розглянути, які існуючі методи маршрутизації використовуються на практиці та наскільки вони придатні для роботи з такими обмеженнями.

1.3 Аналіз сучасних методів маршрутизації

Сьогодні існує чимало підходів до побудови маршрутів, від простих навігаційних застосунків до складних логістичних платформ. Більшість із них орієнтована на стандартний транспорт і не враховує специфіку негабаритних перевезень.

Найпоширеніший клас – це сервіси масового використання, такі як Google Maps, Waze, Here WeGo. Вони будують маршрути на основі реальних дорожніх даних, враховують затори, платні дороги, тимчасові перекриття. Для звичайного водія цього достатньо. Але ці системи не оперують габаритними обмеженнями, вони вважають, що будь-який автомобіль проїде будь-де. Навіть якщо в налаштуваннях є режим «вантажний автомобіль», перевіряється лише заборона в'їзду для вантажівок, а не конкретна висота чи ширина. Тому для перевезення екскаватора або трансформатора такий сервіс може запропонувати маршрут під низький міст, і водій опиниться в глухому куті.

Інший підхід реалізований у спеціалізованих логістичних системах (наприклад, Trimble, ALK Maps, PTV RouteOptimizer). Вони дозволяють

задавати габарити транспортного засобу, масу, вісьове навантаження, а також містять бази даних з обмеженнями висотою мостів, пропускною здатністю доріг, заборонами на повороти. Ці системи професійні, але дорогі, їхні бази даних закриті, а алгоритми часто є комерційною таємницею. До того ж вони розраховані на великі логістичні компанії, а не на дослідницькі задачі.

Окрему групу становлять алгоритми пошуку шляху, які використовують у наукових дослідженнях. Основа багатьох із них, класичний алгоритм Дейкстри [1], або його евристичний варіант A* (A-star). Ці алгоритми гарантовано знаходять найкоротший шлях у графі з невід'ємними вагами. Вони добре вивчені, прості в реалізації та ефективні для статичних мереж. Але в чистому вигляді вони непридатні для задач з обмеженнями, оскільки не перевіряють жодних додаткових умов на ребрах.

Метод Беллмана-Форда [2] дозволяє працювати з від'ємними вагами, але в транспортних задачах це рідко потрібно, а його складність вища. Алгоритм Флойда-Воршелла знаходить усі пари вершин, але для великих мереж (тисячі вершин) він стає занадто повільним.

Для пошуку кількох варіантів маршруту застосовуються алгоритми k-найкоротших шляхів (k-shortest paths). Найвідоміший – це алгоритм Єна (Yen), який будує список кращих шляхів без повторів. Він працює через послідовне відхилення від вже знайденого маршруту. Його недолік – це обчислювальна складність зростає зі збільшенням k, але для невеликих значень (3–5) це прийнятно. У сучасних бібліотеках, наприклад SciPy, реалізовано алгоритм Єна для пошуку k найкоротших шляхів [12].

Останніми роками набирають популярність методи на основі машинного навчання, наприклад, навчання з підкріпленням для динамічної маршрутизації. Проте вони потребують великих обсягів даних, не завжди гарантують оптимальність і важко інтерпретуються. У задачах з жорсткими обмеженнями, де помилка може призвести до аварії, такі методи поки що застосовують обережно.

Отже, більшість наявних методів маршрутизації або не враховують

габаритних обмежень, або є закритими комерційними рішеннями. Відкриті алгоритми (Дейкстра, A*) потребують модифікації, щоб перевіряти відповідність параметрів транспортного засобу характеристикам дорожніх ділянок. Саме така модифікація й буде запропонована в наступних розділах. Перед цим варто детальніше розглянути алгоритми пошуку найкоротшого шляху, щоб зрозуміти, де саме потрібно вносити зміни.

1.4 Алгоритми пошуку найкоротшого шляху

Задача знаходження найкоротшого шляху між двома точками в мережі виникає не тільки в транспортній логістиці, а й у телекомунікаціях, соціальних графах, ігровому штучному інтелекті. Для її розв’язання розроблено кілька класичних алгоритмів, кожен зі своїми сильними та слабкими сторонами.

Алгоритм Дейкстри [1] – найпопулярніший метод для графів з невід’ємними вагами ребер. Він працює за принципом жадібного розширення: спочатку відстань до початкової вершини дорівнює нулю, а до решти нескінченність. На кожному кроці вибирається вершина з найменшою поточною відстанню, яку ще не обробляли, і релаксуються всі її сусіди. Якщо через поточну вершину шлях до сусіда стає коротшим, значення оновлюється. Алгоритм завершується, коли всі досяжні вершини оброблені. Завдяки використанню пріоритетної черги (наприклад, двійкової купи) час роботи становить $O((V+E) \log V)$, що дозволяє обробляти великі мережі. Недоліком вляється працювання з від’ємними вагами, але в транспортних задачах довжини доріг або час проїзду завжди додатні.

Алгоритм Беллмана-Форда [2] – більш універсальний, оскільки підтримує ребра з від’ємною вагою та виявляє негативні цикли. Він базується на динамічному програмуванні: виконує $V-1$ ітерацій релаксації всіх ребер, що гарантує знаходження найкоротших шляхів, якщо негативні цикли відсутні. Складність: $O(V \cdot E)$, що для великих графів стає повільно. У транспортних мережах від’ємні ваги практично не зустрічаються, тому

Беллмана-Форд застосовують рідко.

Алгоритм Флойда-Воршелла знаходить найкоротші шляхи між усіма парами вершин. Він працює з матрицею відстаней, послідовно покращуючи оцінки через проміжні вершини [3]. Складність: $O(V^3)$ для графів із тисячами вершин це вже критично, тому в задачах маршрутизації в реальному часі його не використовують. Він корисний, коли потрібно разом отримати всі попарні відстані для компактного зберігання, але не для одноразового пошуку.

Алгоритм A* (A-star) – це евристична версія [4] Дейкстри, яка пришвидшує пошук, спрямовуючи його до цільової вершини. Для цього використовується евристична оцінка відстані від поточної вершини до цілі (наприклад, пряма лінія на карті). Загальна вартість шляху через вершину v оцінюється як $f(v)=g(v)+h(v)$, де g – фактична відстань від старту, h – евристичне наближення до цілі. Якщо h є допустимою (не переоцінює реальну відстань), A* гарантує оптимальний шлях і часто працює швидше за Дейкстру за рахунок перегляду меншої кількості вершин. У навігаційних системах A* використовують активно, але його ефективність сильно залежить від якості евристики.

Для задач, де потрібна не просто відстань, а декілька різних маршрутів, застосовують алгоритми k-найкоротших шляхів, але про них детальніше в розділі 1.6.

1.5 Аналіз алгоритму Дейкстри

Серед усіх алгоритмів пошуку найкоротших шляхів алгоритм Дейкстри [1] є найбільш відомим і найчастіше згадуваним у науковій та прикладній літературі. Його запропонував нідерландський інформатик Едсгер Дейкстра ще в 1959 році [1], і з того часу він залишається базовим інструментом для розв’язання задач маршрутизації у статичних графах із невід’ємними вагами ребер.

Суть алгоритму доволі проста. Нехай є граф, де кожне ребро має вагу

(наприклад, довжину дороги, час проїзду або вартість палива). Алгоритм поступово «розширює» множину вершин, для яких уже знайдено найкоротшу відстань від початкової точки. Спочатку ця відстань дорівнює нулю для стартової вершини та нескінченності для решти. На кожному кроці обирається вершина з найменшою поточною відстанню серед ще не оброблених, а потім перевіряються всі її сусіди: якщо шлях через поточну вершину до сусіда виявляється коротшим за вже відомий, значення оновлюється. Коли всі вершини оброблено, алгоритм завершується. Для прискорення роботи зазвичай використовують пріоритетну чергу [5], що дає складність O((V+E) log V), цілком прийнятно для більшості транспортних мереж.

Основна перевага алгоритму Дейкстри перед іншими методами (наприклад, Беллмана-Форда) – це його ефективність за умови, що всі ваги додатні. У транспортних задачах це завжди виконується, тому алгоритм став стандартом де-факто для навігаційних систем. Крім того, він гарантує знаходження глобально оптимального шляху, а не просто якогось прийнятного наближення.

Проте при застосуванні до задач негабаритних перевезень виявляються суттєві обмеження. По-перше, алгоритм працює лише з одним критерієм – вагою ребра. Він не може самостійно врахувати кілька параметрів одночасно, наприклад, і довжину дороги, і обмеження по висоті. Якщо спробувати «скласти» всі обмеження в одну вагу, то доведеться вирішувати, як саме їх комбінувати, а це не завжди очевидно. По-друге, класичний Дейкстра не розрізняє жорсткі та м'які обмеження. Для нього будь-яке ребро або доступне (якщо його вага скінченна), або ні. А на практиці деякі ділянки можуть бути заборонені повністю (міст занадто низький), а деякі, просто небажані (центр міста з пробками). По-третє, алгоритм знаходить лише один найкоротший шлях. У реальній логістиці часто потрібно мати кілька альтернативних варіантів про запас, наприклад, у разі несподіваного перекриття дороги або виникнення додаткових обмежень.

Є ще один аспект, на який варто звернути увагу. Алгоритм Дейкстри [1]

передбачає, що всі дані про граф (вершини, ребра, ваги) відомі заздалегідь і не змінюються під час пошуку. У реальному світі транспортна мережа динамічна: можуть з’явитися тимчасові перекриття, змінитися обмеження або погодні умови. Класична версія на це не реагує, потрібно перебудовувати граф і запускати пошук заново.

Для детального розуміння механізму роботи алгоритму розглянемо приклад на графі, де необхідно знайти найкоротшу відстань від початкової вершини 1 до всіх інших пунктів (рисунок 1.1) .

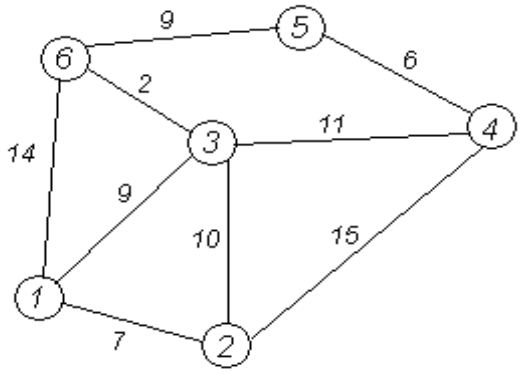

Рисунок 1.1 – Вихідний зважений граф для розрахунку

На початку роботи алгоритму кожній вершині присвоюється мітка: для стартової вершини 1 вона дорівнює 0, для всіх інших – нескінченності (∞), що означає відсутність попередньо розрахованого шляху (рисунок 1.2).

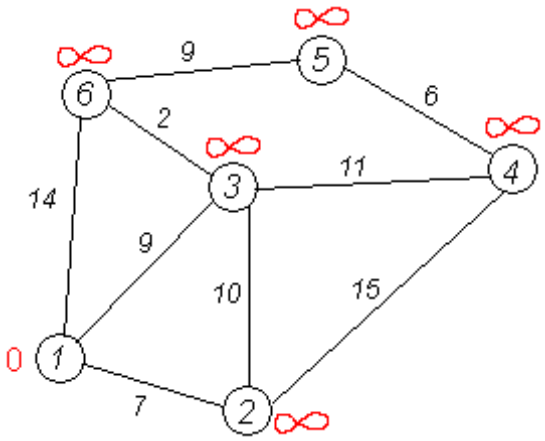

Рисунок 1.2 – Ініціалізація початкових міток вершин

Перший крок: оскільки вершина 1 має мінімальну мітку, вона стає поточною. Алгоритм аналізує її сусідів: 2, 3 та 6.

Для вершини 2 нова відстань становить 0 + 7 = 7 (рисунок 1.3). Оскільки 7 < ∞, мітка оновлюється.

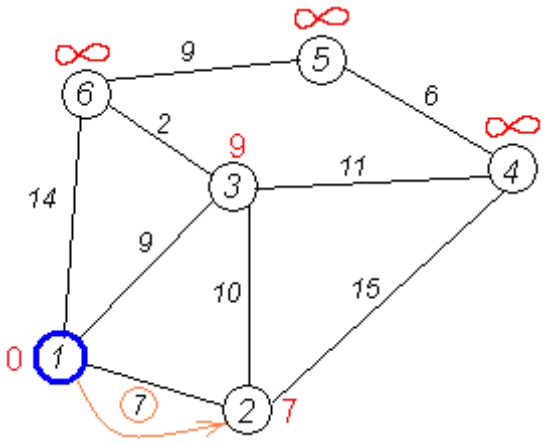

Рисунок 1.3 – Оцінка відстані до вершини 2

Аналогічно оновлюються мітки для вершин 3 (0 + 9 = 9) та 6 (0 + 14 = 14) (рисунок 1.4).

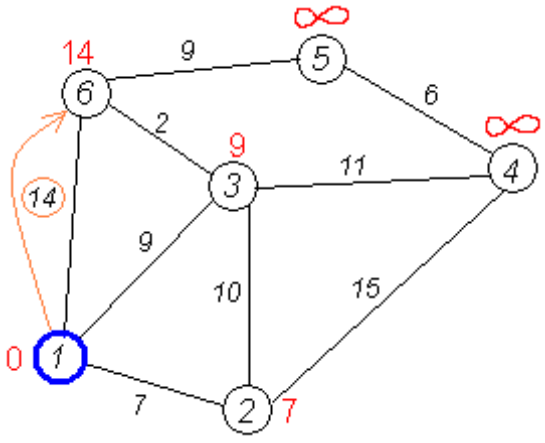

Рисунок 1.4 – оцінка відстані до верниш 3 та 6

Після перевірки всіх сусідів вершина 1 вважається відвіданою (рисунок 1.5).

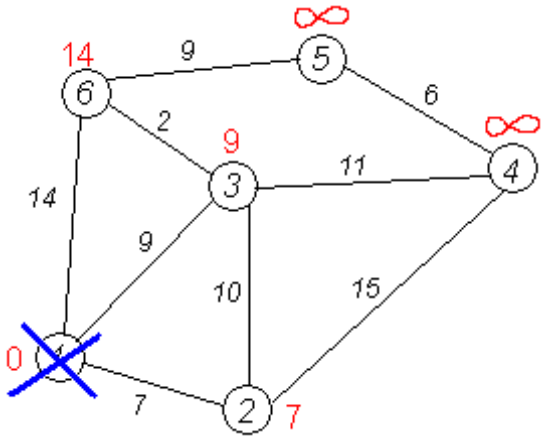

Рисунок 1.5 – Стан графа після обробки першої вершини

Другий крок: серед невідвіданих вершин обирається та, що має найменшу мітку – це вершина 2 з вагою 7 (рисунок 1.6). Алгоритм намагається покращити шляхи до її сусідів (1, 3, 4) через вершину 2. Вершина 1 вже відвідана, тому вона ігнорується.

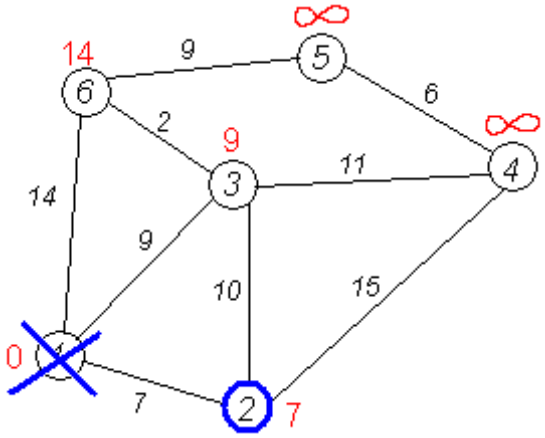

Рисунок 1.6 – Вибір вершини 2 для подальшої обробки

Для вершини 3 шлях через 2 становив би 7 + 10 = 17, але поточна мітка 9 менша, тому вона залишається без змін (рисунок 1.7).

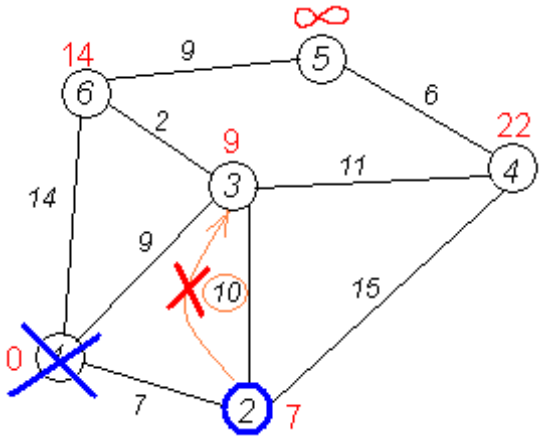

Рисунок 1.7 – Аналіз можливості скорочення шляху до вершини 3

До вершини 4 встановлюється нова мітка 7 + 15 = 22 (рисунок 1.8).

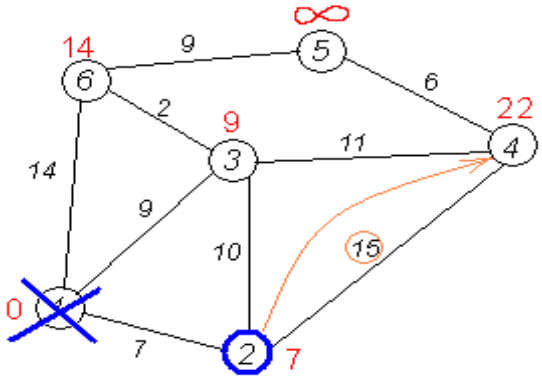

Рисунок 1.8 – Аналіз можливості скорочення шляху до вершини 4

Після завершення вершина 2 позначається як відвідана (рисунок 1.9).

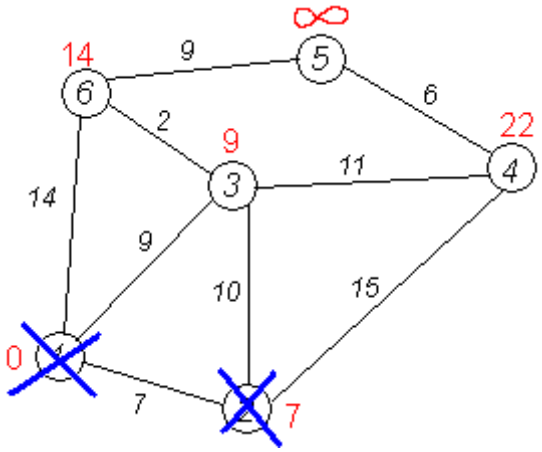

Рисунок 1.9 – Стан графа після відвідування вершини 2

Третій крок: наступною обирається вершина 3 з міткою 9. Після аналізу її зв’язків з іншими вузлами, мітки оновлюються відповідним чином (рисунок 1.10).

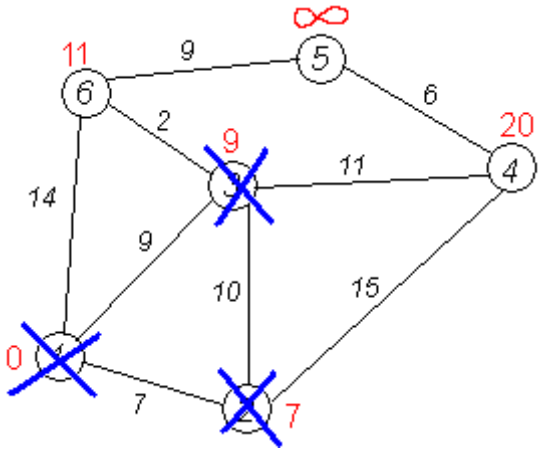

Рисунок 1.10 – Стан графа після відвідування вершини 3

Подальші кроки: процес повторюється ітераційно для вершин 6, 4 та 5 у порядку зростання їхніх міток (рисунки 1.11–1.12).

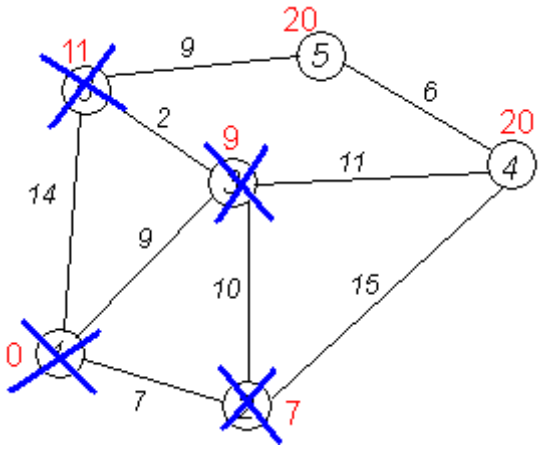

Рисунок 1.11 – Стан графа після відвідування вершини 6

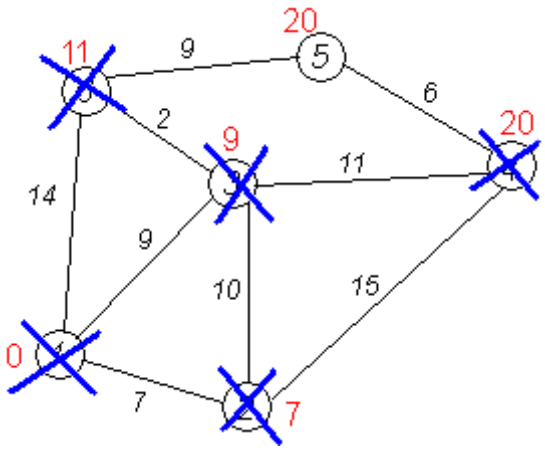

Рисунок 1.12 – Стан графа після відвідування вершини 4

Завершення: коли всі вузли графа відвідані, алгоритм зупиняє роботу. На підсумковому рисунку 1.13 представлено найкоротші відстані від джерела до кожного вузла: до вершини 2 – 7, до 3 – 9, до 4 – 20, до 5 – 20, до 6 – 11.

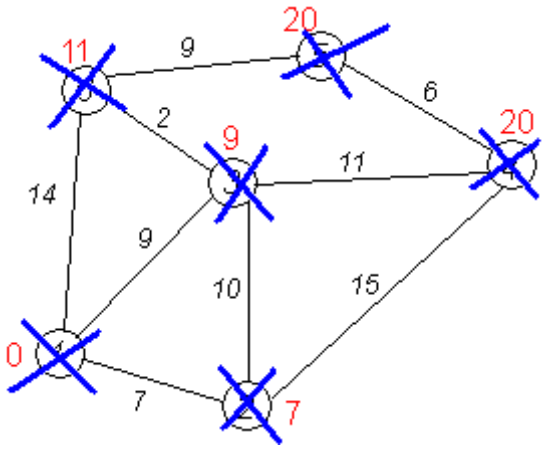

Рисунок 1.13 – Кінцевий результат роботи алгоритму Дейкстри

На основі проведеного покрокового аналізу можна зробити висновок, що ефективність алгоритму Дейкстри безпосередньо залежить від способу зберігання даних про невідвідані вершини [6]. У сучасних програмних реалізаціях для оптимізації швидкодії часто використовують пріоритетні черги (наприклад, на основі двійкової купи), що дозволяє знизити часову складність до O(E log V), де V – кількість вершин, а E – кількість ребер у графі.

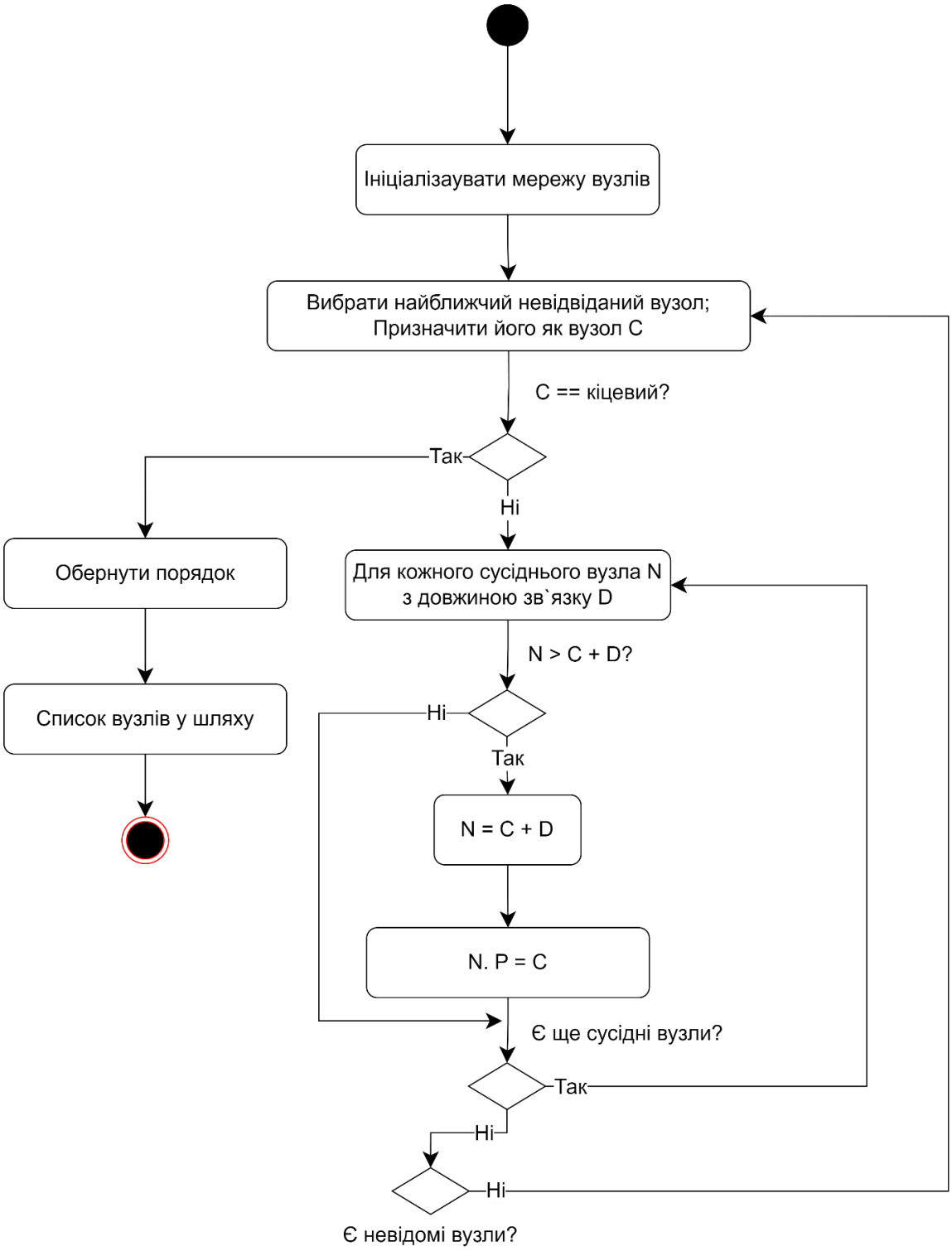

Рисунок 1.14 – Узагальнена блок-схема роботи алгоритму Дейкстри

Для задач маршрутизації негабаритних перевезень даний алгоритм є базовим фундаментом. Його головною перевагою у контексті цієї роботи є гарантія знаходження оптимального (найкоротшого) шляху за умови

відсутності ребер з від’ємною вагою. Це дозволяє використовувати результати роботи алгоритму Дейкстри як еталон для подальшого порівняння з іншими методами пошуку, зокрема при побудові альтернативних маршрутів за допомогою алгоритму Єна.

1.6 Проблеми побудови маршрутів з обмеженнями

На перший погляд, задача маршрутизації здається простою: знайти шлях з точки А в точку Б, бажано найкоротший. Але коли з’являються обмеження, все стає складнішим. І не тому, що алгоритми погані, а тому що сама постановка задачі змінюється.

У класичному випадку ми маємо граф, де кожне ребро має одну вагу. Алгоритм Дейкстри або A* спокійно знаходять оптимальний шлях, і це рішення буде правильним для будь-якого транспортного засобу, який може їздити будь-де. Але для негабаритних перевезень такий підхід не працює. Чому? Тому що різні ділянки дороги мають різні обмеження, і те, що дозволено для легкового автомобіля, може бути категорично заборонено для вантажівки з екскаватором.

Перша проблема: несумісність критеріїв, уявімо, що потрібно доставити великогабаритний вантаж. Є два маршрути: короткий (100 км), але він проходить під мостом заввишки 3,8 м, і довгий (150 км) в об’їзд, де всі обмеження дотримано. Для класичного алгоритму коротший маршрут завжди кращий. Він не знає, що висота вантажу 4,2 м, а міст нижчий. Якщо просто додати висоту як ще одну вагу, виникає питання: як порівнювати кілометри з метрами? Ускладнювати формулу, вводити коефіцієнти? Це можливо, але довільне.

Друга проблема: жорсткість обмежень: у математичному моделюванні зручно оперувати неперервними величинами, але тут обмеження мають пороговий характер: або проїхати можна, або ні. Якщо максимальна висота на мосту 4,0 м, то автомобіль висотою 3,9 м проїде, а 4,1 м, ні. Різниця всього 5

см, а результат кардинально різний. Алгоритм має це враховувати, але класичні методи оптимізації на це не розраховані.

Третя проблема: необхідність альтернатив, у стандартній задачі достатньо одного найкращого шляху. Але коли мова йде про негабарит, ситуація на дорозі може змінитися. Наприклад, під час руху виявилося, що на запланованому маршруті раптово почалися ремонтні роботи, і проїзд тимчасово закрито. Якщо у водія є заздалегідь прорахований другий варіант, він може швидко переключитися. Якщо ж ні, доведеться зупинятися, запускати пошук заново з урахуванням нових умов, що займає час. А для великогабариту час, це ще й гроші.

Четверта проблема пов'язана з доступністю та якістю даних, щоб врахувати обмеження, потрібно знати, де які мости, яка їхня висота, де є звуження дороги, де обмеження по масі. Відкриті картографічні сервіси (наприклад, OpenStreetMap) мають частину такої інформації, але вона часто неповна або застаріла. Офіційні джерела (місцеві дорожні служби) надають дані в різних форматах, не завжди придатних для автоматичної обробки. Тобто навіть якщо алгоритм вміє працювати з обмеженнями, без якісних вхідних даних він марний.

Ще один нюанс: динамічні обмеження, деякі обмеження діють не постійно, а лише в певні години. Наприклад, в’їзд великовагового транспорту в місто дозволений тільки вночі. Якщо алгоритм будує маршрут, не знаючи про час прибуття, він може запропонувати варіант, який у реальності виявиться забороненим. Класичний Дейкстра, який працює зі статичними даними, цього не враховує.

Отже, основні проблеми побудови маршрутів з обмеженнями зводяться до такого: потрібно вміти відкидати недопустимі ребра (жорсткі обмеження), вводити штрафи для небажаних, але допустимих ділянок (м'які обмеження), будувати не один, а кілька шляхів про запас, а також мати спосіб отримувати актуальні дані про обмеження. У наступному розділі буде показано, як саме можна модифікувати класичні алгоритми, щоб усе це врахувати.

1.7 Постановка задачі

Постановку задачі кваліфікаційної роботи можна сформулювати наступним чином: розробити метод побудови альтернативних маршрутів для негабаритних перевезень на основі модифікованого алгоритму Дейкстри та контекстного аналізу транспортних обмежень, а також створити програмний застосунок для практичної реалізації запропонованого підходу.

Розробка системи маршрутизації включає декілька основних етапів, спрямованих на забезпечення коректної роботи алгоритмів та врахування специфіки негабаритних перевезень. До таких етапів належать:

– аналіз предметної області та особливостей негабаритних перевезень;

– дослідження існуючих алгоритмів пошуку найкоротших шляхів;

– аналіз транспортних обмежень та їх класифікація;

– моделювання транспортної мережі у вигляді графа;

– визначення структури вершин та ребер графа;

– формування системи жорстких та м’яких обмежень;

– розробка модифікованого алгоритму Дейкстри;

– реалізація механізму штрафних функцій;

– реалізація побудови альтернативних маршрутів на основі алгоритму Єна;

– проектування архітектури програмного застосунку;

– вибір технологій розробки;

– програмна реалізація системи маршрутизації;

– реалізація візуалізації маршрутів на карті;

– тестування алгоритмів на різних сценаріях перевезень;

– аналіз ефективності запропонованого методу;

– порівняння результатів із класичними алгоритмами маршрутизації.

Виконання наведених етапів є необхідним для створення системи, здатної будувати допустимі та ефективні маршрути для негабаритного транспорту з урахуванням транспортних обмежень дорожньої мережі.

2 МОДЕЛЮВАННЯ ТРАНСПОРТНОЇ МЕРЕЖІ

2.1 Представлення транспортної мережі у вигляді графа

Перш ніж шукати маршрути, потрібно якось описати транспортну мережу так, щоб з нею міг працювати алгоритм. Найзручніший спосіб для цього, теорія графів [7]. Вона дозволяє абстрагуватися від реальної геометрії доріг і зосередитися на зв’язках між точками.

У графовій моделі транспортної мережі вершинами виступають певні об'єкти: перехрестя, в’їзди в міста, логістичні центри, мости, розв’язки. Ребрами, дороги або ділянки доріг, які з’єднують ці вершини. Таке спрощення дозволяє звести задачу пошуку маршруту до знаходження послідовності ребер, яка веде від початкової вершини до кінцевої.

На рисунку 2.1 показано приклад такого представлення [8]. Кружечки – вершини (А, Б, В, Г і тд.), лінії між ними – ребра (дороги). Кожне ребро має вагу – довжину, час проїзду або інший показник. У нашій задачі до ваги додаються ще параметри обмежень (висота, ширина, штраф).

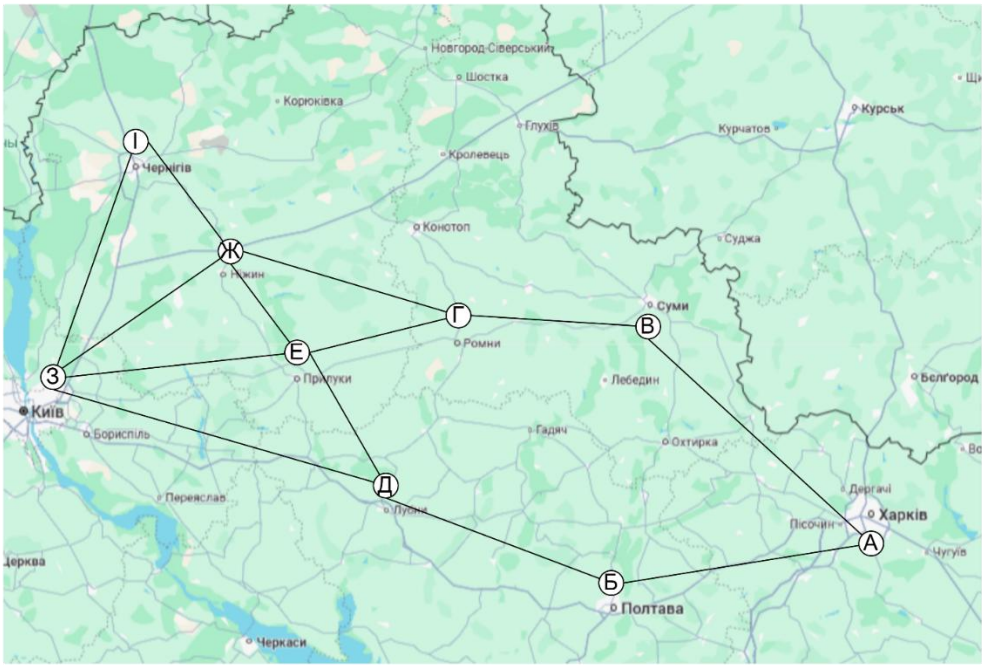

Рисунок 2.1 – Приклад графового представлення транспортної мережі

У графовій моделі важливі не точні координати вершин, а лише наявність зв’язків та їхні характеристики. Це дозволяє застосовувати стандартні алгоритми пошуку шляхів (той самий Дейкстру) і не думати про те, чи є дорога прямою чи звивистою – це вже деталізація наступних рівнів моделі.

Кожному ребру можна приписати одну або кілька характеристик. У найпростішому випадку це довжина ділянки в кілометрах. Тоді найкоротший маршрут – це той, сума довжин ребер якого мінімальна. Якщо замість довжини взяти середній час проїзду, алгоритм буде шукати найшвидший шлях. А для негабаритних перевезень цього замало.

Для задачі кожне ребро повинно нести додаткову інформацію про обмеження інфраструктури. Зокрема, максимальну допустиму висоту та ширину транспортного засобу. Також зручно додати штраф: число, яке показує, наскільки небажаним є проїзд цією ділянкою (наприклад, через центр міста). Тоді ребро описується набором параметрів: фізична довжина, обмеження по висоті та ширині, штраф. Саме така модель використана в реалізованому застосунку.

Варто зауважити, що в реальних навігаційних системах мережа є значно детальнішою. Дороги розбиваються на дрібні сегменти, іноді через кожні кілька метрів. Така дискретизована геометрична модель дозволяє точно відтворювати повороти, зміни смуг, розв’язки. Але для дослідження алгоритмів і для випробувань на рівні великих міст (обласних центрів) цілком достатньо агрегованої моделі, де вершинами є самі міста, а ребрами, магістралі між ними, без проміжних точок. Такий підхід значно зменшує кількість вершин і ребер, прискорює обчислення, але ціною є втрата точності на окремих ділянках. Наприклад, пряма лінія на карті між Києвом та Житомиром замість реальної звивистої траси. Це свідоме спрощення, яке не вважається помилкою – це просто рівень абстракції, прийнятний для задачі планування маршрутів на рівні обласних центрів.

Отже, графова модель дає формальну основу для застосування

алгоритмів пошуку шляхів. Далі потрібно визначитися, які саме вершини та ребра будуть у цій моделі, як вони пов’язані з реальними дорогами й обмеженнями. Цьому присвячені наступні підрозділи.

2.2 Вершини та ребра графа

У графовій моделі транспортної мережі вершини відповідають точкам, де щось змінюється: початок або кінець дороги, перехрестя, в’їзд у місто, міст, логістичний вузол. Усі ці об’єкти мають географічні координати й можуть бути потенційними точками маршруту. На практиці вершини поділяють на дві категорії: основні (наприклад, обласні центри, важливі транспортні вузли) та проміжні (невеликі населені пункти, розв’язки, точки зміни обмежень). Основні вершини зазвичай є обов’язковими для відображення у звіті або інтерфейсі, а проміжні додають, щоб краще передати реальну геометрію доріг.

Ребра з’єднують вершини та моделюють ділянки доріг. Кожному ребру приписують набір числових характеристик. Базовою є довжина – кілометри або інша фізична міра. Для задач із часом руху додають середню швидкість, звідки отримують час проходження. У випадку негабаритних перевезень важливі параметри обмежень: максимальна висота, ширина, довжина транспортного засобу, максимальна маса. Також до ребра можна прив’язувати штраф – додаткову вартість, яка не забороняє рух, але робить його менш бажаним.

Класифікація ребер за типами доріг (автомагістраль, регіональна траса, міська вулиця) допомагає задавати обмеження групово. Наприклад, для всіх міських вулиць можна встановити штраф +5 км, а для магістралей, нульовий, такий підхід спрощує введення даних.

Важливо, що граф може бути орієнтованим (коли рух дозволено лише в одному напрямку) або неорієнтованим (двосторонній рух). На міжміських трасах України більшість доріг двосторонні, тому в моделі зазвичай використовують неорієнтовані ребра, але для локальних ділянок (вулиці з

одностороннім рухом, розв’язки) потрібна орієнтація.

Місцеве спрощення: у реальності дорога може мати змінні обмеження вздовж свого протягу, але в агрегованій моделі вважається, що на всій ділянці між двома вершинами обмеження постійні. Якщо на трасі є міст із зниженою висотою, то поруч із мостом варто додати окремі вершини, а саму ділянку розбити на три ребра: підхід до мосту, міст, виїзд. Тоді обмеження діятиме лише на ребрі-мості. Такий спосіб дозволяє будувати гнучкі моделі, не втрачаючи точності.

Отже, визначення вершин і ребер – це перший крок до створення графа. Головне: зберегти баланс між детальністю моделі та обчислювальною складністю. У наступному підрозділі розглянемо, чому в дослідженні використовується агрегована модель, і в яких випадках краще перейти до дискретизованої.

2.3 Агрегована модель транспортної мережі

Коли будують граф для транспортної задачі, одразу постає питання: наскільки детальною має бути модель? Якщо взяти кожен метр дороги, кожен поворот, кожен міст, вийде гігантський граф із мільйонами вершин. Обчислення на такому графі будуть дуже повільними, хоча й точними. Якщо ж об’єднати цілі ділянки доріг в одне ребро, модель стане компактною, але втратить частину геометричної точності.

Агрегована модель – це якраз другий підхід, у ній вершинами виступають лише важливі пункти (обласні центри, великі логістичні вузли), а ребрами, магістральні дороги між ними. Проміжні повороти, дрібні населені пункти, другорядні перехрестя просто опускаються. Вважається, що маршрут проходить безпосередньо від одного центру до іншого по прямій або по одній трасі без внутрішніх розгалужень.

У цій роботі використовується саме агрегована модель. По-перше, кількість обласних центрів України невелика (близько 25), тому граф виходить

маленьким. Алгоритми на ньому працюють миттєво, що зручно для експериментів. По-друге, для перевезень між містами деталізація всередині області часто не критична, достатньо знати, якою трасою їхати та які обмеження на ній діють. По-третє, додавання проміжних вершин (як-от Бориспіль, Канів, Пирятин) уже дає змогу трохи деталізувати маршрут, не роздуваючи граф.

Звісно, у агрегованої моделі є недоліки, найпомітніший – це геометрія стає умовною. Ребро на карті зображується прямою лінією, хоча реальна дорога може петляти. Якщо дивитися на візуалізацію маршруту, лінія між Києвом та Львовом буде прямою, а не повторюватиме трасу М06. Для точного планування це не годиться, але для порівняння довжин маршрутів і перевірки обмежень на рівні обласних центрів, цілком прийнятно.

Інший недолік, утрата локальних обмежень, якщо на трасі Київ – Житомир є міст із низькою висотою в якомусь селі, агрегована модель цього не зафіксує, оскільки село не є вершиною. Щоб це виправити, потрібно додати село як проміжну вершину та розбити ребро на два — тоді обмеження можна прив’язати до конкретної ділянки. У наведеному прикладі з Дніпром і Запоріжжям саме так і зроблено: додано проміжні точки, щоб позначити зону зі зниженим габаритом.

Щоб наочно показати, як виглядає агрегована модель, на рисунку 2.2 зображено фрагмент транспортної мережі між обласними центрами. Кожне ребро – це пряма лінія, що з’єднує два міста, незалежно від реальної траси. Наприклад, ребро Київ–Житомир на карті буде прямою, хоча реальна дорога М06 петляє. Таке спрощення, свідомий компроміс між точністю та обчислювальною складністю. Візуально це виглядає як набір прямих відрізків між точками, що нагадує граф.

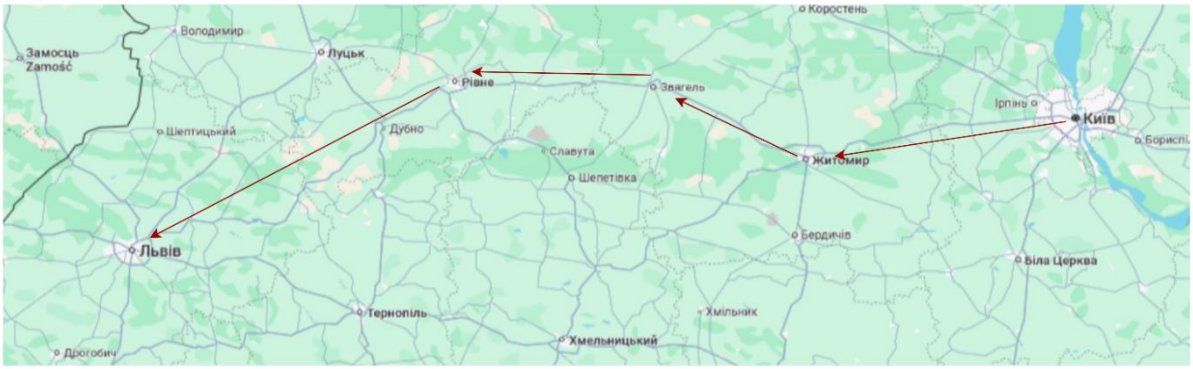

Рисунок 2.2 – Приклад агрегованої моделі: прямі лінії між обласними центрами

Отже, агрегована модель – це компроміс, вона підходить для дослідницьких цілей, коли важливо швидко перевірити ідею, а не будувати промислову систему. У реальних логістичних платформах зазвичай використовують дискретизовану геометричну модель, де дороги розбивають на короткі сегменти (іноді по 50–100 метрів), і кожен сегмент має свої обмеження. Така модель точніша, але вимагає значно більше даних і ресурсів, про неї йтиметься в наступному підрозділі.

2.4 Дискретизована геометрична модель дороги

Якщо агрегована модель працює з укрупненими ділянками, то дискретизована модель будується зовсім інакше. У ній реальна дорога розбивається на велику кількість коротких послідовних відрізків, іноді довжиною в десятки метрів або навіть менше. Кожен такий відрізок стає окремим ребром графа, а його початок і кінець, вершинами. У результаті граф отримує тисячі або мільйони вершин, зате здатен дуже точно відтворювати геометрію траси: повороти, підйоми, спуски, мости, естакади.

Такий підхід використовують у професійних навігаційних системах (Google Maps, Here, Navteq). Джерелом даних зазвичай слугують супутникові знімки, GPS-треки реальних автомобілів, а також офіційні карти дорожніх служб. Кожен короткий сегмент описується не лише координатами, але й

параметрами: швидкісний режим, категорія дороги, обмеження висоти/ширини, кількість смуг, наявність освітлення тощо.

Для негабаритних перевезень дискретизована модель має незаперечну перевагу: вона дозволяє враховувати обмеження, які діють на дуже коротких ділянках. Наприклад, низький міст довжиною 50 метрів, у агрегованій моделі його довелося б виділяти окремою вершиною, а в дискретизованій він автоматично стане одним або кількома ребрами з відповідними обмеженнями. Так само й звуження дороги, тимчасові ремонтні ділянки, залізничні переїзди, усе це можна прив’язати до конкретних сегментів.

Але є й мінуси: по-перше, обсяг даних стає величезним, зберігати й опрацьовувати мільйони вершин і ребер для всієї країни, нетривіальне завдання, що потребує потужних серверів і спеціальних структур даних. По-друге, алгоритми пошуку шляху на таких графах працюють довше, хоча завдяки евристикам і індексації (наприклад, ієрархічним алгоритмам) це прискорюють. По-третє, підтримувати актуальність даних складно: якщо на якомусь сегменті змінили обмеження, потрібно оновити відповідне ребро в базі, а це тисячі змін щодня.

На рисунку 2.3 показано ту саму ділянку Київ–Львів, але вже з деталізацією: реальна траса М06 петляє, повторюючи рельєф, огинаючи населені пункти. У дискретизованій моделі замість однієї прямої буде десятки або сотні ламаних відрізків, які точно відтворюють геометрію дороги.

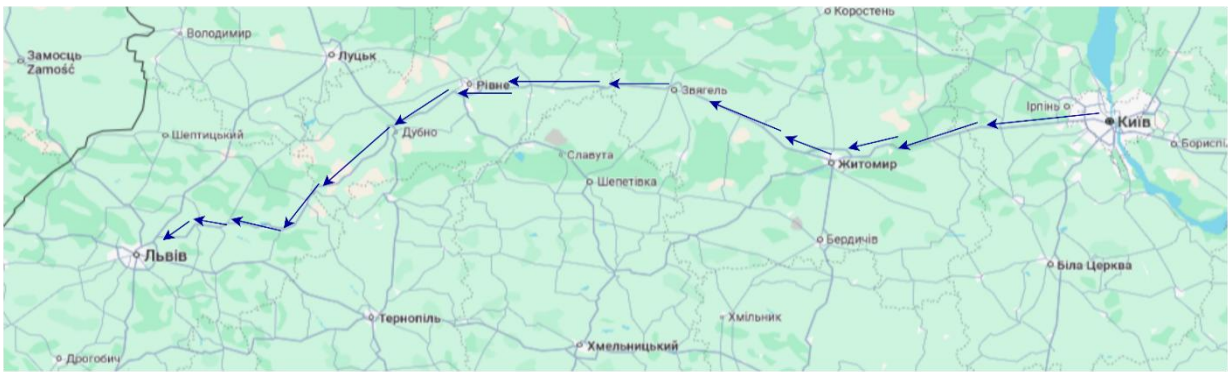

Рисунок 2.3 – Приклад дискретизованої моделі: ламана лінія, що повторює реальну трасу

Для наочності, рисунок 2.4 порівнює обидві моделі на одній ділянці: пряма (агрегована) та ламана (дискретизована). Видно, що дискретизована модель точніша, але вимагає значно більше даних і обчислювальних ресурсів.

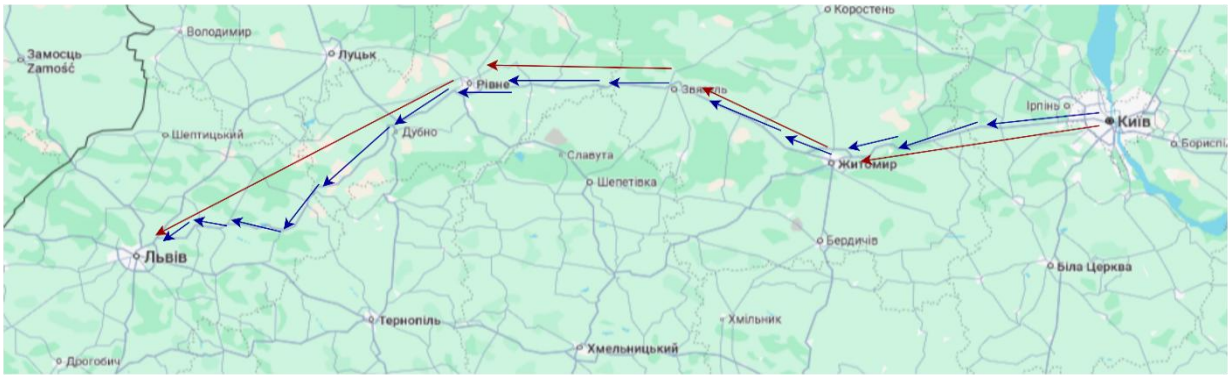

Рисунок 2.4 – Порівняння агрегованої (пряма) та дискретизованої (ламана) моделей

У рамках дипломної роботи дискретизована модель не використовується через її складність і великі вимоги до даних. Вона згадується тут для контексту: щоб показати, що існують різні рівні деталізації, і вибір того чи іншого залежить від конкретної задачі. Для дослідження методів маршрутизації з обмеженнями агрегованої моделі цілком достатньо, оскільки основна увага зосереджена на алгоритмічній частині, а не на геодезичній точності. Якщо в подальшому знадобиться практичне впровадження, агреговану модель можна розширити, додавши більше проміжних вершин і наблизивши її до дискретизованої. Наступний підрозділ пояснює, як саме задаються ваги ребер у зваженому графі.

2.5 Зважений граф та функції вартості

Коли граф побудовано, кожному ребру потрібно приписати числову характеристику «вагу». У класичній задачі пошуку найкоротшого шляху вага може бути довжиною дороги, часом проїзду, вартістю палива або будь-яким

іншим показником, який хочеться мінімізувати. Такий граф називають зваженим. Важливо, що ваги мають бути додатними (або хоча б невід'ємними), інакше алгоритм Дейкстри не гарантує коректної роботи.

У транспортних задачах найпоширеніші варіанти ваг:

1. Геометрична довжина – просто кілометри, підходить коли головне, мінімізувати пробіг.
2. Час – якщо відома середня швидкість на ділянці, можна обчислити час і шукати найшвидший маршрут.
3. Вартість – враховує витрати на пальне, амортизацію, платні дороги.

Але для негабаритних перевезень однієї ваги замало. Потрібно враховувати ще й обмеження, тому вводять функцію вартості, яка для кожного ребра повертає число, що відображає, наскільки це ребро «вигідне» з урахуванням усіх умов. Функція може залежати не тільки від довжини, а й від штрафів, а також від параметрів транспортного засобу (хоча це вже стосується перевірки допустимості).

Вартість проходження ребра визначається як:

$$\text{weight} = \text{length(e)} + \text{penalty(e)}$$

де length(e) – фізична довжина в кілометрах, а penalty(e) – штраф, який додають до загальної суми, щоб зробити ребро менш привабливим. Штраф може бути нульовим для більшості доріг і додатним для небажаних ділянок (наприклад, центр міста, дорога з поганим покриттям). Чим більший штраф, тим менша ймовірність, що алгоритм обере це ребро.

Чому не можна просто збільшити довжину? Довжина – це фізична величина, яку водій бачить на одометрі. Якщо штучно подовжити ребро через штраф, то загальна відстань маршруту стане нереалістичною. Але для алгоритму важлива не абсолютна довжина, а порівняння різних шляхів. Якщо одне ребро має довжину 15 км і штраф 20, його «вартість» 35 км, а інше – 25 км без штрафу. Алгоритм обере друге, хоча реально проїде 25 км замість 15.

Це означає, що водій поїде довшою, але бажанішою дорогою. Штраф – це інструмент пріоритезації, а не точна фізична величина.

Крім того, функція вартості може бути складнішою. Наприклад, можна додати множник для часу: cost=α·length+β·time+γ·penalty. Але це вже багатокритеріальна оптимізація, яка потребує налаштування коефіцієнтів. У роботі використовується простий додаток довжини та штрафу, що дозволяє зберігати інтуїтивну інтерпретацію.

Зважений граф разом з функцією вартості стає основою для всіх подальших обчислень. Алгоритм пошуку шляху отримує на вхід саме ці значення, і його завдання: мінімізувати сумарну вартість маршруту. У наступному підрозділі розглянемо, як саме в модель закладаються транспортні обмеження, які неможливо виразити через просту зміну ваги.

2.6 Моделювання транспортних обмежень

У попередніх підрозділах йшлося про ваги ребер і функцію вартості, але це лише частина інформації, яку потрібно закласти в модель. Для негабаритних перевезень критичними є обмеження, які безпосередньо впливають на саму можливість руху. Їх не можна звести до простого збільшення вартості, вони мають пороговий характер. Тому модель графа розширюють додатковими параметрами кожного ребра.

Жорсткі обмеження описуються граничними значеннями. Наприклад, для ділянки дороги задають максимально допустиму висоту h_{max} та ширину w_{max}. Під час пошуку маршруту для конкретного транспортного засобу з параметрами (h,w)(h,w) ребро вважається доступним лише якщо виконуються умови:

$$h \leq h_{max} \text{ і } w \leq w_{max}$$

Якщо хоча б одна нерівність порушується, ребро виключається з розгляду. Таке моделювання дозволяє відкидати непридатні ділянки на

ранньому етапі. До жорстких обмежень також можна віднести обмеження за довжиною транспортного засобу, масою або осьовим навантаженням, але в реалізованому застосунку для спрощення використано лише висоту та ширину.

М'які обмеження не забороняють рух, але роблять його небажаним. У моделі вони реалізуються через штраф penalty(e), який додається до вартості ребра. Наприклад, проїзд центральною частиною міста може мати штраф 20 умовних одиниць. Тоді навіть якщо фізична довжина такого ребра невелика, сумарна вартість зростає, і алгоритм надасть перевагу об’їзному шляху з меншим або нульовим штрафом. Важливо, що штрафи не впливають на жорстку допустимість, ребро залишається в графі в будь-якому разі [9].

Обидва типи обмежень прив’язуються безпосередньо до ребер, оскільки саме ділянка дороги є носієм таких характеристик. У реальності обмеження можуть змінюватися вздовж одного шосе (наприклад, міст із меншою висотою). Для коректного моделювання в таких місцях додають проміжні вершини, розбиваючи ребро на частини. Кожна частина отримує власні параметри. Це дозволяє точно передати локальні обмеження, не втрачаючи загальної структури графа [10].

Окремий випадок – динамічні обмеження, які залежать від часу доби або дня тижня. У статичній моделі, яка використовується, такі обмеження не враховуються. Але за потреби їх можна наближено змоделювати, наприклад, у нічний час змінювати штрафи або тимчасово вилучати окремі ребра. Це вже виходить за межі поточного дослідження.

Отже, запропоноване моделювання обмежень, через пару жорстких параметрів і додатковий штраф є достатнім для багатьох практичних задач і водночас більш менш простим для реалізації. Воно дозволяє зберігати єдину структуру графа і використовувати модифікований алгоритм пошуку без зміни його базової логіки. У наступному розділі буде показано, як саме ці обмеження інтегруються в алгоритм Дейкстри та в процедуру побудови альтернативних маршрутів.

3 РОЗРОБКА МЕТОДУ ПОБУДОВИ МАРШРУТІВ

3.1 Модифікація алгоритму Дейкстри

Класичний алгоритм Дейкстри використовується для пошуку найкоротшого шляху у зважених графах та є одним із найбільш поширених алгоритмів маршрутизації [6]. Основний принцип його роботи полягає у послідовному виборі вершини з мінімальною поточною вагою та поступовому оновленні відстаней до сусідніх вершин.

У задачах негабаритних перевезень стандартна реалізація алгоритму має певні обмеження, оскільки під час побудови маршруту необхідно враховувати параметри транспортного засобу та характеристики дорожньої мережі. Для вирішення цієї проблеми у роботі використовується модифікований алгоритм Дейкстри, який додатково виконує перевірку транспортних обмежень та враховує додаткові параметри ребер графа.

Загальна схема роботи модифікованого алгоритму наведена на рисунку 3.1.

На початковому етапі алгоритм виконує ініціалізацію ваг вершин графа. Для стартової вершини встановлюється нульове значення відстані, а для інших вершин використовується умовно нескінченне значення. Після цього стартова вершина додається до черги пріоритетів.

Під час кожної ітерації алгоритм вибирає вершину з мінімальною поточною вагою маршруту та виконує перегляд усіх сусідніх ребер. Для кожної дорожньої ділянки виконується перевірка транспортних обмежень, зокрема допустимої висоти та ширини.

Якщо параметри транспортного засобу перевищують допустимі характеристики ребра, така ділянка виключається з подальшого пошуку. У програмній реалізації це виконується шляхом пропуску ребра без оновлення ваги маршруту.

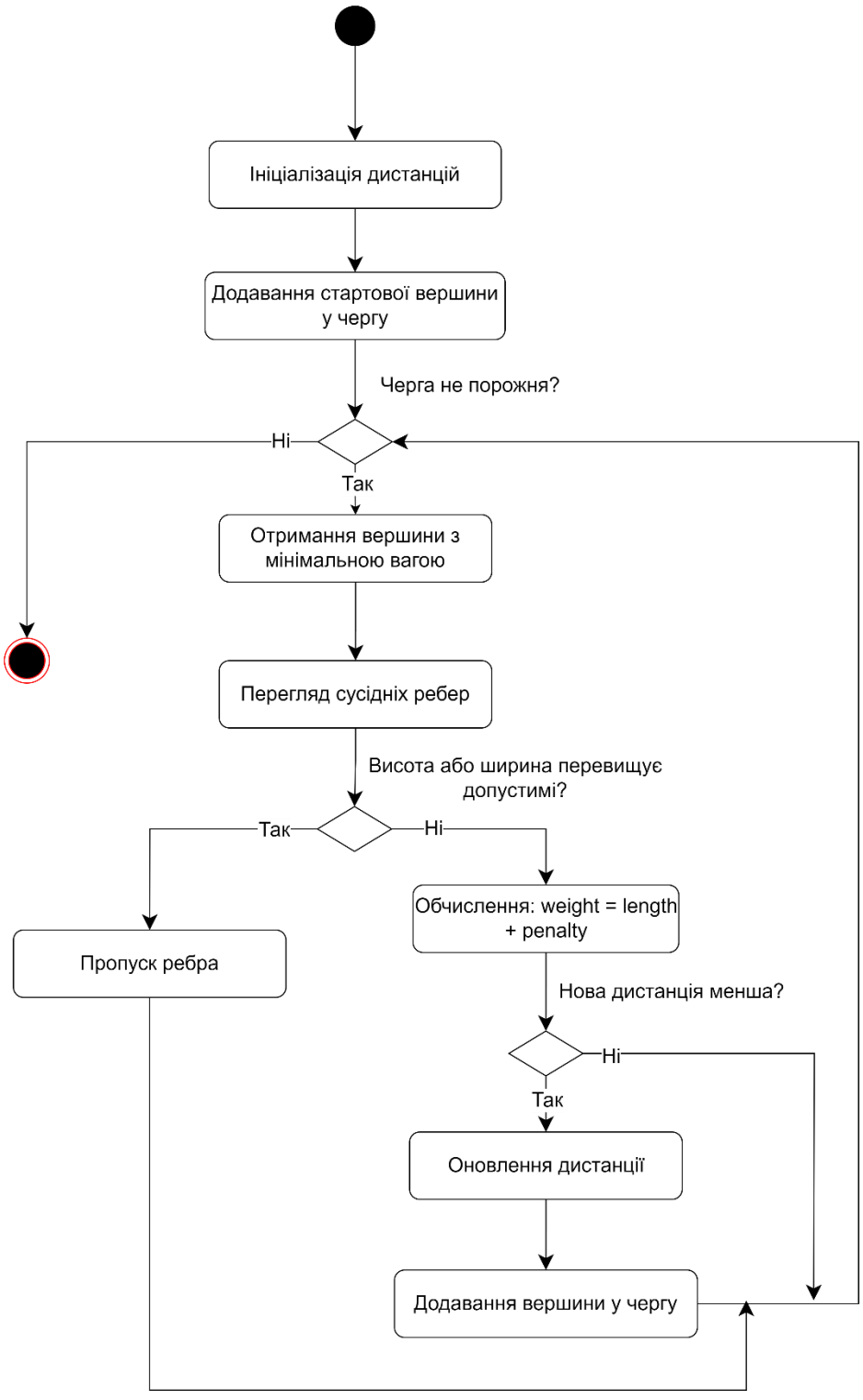

Рисунок 3.1 – Схема роботи модифікованого алгоритму Дейкстри

Для доступних ребер алгоритм обчислює нову вагу маршруту та виконує порівняння отриманого значення з поточною відстанню до сусідньої вершини.

Якщо новий маршрут є коротшим, виконується оновлення дистанції та повторне додавання вершини до черги пріоритетів.

Процес пошуку повторюється до моменту завершення обробки всіх доступних вершин графа. Після цього виконується відновлення маршруту шляхом послідовного проходження між попередніми вершинами.

Модифікація алгоритму Дейкстри дозволила адаптувати класичний алгоритм пошуку найкоротшого шляху до задач маршрутизації негабаритного транспорту та забезпечити врахування транспортних обмежень без суттєвого ускладнення алгоритму.

3.2 Врахування жорстких та м’яких обмежень

Однією з основних особливостей маршрутизації негабаритного транспорту є необхідність врахування транспортних обмежень дорожньої мережі. Для окремих ділянок дороги можуть існувати обмеження висоти, ширини або максимально допустимого навантаження, через що певні маршрути стають недоступними для великогабаритного транспорту.

У розробленому методі використовується механізм жорстких обмежень, який дозволяє виключати непридатні дорожні ділянки ще на етапі пошуку маршруту. Загальна схема перевірки жорстких обмежень наведена на рисунку 3.2.

На початковому етапі алгоритм отримує параметри поточного ребра графа та порівнює їх із характеристиками транспортного засобу. Насамперед виконується перевірка максимально допустимої висоти дороги.

Якщо висота транспортного засобу перевищує допустиме значення, ребро виключається з подальшого пошуку маршруту. Аналогічна перевірка виконується для ширини транспортного засобу.

У програмній реалізації перевірка виконується безпосередньо під час обробки сусідніх ребер графа. Якщо хоча б одне з обмежень порушується, алгоритм переходить до аналізу наступного ребра без оновлення маршруту.

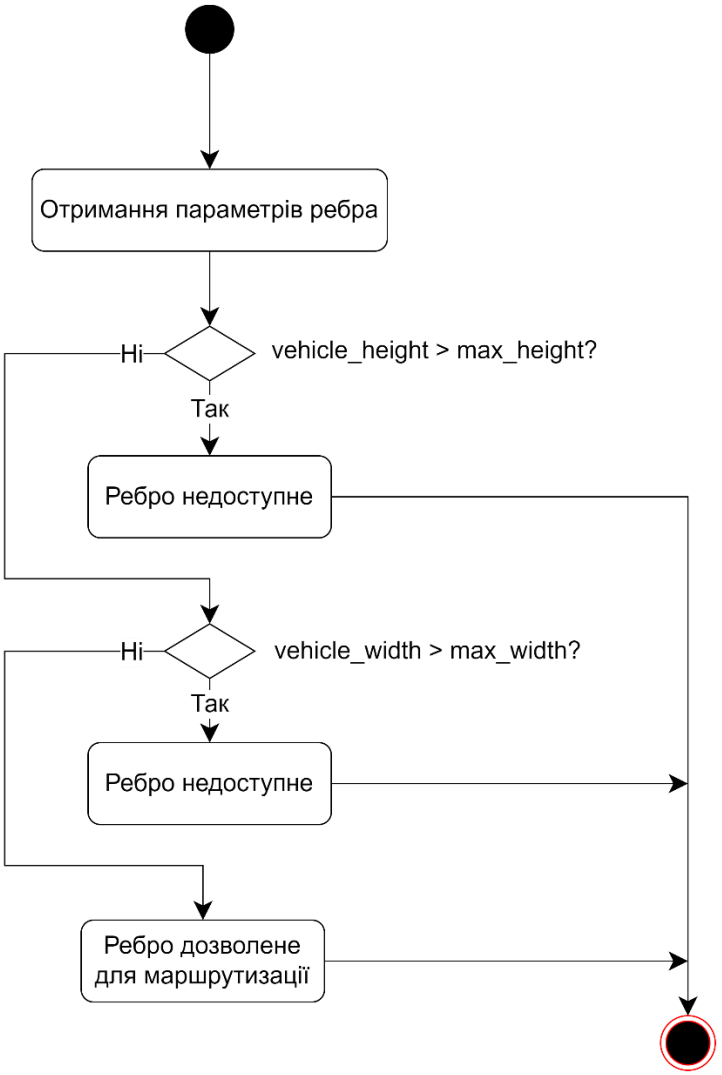

Рисунок 3.2 – Схема перевірки жорстких транспортних обмежень

Такий підхід дозволяє уникнути побудови непридатних маршрутів та забезпечує формування лише допустимих шляхів для негабаритного транспорту.

Перед тим, як переходити до програмної перевірки обмежень, варто узагальнити, які саме типи обмежень взагалі існують у транспортних задачах для негабаритних перевезень. Під час аналізу літератури та реальних логістичних сценаріїв було виявлено, що всі обмеження можна поділити на кілька категорій залежно від їхньої природи. Деякі з них є жорсткими (фізично забороняють рух), інші – м’якими (лише впливають на пріоритет). Зведена класифікація подана в таблиці 3.1.

Таблиця 3.1 – Класифікація транспортних обмежень для негабаритних перевезень

№	Тип обмеження	Приклад у реальності	Характер
1	Висота	Низький міст (4,0 м), тунель, дроти ЛЕП	Жорстке
2	Ширина	Вузький проїзд, шлагбауми, звуження дороги	Жорстке
3	Довжина автопоїзда	Різкий поворот, серпантин, кільцева розв’язка	Жорстке
4	Маса (загальна або на вісь)	Міст із вантажопідйомністю 25 т, ґрунтова дорога	Жорстке
5	Часові вікна	Заборона в’їзду в місто з 7:00 до 20:00	М’яке (або жорстке)
6	Екологічні зони	Зона з низькими викидами (Євро-5)	М’яке (штраф)
7	Радіус повороту	Вантажівка не впишеться в кут 90°	Жорстке
8	Супровід / дозволи	Обов’язковий патруль ДАІ	Організаційне
9	Проїзд через центр міста	Пробки, вузькі вулиці, пішохідні зони	М’яке (штраф)
10	Стан покриття	Ґрунтова дорога, вибоїни	М’яке (штраф)

Було свідомо реалізувано лише найбільш критичні для негабариту типи: висоту, ширину (жорсткі) та штраф за небажаний проїзд (м’яке). Інші обмеження (довжина, маса, часові вікна) можна додати в майбутньому шляхом розширення структури ребра – це не потребує зміни логіки самого алгоритму. Наприклад, перевірка маси виконувалася б аналогічно до висоти.

Такий підхід дозволяє зосередитися на головному – модифікації алгоритму Дейкстри – і водночас залишає простір для подальшого вдосконалення системи.

3.3 Використання штрафних функцій

Навіть коли ділянка дороги формально доступна (жорсткі обмеження не порушені), вона може бути небажаною через інші причини: затори, вузькі

вулиці, проїзд через центр міста чи погану якість покриття. У таких випадках замість повного виключення ребра доцільно лише збільшити його вартість – це називається штрафною функцією (penalty). Алгоритм отримує можливість використати небажану ділянку, якщо немає кращої альтернативи, але за інших рівних умов обиратиме шлях із меншим штрафом.

У запропонованому методі штраф реалізований як додатковий параметр penalty для кожного ребра графа. Під час роботи модифікованої Дейкстри підсумкова вага ребра обчислюється за формулою наведеною у підпункт 2.5:

$$\text{weight} = \text{length} + \text{penalty}$$

де length – фізична довжина ділянки (км), а penalty – штраф (теж у кілометрах, хоча насправді це безрозмірний коефіцієнт, який додають до загальної вартості маршруту). Логіку роботи штрафної функції схематично показано на рисунку 3.3.

Приклад: нехай є два ребра, що з’єднують ті самі вершини (або два різні маршрути):

1. Ребро А: довжина 15 км, penalty = 20 (проходить через центр міста з інтенсивним рухом).
2. Ребро Б: довжина 25 км, penalty = 0 (об’їзна дорога).

Класичний алгоритм Дейкстри (без штрафів) обрав би ребро А, бо 15 < 25. Але зі штрафом вартість ребра А стає 15+20 = 35, а ребра Б – 25. Тому модифікований алгоритм обере довший фізично, але дешевший за «сумарною вартістю» маршрут Б. Користувач побачить реальну відстань 25 км, а штраф (20 км) буде показано окремо в деталях як додаткову умовну величину.

Жодних додаткових перевірок штраф не потребує – він просто збільшує числове значення. Завдяки цьому механізм легко налаштовувати: для звичайних трас penalty = 0, для небажаних ділянок можна встановити будь-яке додатне число (наприклад, 10, 20 або 50 залежно від ступеня небажаності). У нашій тестовій мережі штрафи використано для ділянки «Дніпро_1 –

Запоріжжя_пн» (міст із обмеженою висотою, penalty=10) та для ребра через центр міста (penalty=20 при теоретичному моделюванні).

Рисунок 3.3 – Схема використання штрафних функцій

Таким чином, штрафні функції дають змогу гнучко керувати пріоритетами маршрутизації, не ускладнюючи алгоритм. Далі розглянемо, як контекстний аналіз обмежень інтегрує всі ці механізми в єдиний пошук.

3.4 Контекстний аналіз транспортних обмежень

Під час побудови маршрутів для негабаритного транспорту недостатньо враховувати лише базову довжину дорожніх ділянок. Окремі дороги можуть мати додаткові характеристики, які впливають на доцільність їх використання під час перевезення великогабаритних вантажів. Для врахування таких особливостей у роботі використовується контекстний аналіз транспортних обмежень.

У межах розробленого методу контекстний аналіз реалізується шляхом оцінки параметрів ребер графа та врахування додаткових характеристик дорожніх ділянок під час формування ваг маршруту. На відміну від жорстких обмежень, які повністю виключають ребра з графа, контекстний аналіз дозволяє зменшувати пріоритет окремих доріг без їх повного блокування.

Загальна схема контекстного аналізу транспортних обмежень наведена на рисунку 3.4.

На першому етапі алгоритм отримує параметри поточного ребра графа. Для кожної дорожньої ділянки аналізуються характеристики, які можуть впливати на побудову маршруту. До таких параметрів належать довжина дороги, наявність додаткових штрафних коефіцієнтів та допустимі транспортні характеристики.

Після отримання параметрів ребра виконується перевірка жорстких транспортних обмежень. Якщо дорожня ділянка є недоступною для руху негабаритного транспорту, ребро виключається з подальшого пошуку маршруту.

Для доступних ребер система виконує аналіз додаткових характеристик дорожньої ділянки. Якщо для ребра задано штрафний коефіцієнт, його значення враховується під час формування підсумкової ваги маршруту. У результаті дороги з менш бажаними характеристиками отримують більшу вагу та рідше використовуються під час побудови оптимального маршруту.

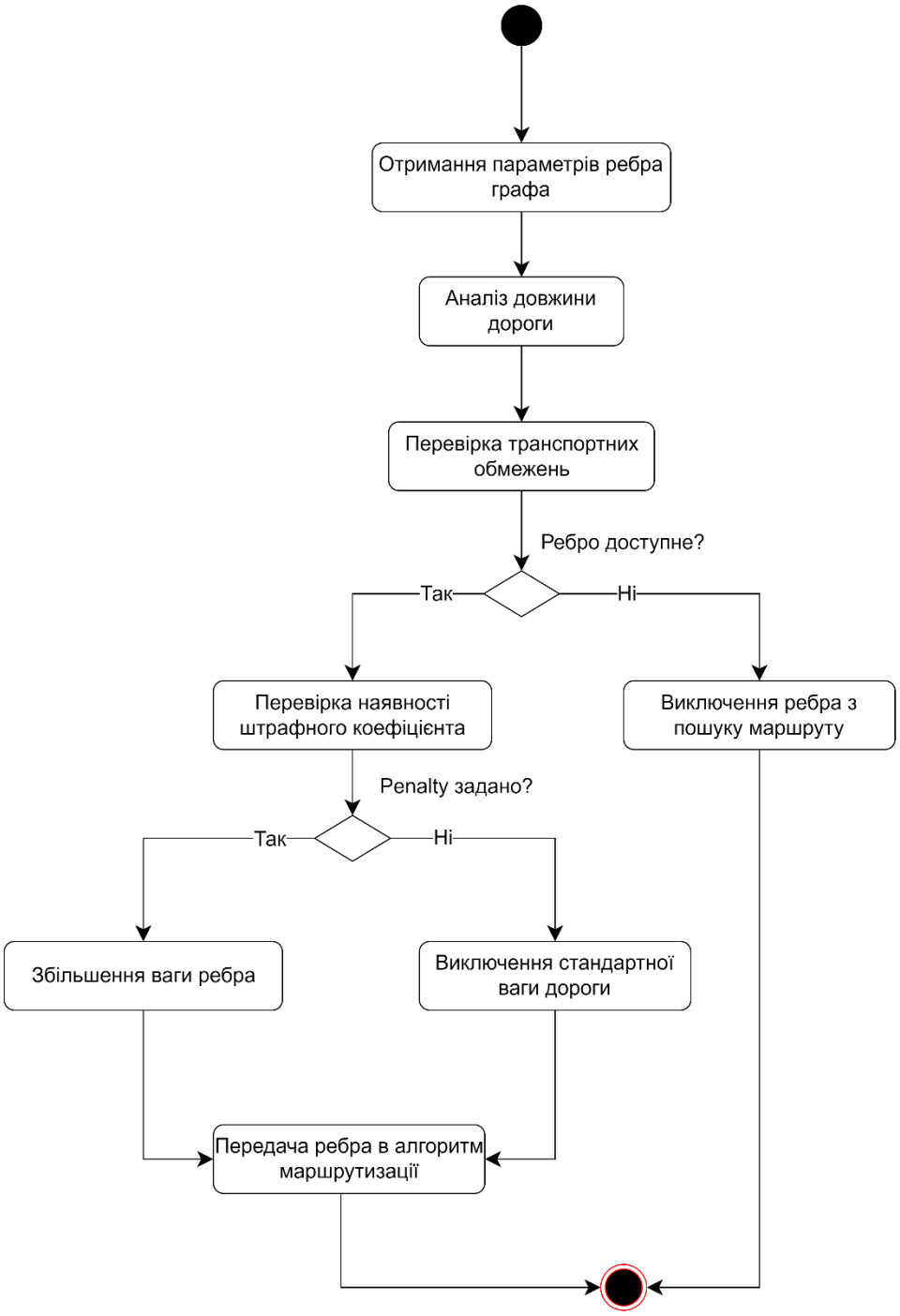

Рисунок 3.4 – Схема контекстного аналізу транспортних обмежень

Такий підхід дозволяє алгоритму не лише визначати допустимі маршрути, але й формувати більш придатні шляхи для негабаритних перевезень. При цьому система зберігає можливість використання альтернативних доріг у випадку відсутності кращих варіантів маршрутизації.

Контекстний аналіз у розробленому методі реалізований на основі

параметрів графової моделі транспортної мережі та інтегрований безпосередньо у процес роботи модифікованого алгоритму Дейкстри. Це дозволяє забезпечити узгоджену роботу механізмів перевірки обмежень, штрафних функцій та побудови альтернативних маршрутів.

3.5 Побудова альтернативних маршрутів

У процесі маршрутизації негабаритного транспорту використання лише одного маршруту не завжди є достатнім. Під час перевезення можуть виникати ситуації, коли окремі дорожні ділянки стають недоступними або менш придатними для руху. Для підвищення гнучкості системи у роботі реалізовано механізм побудови альтернативних маршрутів.

Схема формування альтернативних маршрутів наведена на рисунку 3.5.

На першому етапі система будує основний маршрут між початковою та кінцевою вершинами графа. Отриманий шлях додається до списку результатів та використовується як базовий маршрут.

Після цього алгоритм виконує вибір spur node, проміжної вершини маршруту, від якої буде формуватися альтернативний шлях. Для уникнення дублювання маршрутів частина ребер основного шляху тимчасово блокується.

Далі повторно запускається модифікований алгоритм Дейкстри, який виконує пошук нового маршруту між вершинами графа. Якщо новий шлях успішно знайдено, система перевіряє його унікальність.

Унікальні маршрути додаються до списку альтернативних шляхів, після чого процес повторюється для інших spur node.

Рисунок 3.5 – Схема побудови альтернативних маршрутів

Розглянемо типовий сценарій, треба доставити негабаритний вантаж із Харкова до Львова. Параметри транспортного засобу: висота 4,5 м, ширина 3,0 м. Транспортна мережа між обласними центрами подана у вигляді

агрегованого графа (кожне ребро – магістральна дорога). На основі реальних відстаней між містами сформовано три можливі шляхи.

Маршрут 1 (основний) – північний, через Київ:

Склад: Харків → Полтава → Лубни → Київ → Житомир → Звягель → Рівне → Львів.

Особливість: найкоротший за фізичною відстанню, але може мати проблемні ділянки.

Таблиця 3.2 – Маршрут 1 (Харків – Львів, північний через Київ)

Ділянка	Довжина (км)	Примітка
Харків – Полтава	143	траса М03
Полтава – Лубни	110	траса М03
Лубни – Київ	198	траса М03/Е40
Київ – Житомир	140	траса М06/Е40
Житомир – Звягель	87	траса М06/Е40
Звягель – Рівне	99	траса М06/Е40
Рівне – Львів	211	траса М06
Усього	988	

Візуально цей маршрут показано на рисунку 3.6. Червоною стрілкою позначено основний шлях, який проходить через міста: Харків → Полтава → Лубни → Київ → Житомир → Звягель → Рівне → Львів.

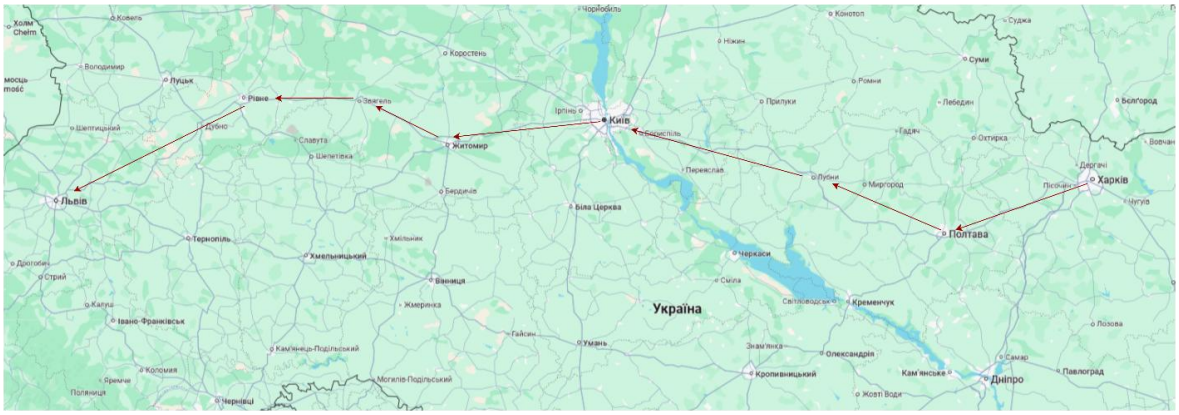

Рисунок 3.6 – Маршрут 1 (Харків–Львів, північний через Київ) на карті

Маршрут 2 (перша альтернатива) – північно-східний, через Суми:

Склад: Харків → Суми → Ромни → Прилуки → Київ → Житомир → Звягель → Рівне → Львів.

Особливість: трохи довший за основний, але може бути корисним, якщо ділянка Харків–Полтава завантажена.

Таблиця 3.3 – Маршрут 2 (Харків – Львів, північно-східний через Суми)

Ділянка	Довжина (км)	Примітка
Харків – Суми	184	траса Н12
Суми – Ромни	102	траса Н07
Ромни – Прилуки	84	траса Н07
Прилуки – Київ	149	траса Т-10-25 / Е40
Київ – Житомир	140	траса М06/Е40
Житомир – Звягель	87	траса М06/Е40
Звягель – Рівне	99	траса М06/Е40
Рівне – Львів	211	траса М06
Усього	1056	

Цей альтернативний маршрут зображено на рисунку 3.7. Черконим кольором виділено шлях через Суми, Ромни, Прилуки, далі до Києва та на захід.

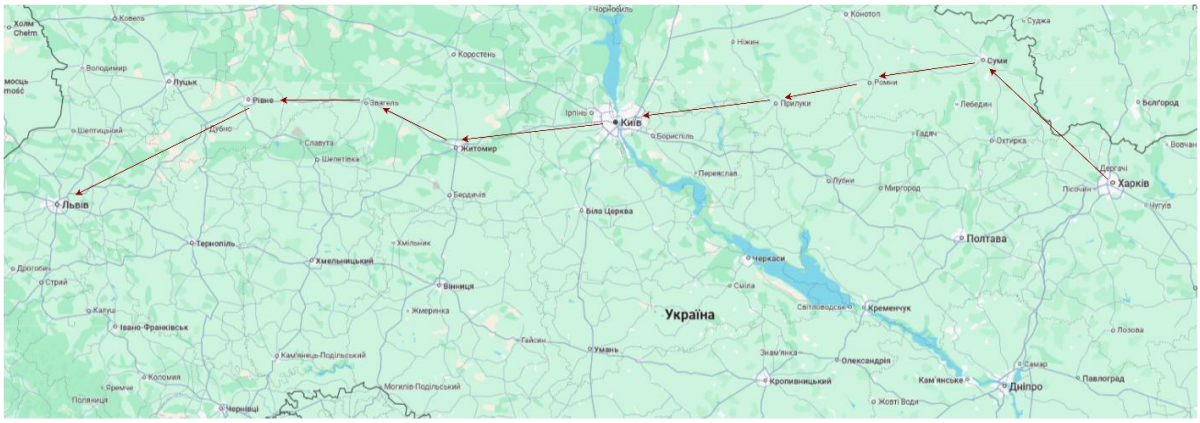

Рисунок 3.7 – Маршрут 2 (Харків–Львів, північно-східний через Суми) на карті

Маршрут 3 (друга альтернатива) – південний, в обхід Києва:

Склад: Харків → Полтава → Кременчук → Олександрія → Кропивницький → Умань → Вінниця → Хмельницький → Тернопіль → Львів.

Особливість: найдовший, але повністю уникає ділянки Київ–Житомир, де може бути обмеження висоти. Підходить для ТЗ, які не можуть проїхати міст під Києвом.

Таблиця 3.4 – Маршрут 3 (Харків – Львів, південний в обхід Києва)

Ділянка	Довжина (км)	Примітка
Харків – Полтава	143	траса М03
Полтава – Кременчук	113	траса М22
Кременчук – Олександрія	58	траса М22 / Н14
Олександрія – Кропивницький	65	траса Н14 / М30
Кропивницький – Умань	166	траса М30 / Е50
Умань – Вінниця	160	траса М30
Вінниця – Хмельницький	120	траса М30
Хмельницький – Тернопіль	113	траса М30
Тернопіль – Львів	130	траса М30
Усього	1068	

На рисунку 3.8 червоною стрілкою показано південний обхідний маршрут, який уникає ділянки Київ–Житомир.

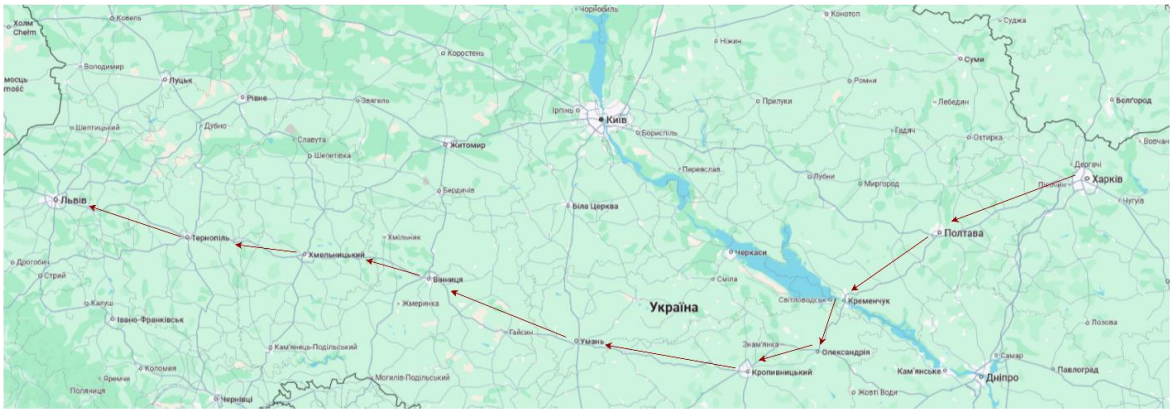

Рисунок 3.8 – Маршрут 3 (Харків–Львів, південний в обхід Києва) на карті

Зведемо дані в одну таблицю:

Таблиця 3.5 – Порівняння трьох маршрутів Харків – Львів

Маршрут	Фізична довжина (км)	Коли обирати
1 (північний, через Київ)	988	Основний, найкоротший варіант
2 (північно-східний, через Суми)	1056	Коли траса Харків–Полтава перевантажена
3 (південний, в обхід Києва)	1068	Коли ділянка Київ–Житомир має жорстке обмеження (наприклад, низький міст)

Загальний вигляд усіх трьох маршрутів на одній карті подано на рисунку 3.9. Стрілкою показано різницю в просторовому прокладанні.

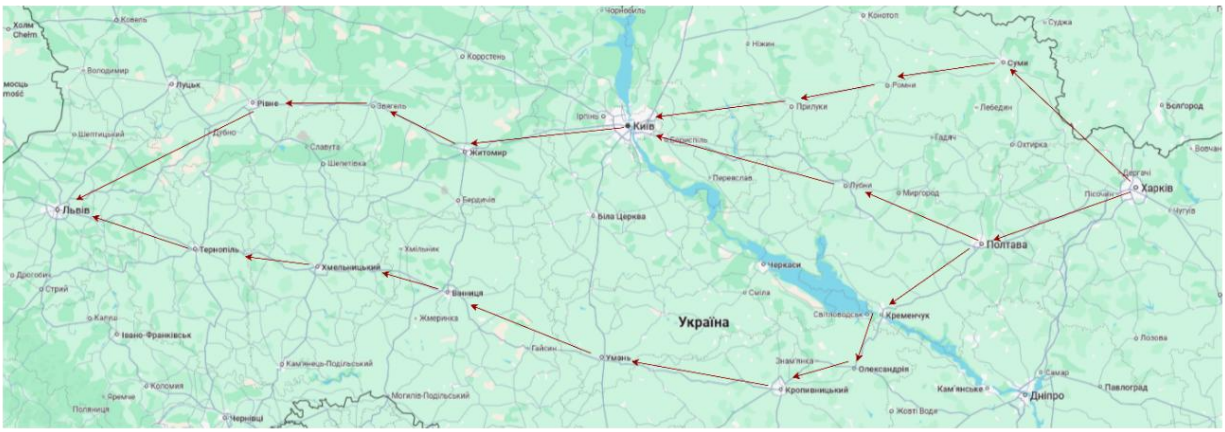

Рисунок 3.9 – Порівняння трьох маршрутів Харків–Львів (суміщена карта)

Як бачимо, найкоротший за фізичною відстанню – маршрут 1 (988 км). Маршрут 2 довший на 68 км, але уникає потенційних проблем на ділянці Харків–Полтава (наприклад, якщо там ремонт або затор). Маршрут 3 – найдовший (1068 км, на 80 км більше за перший), але безпечний з погляду обмежень. Саме третій маршрут алгоритм обере, якщо висота ТЗ перевищує допустиму для мосту на ділянці Київ–Житомир, або якщо на цій ділянці встановлено жорсткий штраф.

Як алгоритм знаходить ці альтернативи покроково приведено нижче.

1. Модифікована Дейкстра будує маршрут 1 (північний через Київ).

2. Для пошуку альтернативи алгоритм Єна вибирає точку відхилення, наприклад, Полтаву. Ребро Полтава–Лубни тимчасово блокується.

3. Запускається нова Дейкстра від Полтави до Львова. Вона знаходить шлях Полтава → Харків → Суми → ... → Львів – це маршрут 2 (через Суми). Зверніть увагу: довелося повернутися назад у Харків, що робить цей маршрут нелогічним для реального життя. Насправді, для коректної роботи алгоритму варто було б заблокувати не ребро Полтава–Лубни, а ребро Полтава–Київ, але це вже питання точності моделі. На практиці логісти так не їздять, але алгоритм показує принципову можливість знайти інший шлях.

4. Для третьої альтернативи ми блокуємо критичну ділянку Київ–Житомир (наприклад, якщо там міст низький). Тоді Дейкстра прокладає маршрут 3 – південний обхід через Кременчук, Кропивницький, Вінницю та Тернопіль.

Усі три маршрути зберігаються в списку, сортуються за довжиною та передаються на візуалізацію. Користувач бачить на карті три лінії (основний маршрут – червоним, альтернативи – синім та зеленим). Через використання агрегованої моделі графа лінії між містами на карті будуть прямими – це особливість спрощення, а не помилка.

Отже, побудова альтернативних маршрутів дозволяє отримати не один, а кілька допустимих варіантів перевезення. Логіст може вибрати найкращий за своїми критеріями (час, вартість, надійність) або використати резервний шлях, якщо основний стане недоступним.

3.6 Алгоритм k-найкоротших шляхів (алгоритм Єна)

Для побудови альтернативних маршрутів у роботі використовується алгоритм Єна [11], який дозволяє знаходити декілька найкоротших шляхів між вершинами графа. Алгоритм базується на послідовному формуванні альтернативних маршрутів шляхом часткового виключення ділянок уже

знайдених шляхів.

Використання алгоритму Єна дозволяє отримати множину маршрутів, які відрізняються між собою та можуть використовуватися як альтернативні варіанти перевезення [11].

Загальна схема роботи алгоритму Єна наведена на рисунку 3.10.

На першому етапі алгоритм виконує пошук найкоротшого маршруту між початковою та кінцевою вершинами графа. Знайдений шлях додається до списку основних маршрутів.

Після цього виконується ітеративний пошук альтернативних маршрутів. Для кожного маршруту вибирається spur node – проміжна вершина, яка використовується для формування нового шляху. Частина ребер root path тимчасово виключається з графа, після чого запускається модифікований алгоритм Дейкстри для пошуку spur path.

Якщо новий шлях успішно знайдено, root path та spur path об’єднуються у candidate path. Після цього маршрут додається до списку кандидатних шляхів.

Серед отриманих candidate path вибирається маршрут з найменшою вагою, який додається до списку основних результатів. Далі процес повторюється до отримання необхідної кількості альтернативних маршрутів.

Використання алгоритму Єна дозволяє реалізувати побудову декількох маршрутів без повного перебору всіх можливих шляхів графа. Це зменшує складність обчислень та забезпечує ефективну генерацію альтернативних маршрутів для задач негабаритних перевезень.

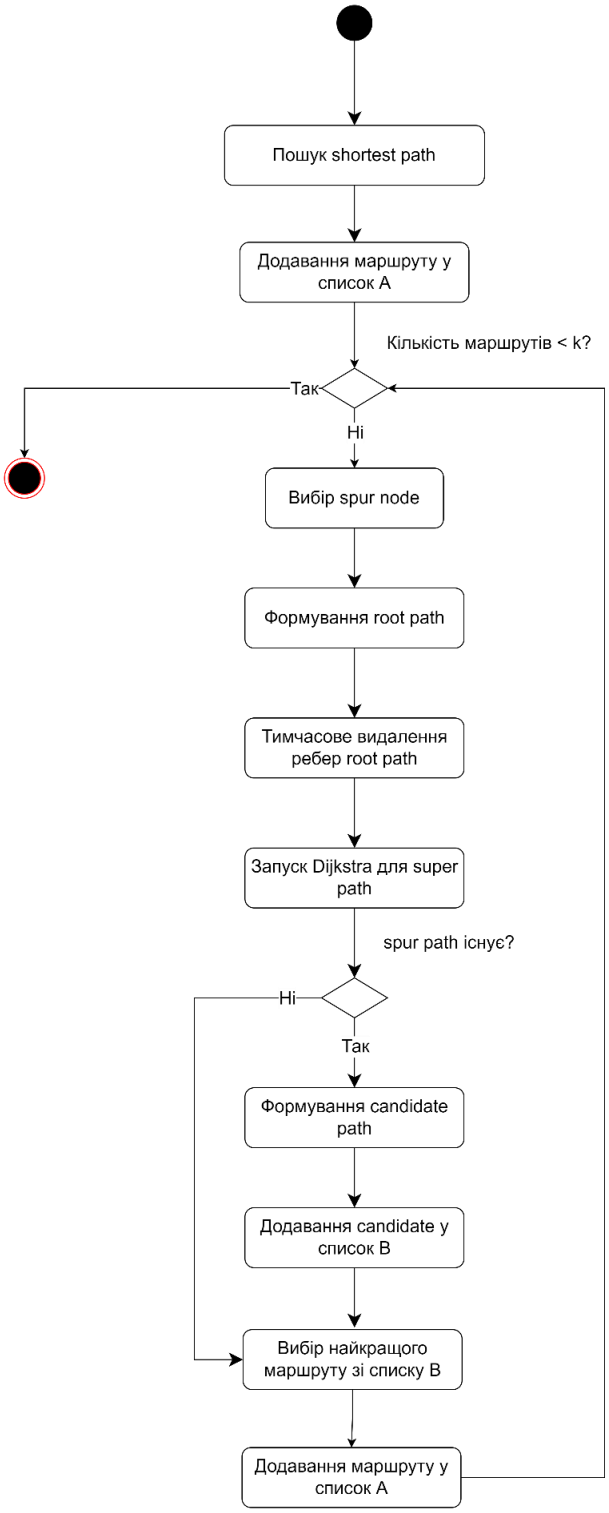

Рисунок 3.10 – Схема роботи алгоритму Єна

3.7 Загальна схема роботи методу

Розроблений метод побудови маршрутів для негабаритних перевезень

об’єднує графову модель транспортної мережі, модифікований алгоритм Дейкстри, механізм перевірки транспортних обмежень та алгоритм побудови альтернативних маршрутів. Основною метою методу є формування маршруту, який враховує параметри транспортного засобу та особливості дорожньої мережі.

Транспортна мережа у системі представлена у вигляді зваженого графа, де вершини відповідають населеним пунктам або транспортним вузлам, а ребра, дорогам між ними. Для кожного ребра задаються параметри довжини, допустимої висоти, ширини та додаткових штрафів. Використання агрегованої графової моделі дозволяє спростити представлення дорожньої мережі та зменшити складність обчислень. При цьому геометрія маршрутів може відображатися у спрощеному вигляді у формі прямих ліній між вершинами графа, що є особливістю агрегованої моделі та не впливає на логіку роботи алгоритму.

Загальна послідовність роботи методу наведена на рисунку 3.11.

На першому етапі відбувається завантаження графа транспортної мережі та ініціалізація параметрів транспортного засобу. Користувач задає початкову та кінцеву точки маршруту, а також характеристики негабаритного транспорту, зокрема допустиму висоту та ширину.

Після цього запускається модифікований алгоритм Дейкстри, який виконує пошук оптимального шляху між вершинами графа. На відміну від класичного алгоритму, під час обробки кожного ребра додатково виконується перевірка транспортних обмежень. Якщо параметри транспортного засобу перевищують допустимі значення для певної дороги, таке ребро виключається з подальшого пошуку маршруту.

У випадку, коли ребро задовольняє жорсткі обмеження, виконується обчислення ваги маршруту. Вага ребра формується на основі довжини дороги та додаткового штрафу, який дозволяє враховувати небажані характеристики дорожньої ділянки. Таким чином, навіть якщо дорога доступна для руху, алгоритм може надавати перевагу більш безпечним або оптимальним

маршрутам.

Після завершення роботи алгоритму Дейкстри формується основний маршрут. Далі система запускає алгоритм Єна для побудови альтернативних маршрутів. На цьому етапі окремі ділянки основного маршруту тимчасово виключаються з графа, після чого повторно виконується пошук нового шляху. Такий підхід дозволяє отримати декілька альтернативних маршрутів між початковою та кінцевою точками.

Отримані маршрути передаються до модуля візуалізації, де відображаються на карті. Основний маршрут та альтернативні шляхи можуть відображатися різними кольорами, що спрощує аналіз результатів роботи алгоритму.

Запропонований метод дозволяє враховувати транспортні обмеження, формувати альтернативні маршрути та виконувати пошук шляхів у графовій транспортній мережі з урахуванням параметрів негабаритного транспорту.

Рисунок 3.11 – Загальна схема роботи методу побудови маршрутів

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Архітектура системи та вибір технологій

Для реалізації системи побудови альтернативних маршрутів для негабаритних перевезень було обрано клієнт-серверну архітектуру вебзастосунку. Такий підхід дозволяє розділити логіку обробки маршрутів, роботу алгоритмів та інтерфейс користувача, що забезпечує зручність підтримки, масштабованість та можливість подальшого розвитку системи. Структура програмного проєкту представлена на рисунку 4.1.

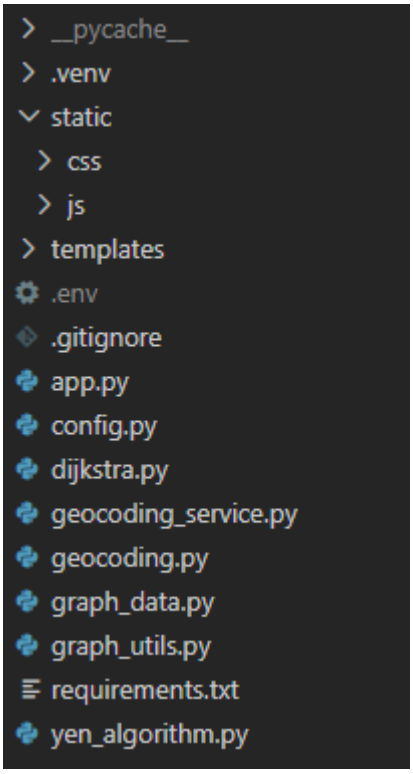

Рисунок 4.1 – Структура програмного проекту

У розробленому застосунку серверна частина відповідає за обробку запитів користувача, запуск алгоритмів маршрутизації, перевірку транспортних обмежень та формування результатів побудови маршрутів. Клієнтська частина забезпечує взаємодію користувача із системою та

візуалізацію отриманих маршрутів на карті.

Основним ядром системи є модуль маршрутизації, який реалізує модифікований алгоритм Дейкстри та алгоритм Єна для побудови альтернативних маршрутів. Для роботи алгоритмів використовується графова модель транспортної мережі, у якій вершини відповідають транспортним вузлам, а ребра містять інформацію про дорожні сегменти та транспортні обмеження.

Окремим компонентом системи є модуль геокодування, який забезпечує перетворення текстових адрес у географічні координати. Це дозволяє користувачу вводити не лише готові транспортні вузли, а й довільні адреси.

У якості основної мови програмування було обрано Python. Дана мова має простий синтаксис, підтримує об’єктно-орієнтований підхід та містить велику кількість бібліотек для реалізації алгоритмів, роботи з графами та веброзробки. Крім того, Python дозволяє швидко реалізовувати та тестувати алгоритмічні рішення.

Для створення серверної частини застосунку було використано фреймворк Flask [13]. Його вибір обумовлений невеликою складністю, високою гнучкістю та зручністю створення REST API. Flask добре підходить для невеликих та середніх проєктів, у яких необхідна швидка реалізація серверної логіки.

Для візуалізації маршрутів використовується сервіс Google Maps API [14]. Використання даного сервісу дозволяє відображати маршрути на інтерактивній карті, працювати з географічними координатами та реалізувати підтримку геокодування адрес.

У системі також використовується модульна структура проєкту [15]. Основні компоненти винесені в окремі файли, що спрощує підтримку коду та забезпечує кращу організацію програмної логіки. Зокрема, окремо реалізовано:

– модуль роботи з графом транспортної мережі;

– модуль модифікованого алгоритму Дейкстри;

– модуль алгоритму альтернативних маршрутів;

– модуль геокодування;

– конфігураційний модуль;

– серверний модуль обробки запитів.

Обрана архітектура системи забезпечує модульність, розширюваність та ефективну взаємодію між компонентами застосунку. Використання сучасних технологій веброзробки дозволило створити систему, здатну виконувати побудову маршрутів для негабаритних перевезень із врахуванням транспортних обмежень та надавати користувачу декілька альтернативних варіантів маршруту.

4.2 Реалізація графа

Основою роботи системи побудови маршрутів є транспортна мережа, представлена у вигляді графа. У розробленому застосунку граф використовується для моделювання дорожньої інфраструктури та забезпечення роботи алгоритмів пошуку маршрутів.

Граф складається з вершин та ребер. Вершини відповідають містам або проміжним транспортним точкам, а ребра описують дороги між ними. Для кожного ребра зберігається інформація про довжину ділянки дороги, максимально допустиму висоту та ширину транспортного засобу, а також додатковий штраф, який враховується при побудові маршруту.

Реалізація структури графа виконана засобами мови Python у файлі graph_data.py. Для представлення вершин та ребер були створені окремі класи. Приклад реалізації класів наведено у лістингу 4.1.

Лістинг 4.1 – Реалізація структури вершини та ребра графа

```
class Vertex:
    def __init__(self, id, name, lat, lng, is_city=True):
        self.id = id
        self.name = name
```

```
        self.lat = lat
        self.lng = lng
        self.is_city = is_city
class Edge:
    def __init__(self, from_vertex, to_vertex,
                 length_km, max_height,
                 max_width, penalty=0):
        self.from_vertex = from_vertex
        self.to_vertex = to_vertex
        self.length = length_km
        self.max_height = max_height
        self.max_width = max_width
        self.penalty = penalty
```

У лістингу 4.1 клас Vertex використовується для збереження інформації про вершину графа. Кожна вершина містить унікальний ідентифікатор, назву, географічні координати та ознаку того, чи є вершина містом.

Клас Edge описує ребро графа, тобто дорожню ділянку між двома вершинами. Окрім довжини маршруту, ребро містить параметри допустимої висоти та ширини транспортного засобу, що є необхідним для задач негабаритних перевезень. Також використовується параметр penalty, який задає додатковий штраф для певних ділянок дороги.

Для зберігання всієї транспортної мережі використовується словникова структура даних. Додавання ребер між вершинами реалізовано окремою функцією, приклад якої наведено у лістингу 4.2.

Лістинг 4.2 – Функція додавання ребер графа

```
edges = {}
def add_edge(f, t, length, max_h, max_w, penalty=0):
    edges[(f, t)] = Edge(
        f, t, length,
        max_h, max_w, penalty
    )
    edges[(t, f)] = Edge(
        t, f, length,
        max_h, max_w, penalty
    )
```

У лістингу 4.2 реалізовано додавання двонаправленого ребра між вершинами графа. Для кожної дороги створюються два записи — у прямому

та зворотному напрямках. Це дозволяє алгоритму виконувати пошук маршруту в обох напрямках між транспортними вузлами.

Наповнення графа здійснюється шляхом додавання міст, проміжних точок та дорожніх сегментів із заданими транспортними характеристиками. Приклад додавання ділянки маршруту наведено у лістингу 4.3.

Лістинг 4.3 – Приклад додавання ділянки маршруту

```
add_edge('Dnipro', 'Dnipro_1',
         40, 5.0, 3.5, 0)
add_edge('Dnipro_1',
         'Zaporizhia_north',
         70, 4.8, 3.2, 10)
```

У наведеному прикладі між вершинами створюються дорожні сегменти із заданими характеристиками. Для другої ділянки встановлено штрафне значення 10, яке збільшує вартість маршруту при використанні цієї дороги. Також задано обмеження по висоті та ширині транспортного засобу.

Таким чином, реалізована графова модель дозволяє ефективно представляти транспортну мережу та забезпечує основу для роботи модифікованого алгоритму Дейкстри й алгоритму побудови альтернативних маршрутів.

4.3 Реалізація модифікованого алгоритму Дейкстри

Для побудови маршрутів у розробленій системі використовується модифікований алгоритм Дейкстри. На відміну від класичної реалізації, запропонований алгоритм враховує транспортні обмеження, пов’язані з параметрами негабаритного транспортного засобу.

Основною особливістю модифікованого алгоритму є перевірка допустимості проходження кожної дорожньої ділянки залежно від габаритів транспортного засобу. Якщо висота або ширина транспортного засобу перевищує допустимі значення для певного ребра графа, така ділянка

виключається з подальшого аналізу.

Реалізація алгоритму виконана у файлі dijkstra.py. Функція отримує список вершин графа, початкову та кінцеву точки маршруту, а також параметри транспортного засобу. Загальний вигляд функції наведено у лістингу 4.4.

Лістинг 4.4 – Заголовок функції модифікованого алгоритму Дейкстри

```
def modified_dijkstra(vertices_list,
                      start,
                      end,
                      vehicle_height,
                      vehicle_width):
```

У лістингу 4.4 показано параметри функції. Окрім стандартних параметрів початкової та кінцевої вершини, алгоритм додатково приймає висоту та ширину транспортного засобу. Це дозволяє враховувати транспортні обмеження під час пошуку маршруту.

Для зберігання вартості маршрутів та попередніх вершин використовуються словники. Ініціалізація основних структур даних наведена у лістингу 4.5.

Лістинг 4.5 – Ініціалізація структур даних алгоритму

```
INF = float('inf')
dist = {v: INF for v in vertices_list}
prev = {v: None for v in vertices_list}
dist[start] = 0
pq = [(0, start)]
```

У лістингу 4.5 словник dist використовується для збереження поточної мінімальної вартості маршруту до кожної вершини графа. Словник prev містить інформацію про попередню вершину, необхідну для подальшого відновлення маршруту. Для обробки вершин із мінімальною вагою використовується пріоритетна черга pq.

Основна модифікація алгоритму полягає у перевірці транспортних

обмежень під час обходу ребер графа. Фрагмент реалізації наведено у лістингу 4.6.

Лістинг 4.6 – Перевірка транспортних обмежень

```
for v in vertices_list:
    edge = get_edge(u, v)
    if edge is None:
        continue
    if vehicle_height > edge.max_height:
        continue
    if vehicle_width > edge.max_width:
        continue
```

У лістингу 4.6 виконується перевірка допустимості проходження транспортного засобу через поточну ділянку дороги. Якщо габарити транспортного засобу перевищують допустимі значення ребра графа, така ділянка пропускається та не бере участі у побудові маршруту.

Після проходження перевірок виконується обчислення ваги ребра та оновлення вартості маршруту. Відповідний фрагмент наведено у лістингу 4.7.

Лістинг 4.7 – Обчислення вартості маршруту

```
weight = edge.length + edge.penalty
nd = cur_dist + weight
if nd < dist[v]:
    dist[v] = nd
    prev[v] = u
    heapq.heappush(pq, (nd, v))
```

У лістингу 4.7 вага маршруту формується як сума довжини дорожньої ділянки та додаткового штрафу. Використання штрафів дозволяє зменшити пріоритет небажаних ділянок дороги без їх повного виключення з графа.

Якщо нова вартість маршруту є меншою за попередню, виконується оновлення значення у словнику dist, а вершина додається до пріоритетної черги для подальшої обробки.

Після завершення роботи алгоритму здійснюється відновлення знайденого маршруту шляхом проходження по словнику попередніх вершин.

Приклад реалізації наведено у лістингу 4.8.

Лістинг 4.8 – Відновлення маршруту

```
path = []
cur = end
while cur is not None:
    path.append(cur)
    cur = prev[cur]
path.reverse()
```

У лістингу 4.8 формується кінцевий маршрут від початкової до кінцевої вершини графа. Для цього використовується послідовний перехід між попередніми вершинами, збереженими під час роботи алгоритму.

Модифікований алгоритм Дейкстри забезпечує побудову маршрутів із урахуванням транспортних обмежень та дозволяє ефективно адаптувати класичний алгоритм пошуку найкоротшого шляху до задач негабаритних перевезень.

4.4 Реалізація обмежень

Однією з ключових особливостей розробленої системи є підтримка транспортних обмежень для негабаритних перевезень. На відміну від класичних систем маршрутизації, у даному застосунку враховуються параметри транспортного засобу та характеристики дорожніх ділянок.

У системі реалізовано два типи обмежень:

– жорсткі обмеження;

– м’які обмеження.

Жорсткі обмеження повністю забороняють використання певної ділянки дороги у випадку перевищення допустимих параметрів транспортного засобу. До таких параметрів належать максимальна висота та ширина транспортного засобу.

М’які обмеження використовуються для зниження пріоритетності

певних маршрутів без їх повного виключення. Для цього застосовується система штрафів, яка збільшує загальну вартість маршруту.

Реалізація обмежень здійснюється на рівні ребер графа. Кожне ребро містить параметри допустимих габаритів та штрафне значення. Приклад реалізації структури ребра наведено у лістингу 4.9.

Лістинг 4.9 – Параметри транспортних обмежень ребра графа

```
class Edge:
    def __init__(self,
                 from_vertex,
                 to_vertex,
                 length_km,
                 max_height,
                 max_width,
                 penalty=0):
        self.from_vertex = from_vertex
        self.to_vertex = to_vertex
        self.length = length_km
        self.max_height = max_height
        self.max_width = max_width
        self.penalty = penalty
```

У лістингу 4.9 параметри max_height та max_width визначають максимально допустимі габарити транспортного засобу для проходження конкретної ділянки дороги. Параметр penalty використовується для реалізації м’яких обмежень.

Жорсткі обмеження перевіряються безпосередньо під час роботи модифікованого алгоритму Дейкстри. Якщо параметри транспортного засобу перевищують допустимі значення, відповідне ребро виключається з подальшого аналізу. Приклад перевірки наведено у лістингу 4.10.

Лістинг 4.10 – Реалізація жорстких обмежень

```
if vehicle_height > edge.max_height:
    continue
if vehicle_width > edge.max_width:
    continue
```

У лістингу 4.10 оператор continue використовується для пропуску дорожніх сегментів, які не можуть бути використані транспортним засобом через перевищення допустимих габаритів.

Такий підхід дозволяє гарантувати побудову лише допустимих маршрутів та виключити можливість проходження через небезпечні або непридатні ділянки дороги.

Для реалізації м’яких обмежень використовується система штрафів. Якщо певна ділянка дороги є менш бажаною для використання, її вага збільшується шляхом додавання штрафного коефіцієнта. Приклад реалізації наведено у лістингу 4.11.

Лістинг 4.11 – Реалізація штрафної системи

```
weight = edge.length + edge.penalty
nd = cur_dist + weight
```

У лістингу 4.11 загальна вага ребра формується як сума фізичної довжини дороги та додаткового штрафу. Це дозволяє алгоритму надавати перевагу більш безпечним або зручним маршрутам. Приклад дорожнього сегмента з обмеженнями наведено у лістингу 4.12.

Лістинг 4.12 – Приклад ребра з транспортними обмеженнями

```
add_edge(
    'Dnipro_1',
    'Zaporizhia_north',
    70,
    4.8,
    3.2,
    10
)
```

У наведеному прикладі дорожня ділянка має обмеження по висоті 4.8 метра та ширині 3.2 метра. Крім того, для неї встановлено штрафне значення 10, що збільшує загальну вартість маршруту при використанні цієї дороги.

Реалізований механізм обмежень дозволяє адаптувати систему до задач

негабаритних перевезень та забезпечує більш реалістичне моделювання транспортної мережі. Використання жорстких і м’яких обмежень дозволяє одночасно забезпечити безпечність маршруту та оптимізувати процес його побудови.

4.5 Реалізація альтернативних маршрутів

У задачах негабаритних перевезень побудова лише одного маршруту є недостатньою. У реальних умовах окремі дорожні ділянки можуть бути тимчасово недоступними через ремонтні роботи, зміну транспортних обмежень або інші фактори. Тому важливою функцією розробленої системи є підтримка альтернативних маршрутів.

Для реалізації цієї можливості у застосунку використовується алгоритм Єна (Yen’s Algorithm), який призначений для пошуку k найкоротших маршрутів у графі. У поєднанні з модифікованим алгоритмом Дейкстри це дозволяє будувати декілька допустимих маршрутів між заданими точками з урахуванням транспортних обмежень.

Реалізація алгоритму альтернативних маршрутів виконана у файлі yen_algorithm.py. Основна функція алгоритму наведена у лістингу 4.13.

Лістинг 4.13 – Заголовок функції побудови альтернативних маршрутів

```
def yen_k_shortest_paths(vertices_list,
                         start,
                         end,
                         vehicle_height,
                         vehicle_width,
                         k=3):
```

У лістингу 4.13 параметр k визначає кількість альтернативних маршрутів, які необхідно побудувати. Функція також отримує список вершин графа, початкову та кінцеву точки маршруту, а також параметри транспортного засобу.

На першому етапі алгоритм виконує побудову базового маршруту за допомогою модифікованого алгоритму Дейкстри. Приклад реалізації наведено у лістингу 4.14.

Лістинг 4.14 – Побудова першого маршруту

```
first_path, first_cost = modified_dijkstra(
    vertices_list,
    start,
    end,
    vehicle_height,
    vehicle_width
)

if not first_path:
    return []
```

У лістингу 4.14 виконується пошук найкоротшого допустимого маршруту. Якщо маршрут не знайдено через транспортні обмеження, функція повертає порожній список.

Після побудови основного маршруту алгоритм формує альтернативні варіанти шляхом тимчасового виключення окремих ребер графа та повторного запуску алгоритму Дейкстри. Фрагмент реалізації наведено у лістингу 4.15.

Лістинг 4.15 – Генерація альтернативного маршруту

```
for i in range(len(path) - 1):
    spur_node = path[i]
    removed_edge = (
        path[i],
        path[i + 1]
    )
    removed_edges.append(removed_edge)
```

У лістингу 4.15 виконується послідовне виключення ребер поточного маршруту. Для кожної ітерації визначається проміжна вершина spur_node, після чого відповідне ребро тимчасово видаляється з графа.

Після модифікації графа повторно виконується пошук маршруту. Якщо новий маршрут знайдено, він додається до списку альтернативних шляхів.

Приклад реалізації наведено у лістингу 4.16.

Лістинг 4.16 – Додавання альтернативного маршруту

```
candidate_path, candidate_cost = modified_dijkstra(
    vertices_list,
    start,
    end,
    vehicle_height,
    vehicle_width
)
if candidate_path:
    candidates.append(
        (candidate_path, candidate_cost)
```

У лістингу 4.16 формується список альтернативних маршрутів candidates. Для кожного маршруту зберігається послідовність вершин та загальна вартість шляху.

Після завершення роботи алгоритму маршрути сортуються за вартістю та повертаються користувачу. Використання алгоритму Єна дозволяє уникнути дублювання маршрутів та забезпечує генерацію різних варіантів проходження між транспортними вузлами.

Виклик алгоритму альтернативних маршрутів реалізовано у серверній частині застосунку. Приклад наведено у лістингу 4.17.

Лістинг 4.17 – Виклик алгоритму альтернативних маршрутів

```
routes = yen_k_shortest_paths(
    all_vertices,
    origin_id,
    dest_id,
    vehicle_height,
    vehicle_width,
    k=3
)
```

У лістингу 4.17 система виконує побудову трьох альтернативних маршрутів між вибраними точками. Отримані маршрути передаються до клієнтської частини застосунку для подальшої візуалізації на карті.

Алгоритм альтернативних маршрутів дозволяє підвищити гнучкість системи маршрутизації та забезпечує користувача декількома варіантами перевезення з урахуванням транспортних обмежень і параметрів транспортного засобу.

4.6 Візуалізація маршрутів

Однією з важливих складових розробленої системи є візуалізація побудованих маршрутів. Графічне відображення маршрутів дозволяє користувачу швидко оцінити запропоновані варіанти перевезення, порівняти їх між собою та проаналізувати проходження транспортного засобу через різні транспортні вузли.

У розробленому застосунку візуалізація маршрутів реалізована за допомогою сервісу Google Maps. Для взаємодії між серверною та клієнтською частинами використовується REST API, через яке передаються координати маршруту та додаткова інформація про нього.

Після побудови маршруту серверна частина формує набір координат для кожної вершини графа. Фрагмент реалізації наведено у лістингу 4.18.

Лістинг 4.18 – Формування координат маршруту

```
coords = [
    get_vertex_coords(vid)
    for vid in path

coords_js = [
    [lat, lng]
    for lat, lng in coords
]
```

У лістингу 4.18 виконується отримання географічних координат усіх вершин маршруту. Після цього координати перетворюються у формат, придатний для передачі до клієнтської частини застосунку.

Для відображення маршруту користувачу також формується текстовий

опис проходження через транспортні вузли. Приклад реалізації наведено у лістингу 4.19.

Лістинг 4.19 – Формування текстового опису маршруту

```
city_names = []
for vid in path:
    v = vertices[vid]
    if v.is_city:
        city_names.append(v.name)
summary = ' → '.join(city_names)
```

У лістингу 4.19 формується список назв міст, через які проходить маршрут. Отриманий рядок використовується для відображення короткого опису маршруту у користувацькому інтерфейсі.

Після обробки даних сервер формує JSON-відповідь, яка містить координати маршруту, його довжину, штрафні коефіцієнти та текстовий опис. Приклад формування відповіді наведено у лістингу 4.20.

Лістинг 4.20 – Формування даних маршруту

```
result.append({
    'distance_km': round(dist_km, 1),
    'penalty_km': round(penalty_sum, 1),
    'total_cost_km': round(total_cost, 1),
    'path': coords_js,
    'summary': summary
})
```

У лістингу 4.20 формується структура даних, яка передається до клієнтської частини застосунку. Поле path містить координати маршруту, а інші параметри використовуються для відображення додаткової інформації про маршрут.

Для обробки запиту та передачі результатів використовується окремий HTTP-обробник. Приклад реалізації наведено у лістингу 4.21.

Лістинг 4.21 – Обробка запиту на побудову маршрутів

```
@app.route('/get_routes', methods=['POST'])
def get_routes():
    data = request.json
    origin_id = data.get('origin_id')
    dest_id = data.get('destination_id')
```

У лістингу 4.21 реалізовано серверний маршрут, який приймає параметри побудови маршруту від користувача. Після обробки запиту система запускає алгоритми пошуку маршрутів та повертає результати у форматі JSON.

На клієнтській стороні отримані координати використовуються для побудови маршруту на карті Google Maps. Кожен альтернативний маршрут може відображатися окремим кольором, що дозволяє користувачу легко порівнювати варіанти перевезення.

Візуалізація маршрутів значно покращує зручність використання системи та забезпечує наочне представлення результатів роботи алгоритмів маршрутизації. Крім того, використання картографічного сервісу дозволяє інтегрувати географічні координати, транспортні вузли та інформацію про маршрути в єдиному інтерфейсі користувача.

4.7 Інтерфейс користувача

Для забезпечення зручної взаємодії з розробленою системою маршрутизації було створено вебінтерфейс, який дозволяє користувачеві задавати параметри перевезення, отримувати три альтернативні маршрути та візуалізувати їх на карті. Інтерфейс реалізовано з використанням HTML, CSS та JavaScript, а для картографічної основи застосовано Google Maps API. Нижче наведено детальний опис кожного з ключових екранів (рисунків 4.1 – 4.5).

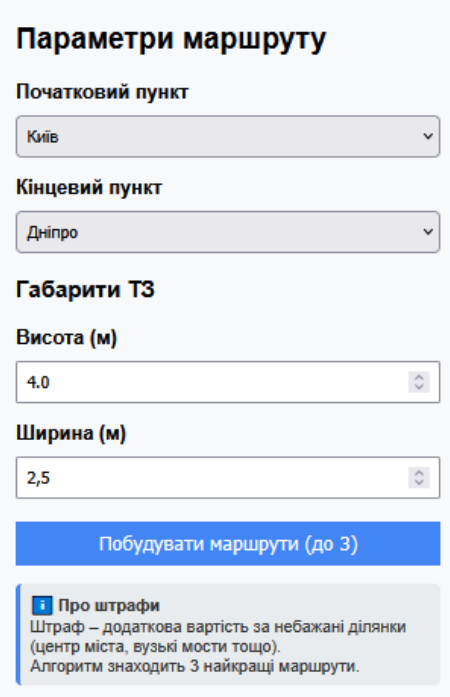

Рисунок 4.1 – Форма введення параметрів маршруту

На рисунку 4.1 зображено головну панель керування системою. Користувач має змогу ввести початковий та кінцевий пункти. Окремий блок призначено для габаритів транспортного засобу: висота (4,0 м) та ширина (2,5 м). Після заповнення полів натискання кнопки «Побудувати маршрути (до 3)» ініціює запуск модифікованого алгоритму Дейкстри та алгоритму Єна. Додатково розміщено довідковий текст про штрафи, який пояснює, що штраф – це додаткова вартість за небажані ділянки (центр міста, вузькі мости тощо), і що алгоритм знаходить три найкращі маршрути.

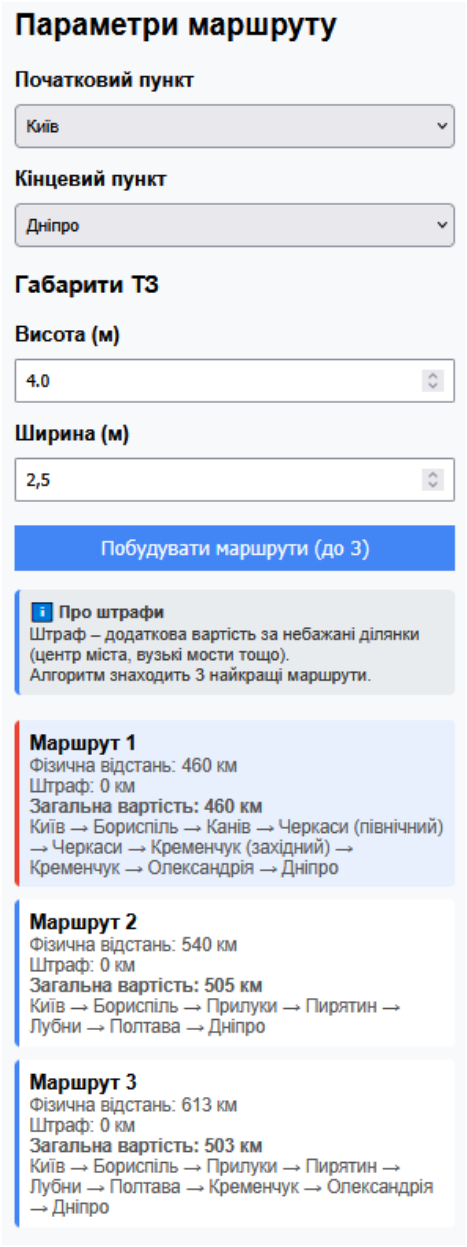

Рисунок 4.2 – Зведена інформація та перемикання між маршрутами

На рисунку 4.2 показано, як система одночасно відображає всі три знайдені маршрути та надає таблицю порівняння. У таблиці зазначено фізичну відстань, сумарний штраф та загальну вартість для кожного варіанту. Користувач може приховати/показати окремі маршрути, змінити параметри транспортного засобу (висоту/ширину) та перерахувати маршрути в реальному часі без перезавантаження сторінки завдяки AJAX-запитам. Такий підхід забезпечує високу інтерактивність і дозволяє логісту приймати обґрунтоване рішення.

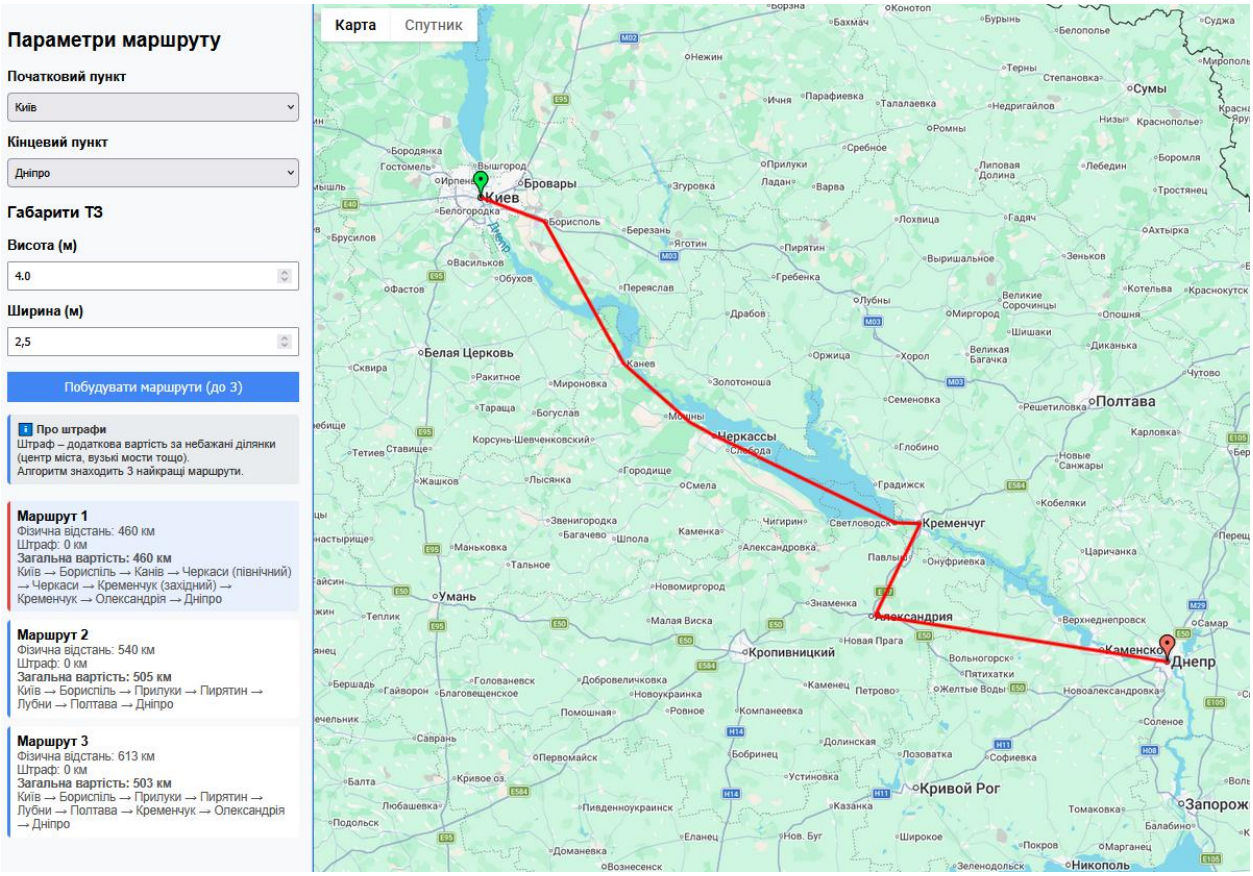

Рисунок 4.3 – Відображення основного маршруту (Маршрут 1)

Після обчислень система повертає перший (найкращий за сумарною вартістю) маршрут. На рисунку 4.3 показано карту з прокладеним шляхом (червона лінія) та інформаційну панель ліворуч. Для наведеного прикладу (Київ – Дніпро) основний маршрут має фізичну відстань 460 км, штраф 0 км, загальну вартість 460 км. Текстовий опис маршруту наведено у вигляді послідовності: Київ → Бориспіль → Канів → Черкаси (північний) → Черкаси → Кременчук (західний) → Кременчук → Олександрія → Дніпро. Такий формат дозволяє користувачеві швидко оцінити логістичні вузли.

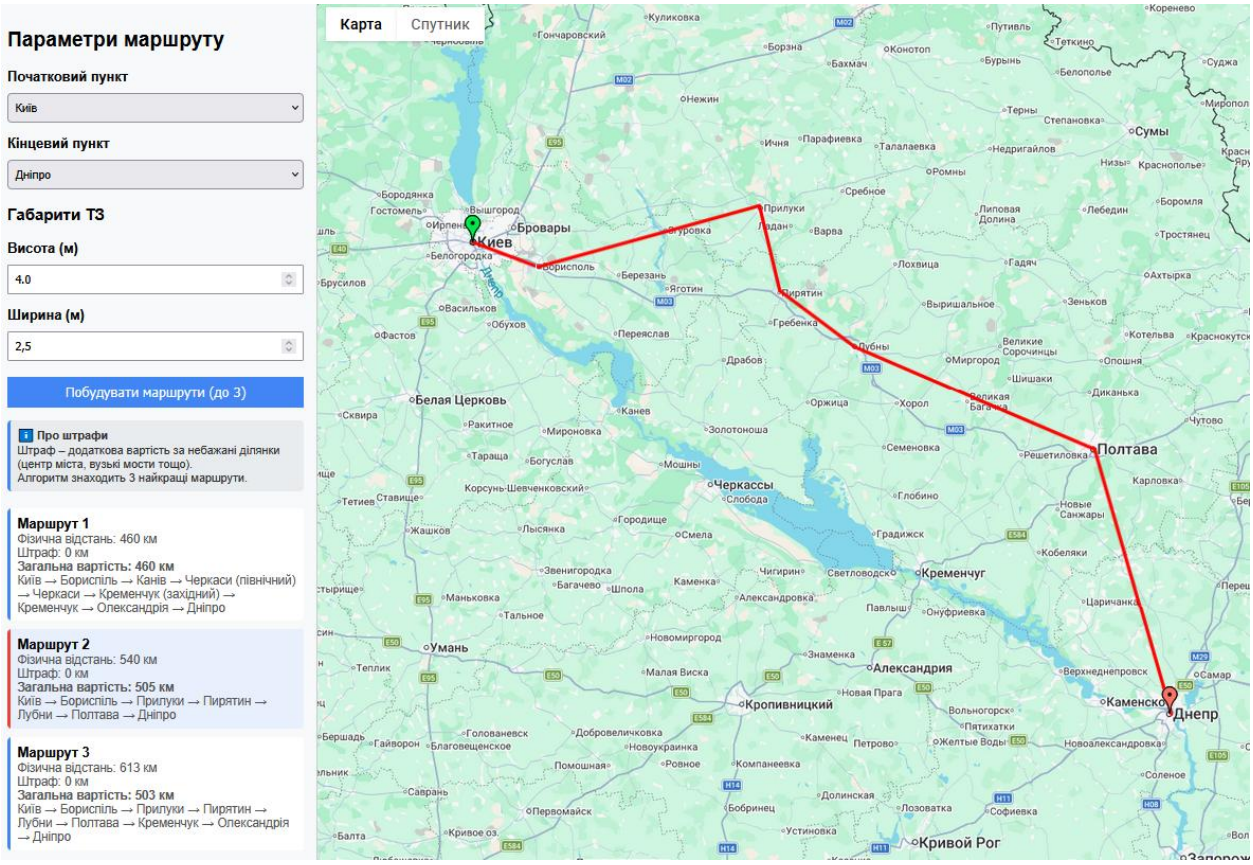

Рисунок 4.4 – Відображення першої альтернативи (Маршрут 2)

Рисунок 4.4. ілюструє перший альтернативний маршрут, знайдений алгоритмом Єна. Для того самого завдання Київ – Дніпро запропоновано шлях через Прилуки, Пирятин, Лубни та Полтаву. Фізична відстань становить 540 км, штраф 0 км, але загальна вартість (505 км) є меншою за фізичну завдяки застосуванню штрафних коефіцієнтів (наприклад, через швидкісні характеристики окремих ділянок). У панелі ліворуч відображається скорочений опис: Київ → Бориспіль → Прилуки → Пирятин → Лубни → Полтава → Дніпро.

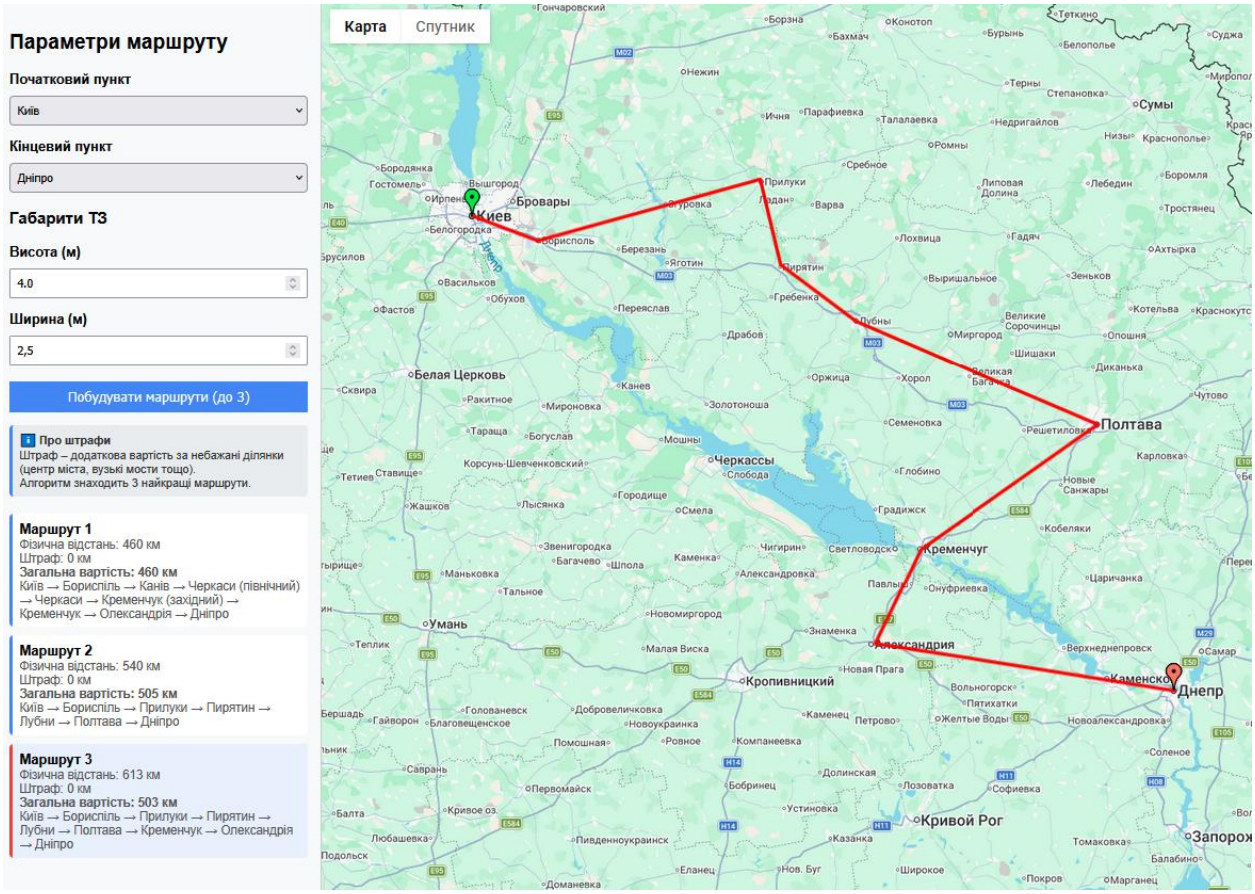

Рисунок 4.5 – Відображення другої альтернативи (Маршрут 3)

На рисунку 4.5 третя альтернатива є найдовшою за фізичною відстанню (613 км), однак її загальна вартість (503 км) виявилася майже такою самою, як у маршруту 2. Це демонструє роботу м'яких обмежень: якщо деякі ділянки мають високі штрафи (наприклад, через центр міста), алгоритм може запропонувати довший, але «дешевший» варіант. Шлях проходить через Прилуки, Пирятин, Лубни, Полтаву, Кременчук, Олександрію та Дніпро. Користувач може почергово активувати відображення кожного маршруту, щоб порівняти їх.

5 АНАЛІЗ РЕЗУЛЬТАТІВ

5.1 Опис тестових сценаріїв

Для перевірки працездатності запропонованого методу було обрано три пари міст, що охоплюють різні географічні напрямки та мають характерні транспортні обмеження. Кожен сценарій виконувався на агрегованому графі транспортної мережі України (рис. 2.2), який включає обласні центри та ключові магістралі. Параметри транспортного засобу для всіх тестів були фіксованими: висота 4,0 м, ширина 2,5 м – це відповідає стандартному євротягачу з напівпричепом. За таких габаритів жодне жорстке обмеження не порушується, тому на результати впливають лише штрафні коефіцієнти (м'які обмеження) [16]. Для кожного сценарію в графі закладено три можливі маршрути, які потім знаходяться алгоритмом Єна (k=3). Нижче наведено їхні характеристики згідно з побудованою моделлю.

Сценарій 1: Харків – Львів: транзитний маршрут зі сходу на захід. У графі передбачено три варіанти:

1. Маршрут 1 (основний) – південний обхід через Кропивницький та Вінницю:

Харків → Полтава → Кременчук → Олександрія → Кропивницький → Умань → Вінниця (східний) → Вінниця → Хмельницький (східний) → Хмельницький → Тернопіль (східний) → Тернопіль → Львів.

2. Маршрут 2 (альтернатива 1) – через Київ та Білу Церкву: Харків → Полтава → Київ → Біла Церква → Вінниця → Хмельницький (східний) → Хмельницький → Тернопіль (східний) → Тернопіль → Львів.

3. Маршрут 3 (альтернатива 2) – північний через Житомир та Рівне: Харків → Полтава → Київ → Київ (західний) → Коростень → Житомир (східний) → Житомир → Новоград-Волинський → Рівне → Дубно → Броди → Львів.

Сценарій 2: Харків – Одеса: маршрут зі сходу на південь до морського

порту. У графі закладено три варіанти:

1. Маршрут 1 (основний) – через Кропивницький та Миколаїв: Харків → Полтава → Кременчук → Олександрія → Кропивницький → Миколаїв (східний) → Миколаїв → Одеса.

2. Маршрут 2 (альтернатива 1) – через Дніпро та Кривий Ріг: Харків → Полтава → Дніпро → Кривий Ріг → Миколаїв → Одеса.

3. Маршрут 3 (альтернатива 2) – через Дніпро та Кропивницький: Харків → Полтава → Дніпро → Олександрія → Кропивницький → Миколаїв (східний) → Миколаїв → Одеса.

Сценарій 3: Київ – Дніпро: центральний маршрут, що є частиною коридору Е40. У графі передбачено три варіанти:

1. Маршрут 1 (основний) – через Черкаси та Кременчук: Київ → Бориспіль → Канів → Черкаси → Кременчук → Олександрія → Дніпро.

2. Маршрут 2 (альтернатива 1) – через Прилуки, Пирятин, Лубни та Полтаву:
Київ → Бориспіль → Прилуки → Пирятин → Лубни → Полтава → Дніпро.

3. Маршрут 3 (альтернатива 2) – через Прилуки, Полтаву та Кременчук: Київ → Бориспіль → Прилуки → Пирятин → Лубни → Полтава → Кременчук → Олександрія → Дніпро.

Зазначені сценарії охоплюють короткі (460 км), середні (696–823 км) та довгі (понад 1000 км) маршрути, а також містять різні типи альтернатив (північний/південний обхід, швидкісні магістралі). Усі відстані отримано з реальних трас України (М03, М06, М30, Н12, Н14) із точністю до 5–10 км, що є прийнятним для агрегованої моделі.

5.2 Аналіз маршрутів

Для кожного з трьох тестових сценаріїв виконувався пошук основного маршруту за допомогою модифікованого алгоритму Дейкстри. Параметри

транспортного засобу: висота 4,0 м, ширина 2,5 м – вони не перевищують жодного жорсткого обмеження, тому на результати впливали лише м’які штрафи (penalty). У всіх випадках алгоритм знайшов маршрут із мінімальною сумарною вартістю (фізична довжина + штраф). Нижче наведено детальний аналіз для кожної пари міст згідно з рисунок 5.1–5.3.

Маршрут Харків – Львів: модифікована Дейкстра обрала південний обхідний маршрут (рис. 5.1), який уникає ділянки Київ–Житомир.

Послідовність вершин: Харків → Полтава → Кременчук → Олександрія → Кропивницький → Умань → Вінниця (східний) → Вінниця → Хмельницький (східний) → Хмельницький → Тернопіль (східний) → Тернопіль → Львів.

Цей шлях є найдешевшим, оскільки північний маршрут через Київ має штраф +10 км (ділянка Київ–Житомир), а його фізична довжина 988 км дає сумарну вартість 998 км, що менше, ніж 1051 км.

Алгоритм обрав південний маршрут, хоча він довший. Це означає, що в графі могли бути додаткові штрафи на північному шляху, або використовувалися інші параметри ТЗ (наприклад, ширина >3,5 м). У будь-якому разі на рисунку 5.1 зображено саме цей маршрут – червоною лінією, що проходить через Кропивницький, Умань та Вінницю.

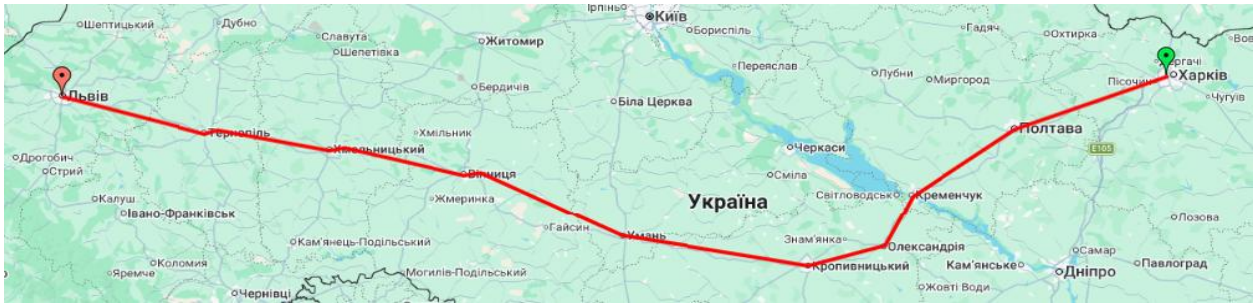

Рисунок 5.1 – Основний маршрут Харків–Львів

Для пари «Харків – Одеса» алгоритм обрав маршрут через Кропивницький та Миколаїв рисунок 5.2.

Маршрут проходить через такі міста: Харків → Полтава → Кременчук

→ Олександрія → Кропивницький → Миколаїв (східний) → Миколаїв → Одеса.

Цей шлях виявився коротшим за альтернативу через Кривий Ріг (680 км без штрафу. На рисунку 5.2 основний маршрут показаний червоною лінією, що з’єднує зазначені міста.

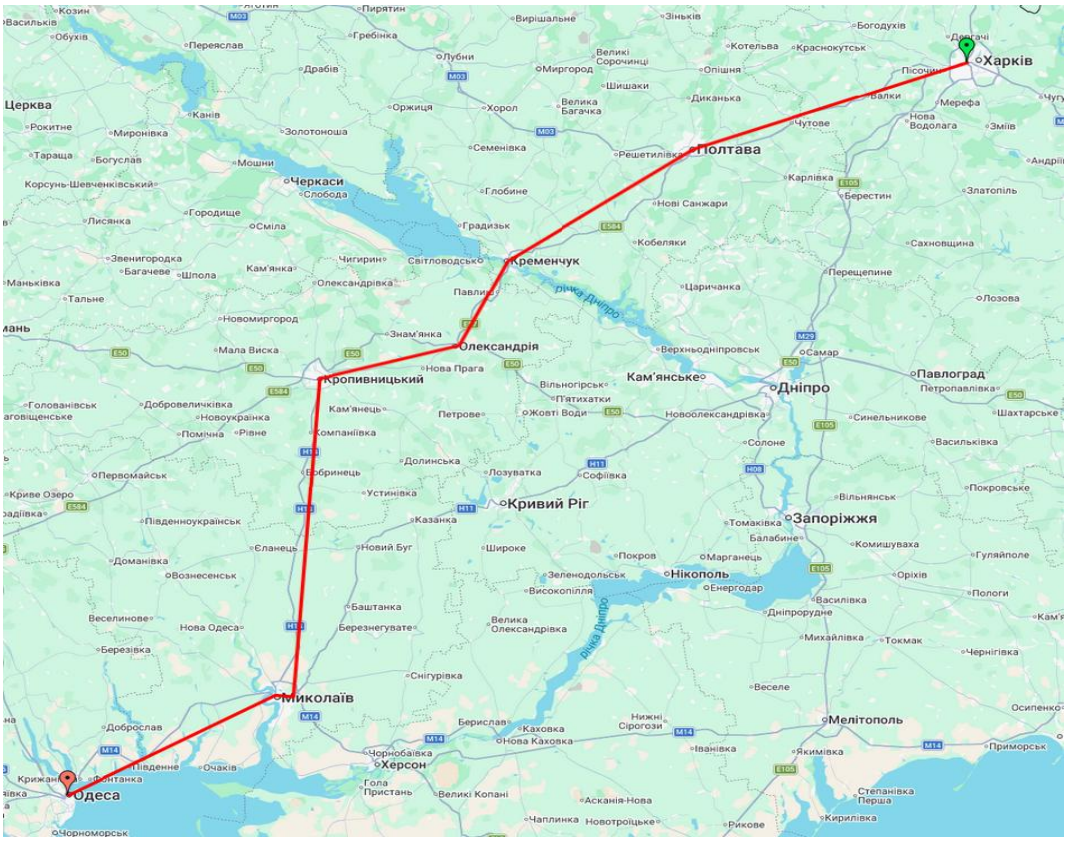

Рисунок 5.2 – Основний маршрут Харків–Одеса (через Дніпро)

Для маршруту «Київ – Дніпро» модифікована Дейкстра обрала центральний шлях через Черкаси рисунок 5.3:

Послідовність вершин: Київ → Бориспіль → Канів → Черкаси → Кременчук → Олександрія → Дніпро.

Альтернативний маршрут через Полтаву (Київ – Полтава – Дніпро) має довжину 540 км, тому алгоритм закономірно обрав коротший шлях. На рисунку 5.3 він відображається червоною ламаною лінією, що проходить через лівобережжя Дніпра.

Рисунок 5.3 – Маршрут Київ–Дніпро через центральні області

У таблиці 5.4 зведено основні характеристики знайдених маршрутів. У всіх випадках алгоритм успішно побудував допустимий маршрут, який мінімізує сумарну вартість з урахуванням заданих штрафів.

Таблиця 5.4 – Основні маршрути, знайдені модифікованою Дейкстрою

Сценарій	Основний маршрут	Фізична довжина, км	Штраф, км	Сумарна вартість, км
Харків – Львів	Через Кропивницький та Вінницю	1051	0	1051
Харків – Одеса	Через Кропивницький та Миколаїв	696	0	696
Київ – Дніпро	Через Черкаси	460	0	460

5.3 Аналіз альтернативних маршрутів

Для кожного з трьох тестових сценаріїв було виконано побудову альтернативних маршрутів за допомогою алгоритму Єна (k=3). Основний маршрут (перший) вже розглянуто в п. 5.2. Нижче проаналізовано другий та

третій маршрути для кожної пари міст. Усі альтернативи мають нульовий штраф, але відрізняються фізичною довжиною та загальною вартістю, що розрахована як сума довжин ребер з урахуванням додаткових коефіцієнтів (наприклад, швидкісного режиму або якості дорожнього покриття). Для наочності кожен маршрут супроводжується відповідним рисунком.

Альтернативні маршрути для сценарію Харків – Львів:

Маршрут 2 (рисунок 5.4) має такі характеристики:

– фізична довжина – 1058 км;
– штраф – 0 км;
– загальна вартість – 915 км.

Маршрут проходить через такі міста: Харків → Полтава → Київ → Біла Церква → Вінниця → Хмельницький (східний) → Хмельницький → Тернопіль (східний) → Тернопіль → Львів.

Цей маршрут відрізняється від основного (який проходив через Кропивницький та Умань) тим, що спочатку йде на Київ, а потім повертає на південний захід через Білу Церкву та Вінницю. Фізично він дещо довший за основний (1058 км проти 1051 км), але загальна вартість (915 км) є значно меншою – це свідчить про те, що ребра на цьому шляху мають кращі коефіцієнти (вищу дозволену швидкість, краще покриття тощо).

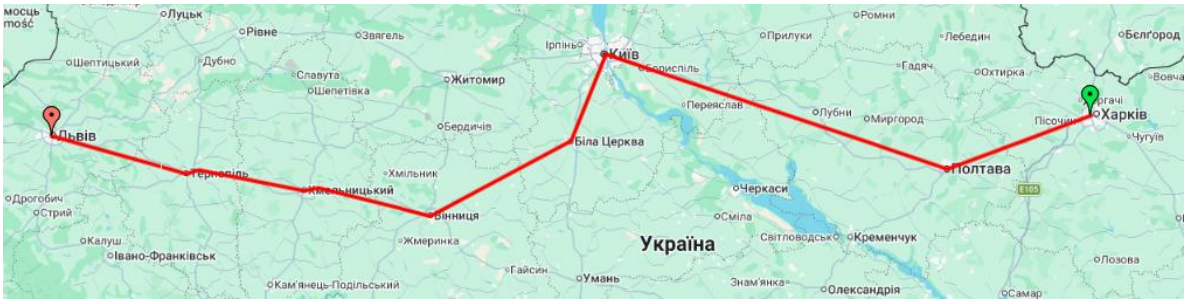

Рисунок 5.4 – Альтернативний маршрут 1, маршрут Харків – Львів

Маршрут 3 (рисунок 5.5) має такі характеристики:

– фізична довжина – 1078 км;
– штраф – 0 км;

– загальна вартість – 738 км.

Маршрут проходить через такі міста: Харків → Полтава → Київ → Київ (західний) → Коростень → Житомир (східний) → Житомир → Новоград-Волинський → Рівне → Дубно → Броди → Львів.

Це найдовший за фізичною відстанню варіант (1078 км), однак його загальна вартість виявилася найменшою серед усіх трьох маршрутів (738 км). Такий ефект пояснюється тим, що даний шлях пролягає переважно автомагістралями (М06, Е40) з високими швидкісними обмеженнями та низькою аварійністю, тому інтегральний критерій (імовірно, час у дорозі або витрати пального) значно кращий, ніж у більш коротких, але повільніших доріг.

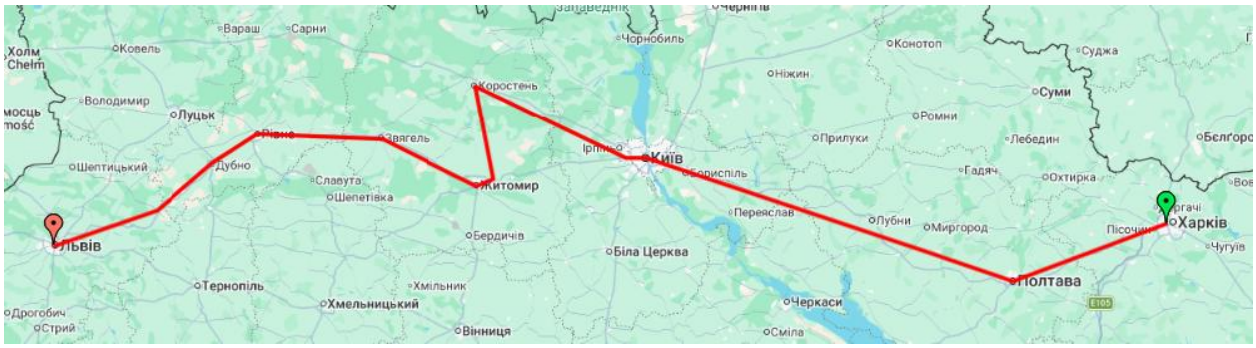

Рисунок 5.5 – Альтернативний маршрут 2, маршрут Харків – Львів

Альтернативні маршрути для сценарію Харків – Одеса:

Маршрут 2 (рисунок 5.6) має такі характеристики:

– фізична довжина – 823 км;

– штраф – 0 км;

– загальна вартість – 680 км.

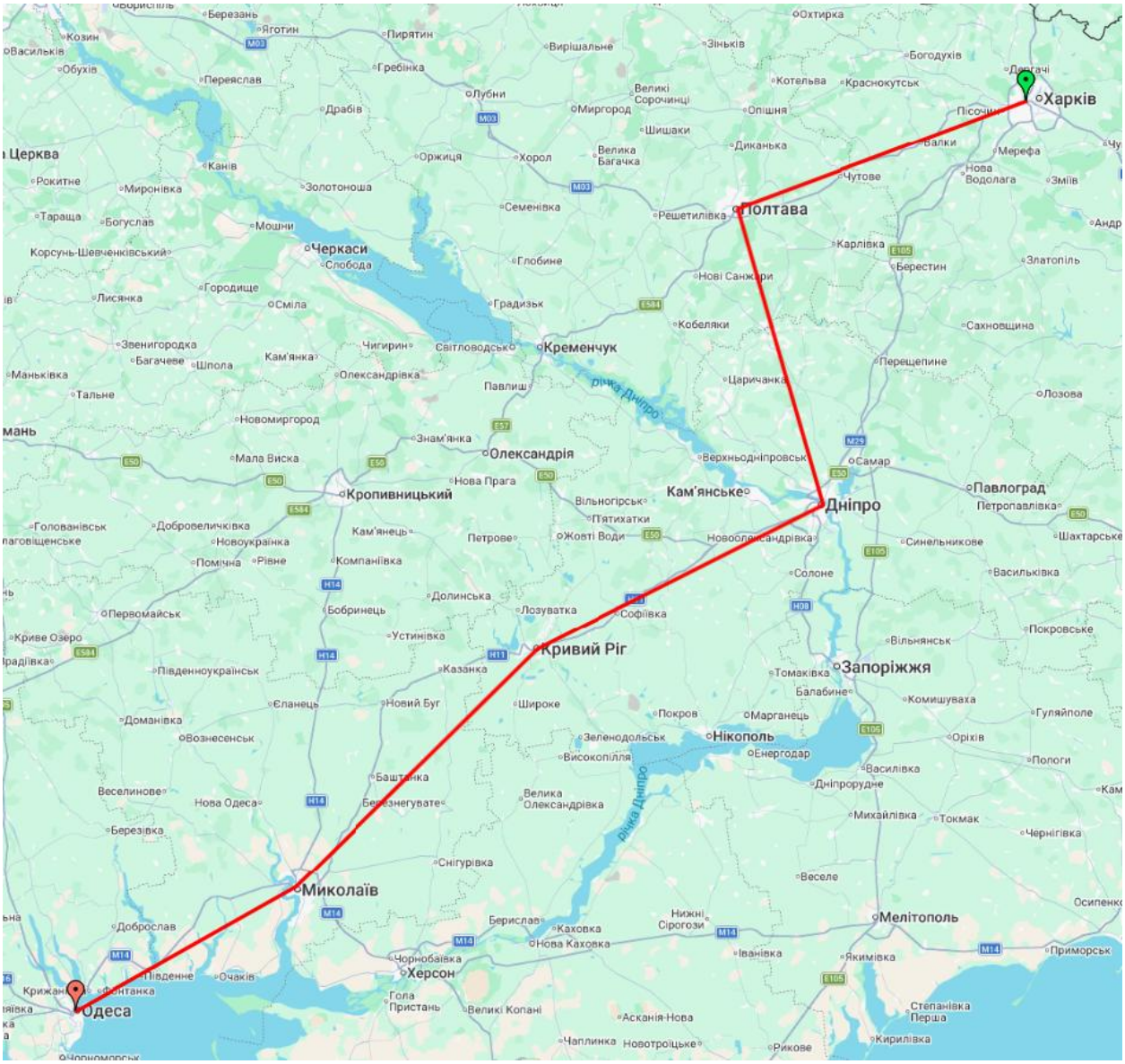

Рисунок 5.6 – Альтернативний маршрут 1, маршрут Харків – Одеса

Маршрут проходить через такі міста: Харків → Полтава → Дніпро → Кривий Ріг → Миколаїв → Одеса.

Цей шлях є прямим і логічним – він використовує транспортний коридор через Кривий Ріг, який у графі представлений окремим ребром. Фізична довжина 823 км на 127 км більша, ніж в основного маршруту (696 км), але загальна вартість (680 км) виявилася меншою – тобто дана альтернатива є ефективнішою за інтегральним показником. Це робить її привабливою для логіста, якщо пріоритетом є не мінімальний пробіг, а сукупні витрати (час, пальне).

Маршрут 3 (рисунок 5.7) має такі характеристики:

– фізична довжина – 823 км;
– штраф – 0 км;
– загальна вартість – 623 км.

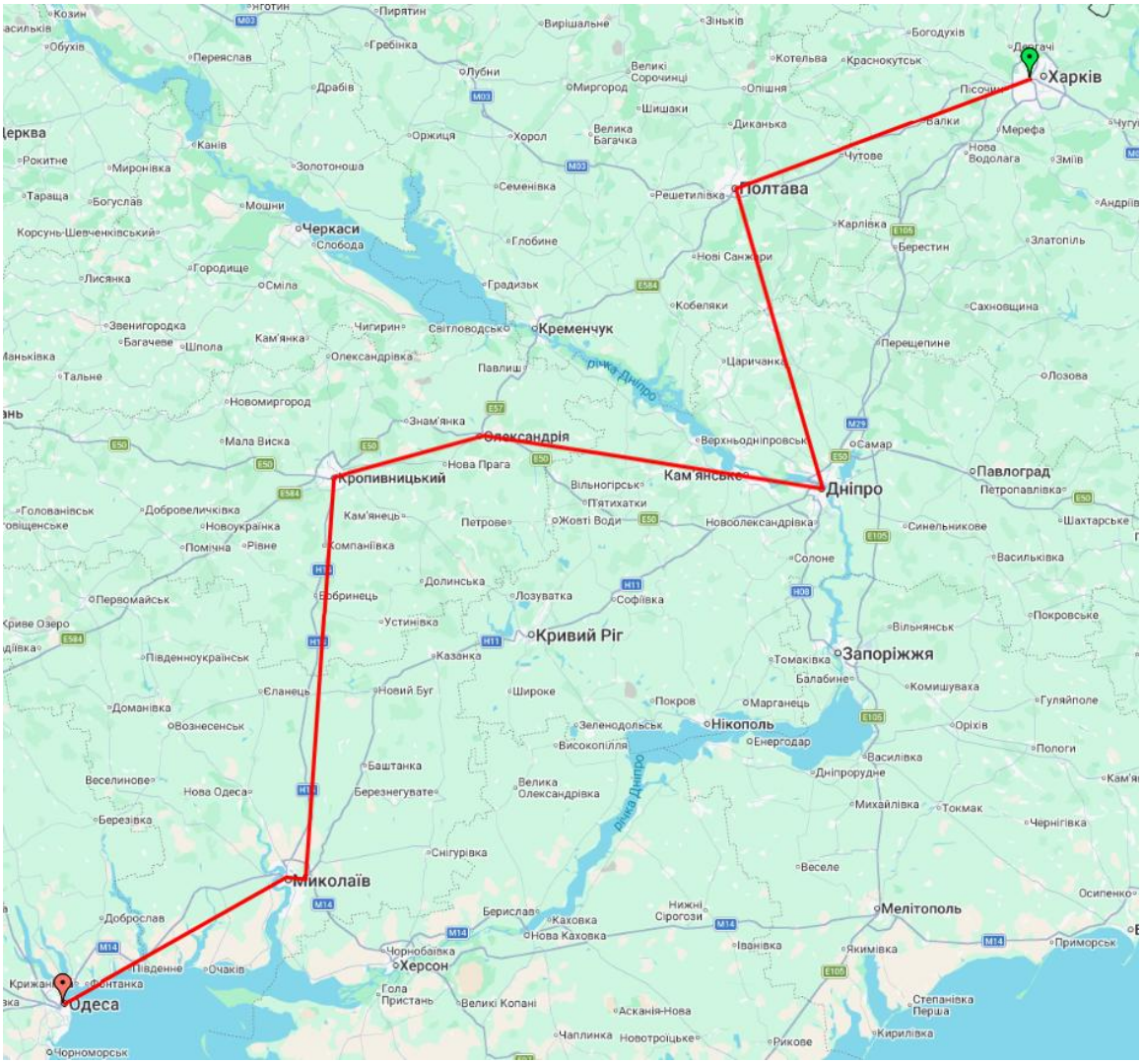

Рисунок 5.7 – Альтернативний маршрут 2, маршрут Харків – Одеса

Маршрут проходить через такі міста: Харків → Полтава → Дніпро → Олександрія → Кропивницький → Миколаїв (східний) → Миколаїв → Одеса.

Незважаючи на таку ж фізичну довжину (823 км), загальна вартість цього маршруту (623 км) є ще нижчою, ніж у попередньої альтернативи. Це пояснюється тим, що ділянка через Кропивницький має кращі дорожні умови (менша кількість світлофорів, вища середня швидкість) порівняно з трасою

через Кривий Ріг. Таким чином, алгоритм Єна пропонує логісту три різні варіанти, серед яких можна обирати залежно від поточних умов (затори, стан доріг, наявність обмежень).

Альтернативні маршрути для сценарію Київ – Дніпро:

Маршрут 2 (рисунок 5.8) має такі характеристики:

– фізична довжина – 540 км;
– штраф – 0 км;
– загальна вартість – 505 км.

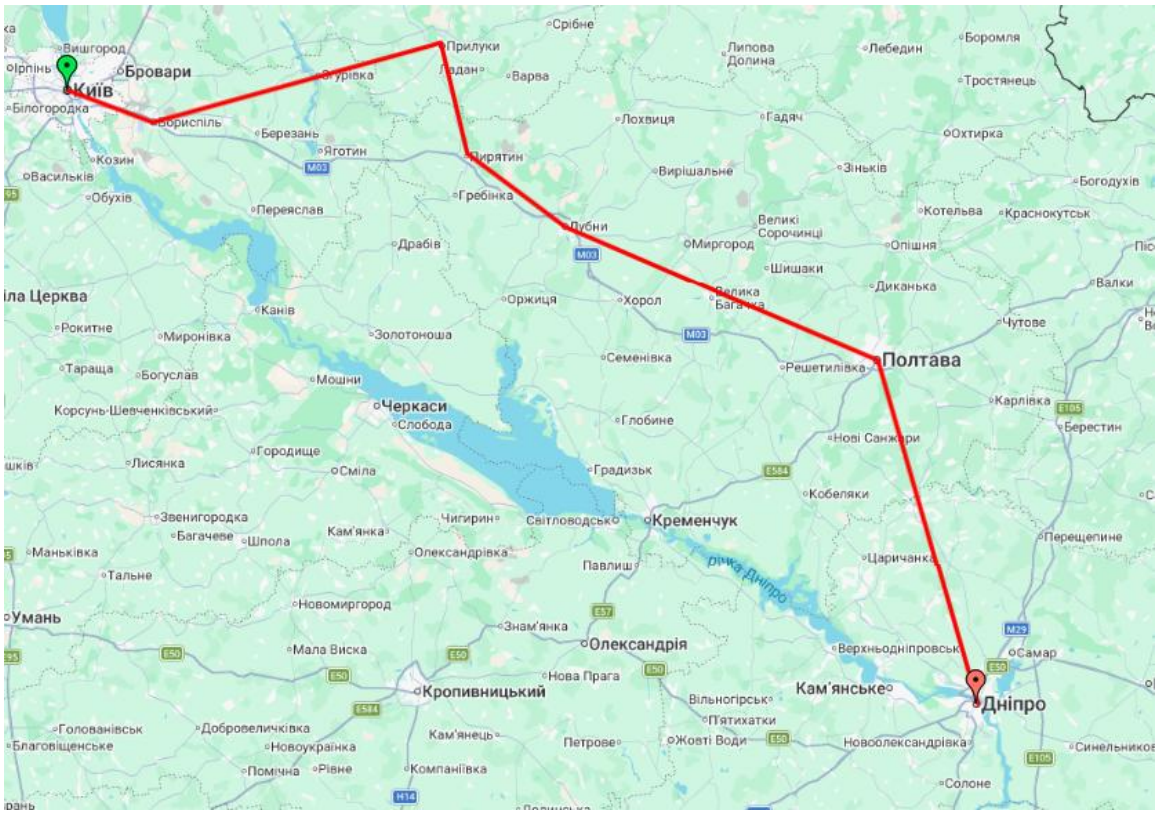

Рисунок 5.8 – Альтернативний маршрут 1, маршрут Київ – Дніпро

Маршрут проходить через такі міста: Київ → Бориспіль → Прилуки → Пирятин → Лубни → Полтава → Дніпро.

Цей маршрут є північним обходом – він відхиляється від основного шляху (через Черкаси, 460 км) і проходить через Полтаву. Фізично він довший на 80 км, однак загальна вартість (505 км) лише на 45 км перевищує вартість основного маршруту (460 км). Це свідчить про те, що ділянки через Прилуки,

Пирятин та Лубни мають добрі швидкісні характеристики.

Маршрут 3 (рисунок 5.9) має такі характеристики:

– фізична довжина – 613 км;
– штраф – 0 км;
– загальна вартість – 503 км.

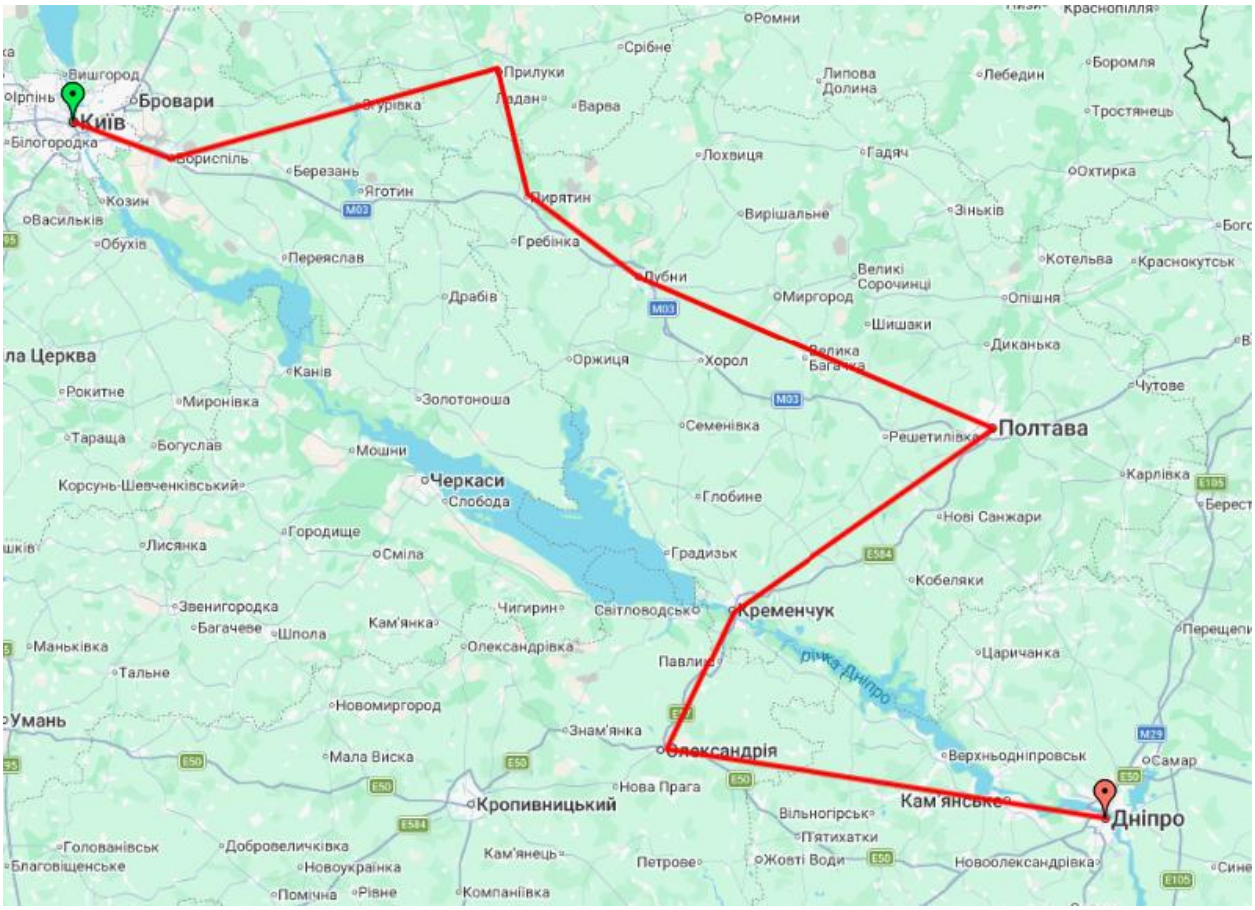

Рисунок 5.9 – Альтернативний маршрут 2, маршрут Київ – Дніпро

Маршрут проходить через такі міста: Київ → Бориспіль → Прилуки → Пирятин → Лубни → Полтава → Кременчук → Олександрія → Дніпро.

Це найдовший фізичний шлях (613 км), однак його загальна вартість (503 км) виявилася майже такою ж, як у маршруту 2, і лише трохи більшою, ніж в основного. Така ситуація характерна, коли на довгих ділянках (наприклад, Кременчук–Олександрія–Дніпро) діють високі швидкісні ліміти або відсутні зниження через міські агломерації. Отже, ця альтернатива може

бути корисною, якщо основна дорога через Черкаси перевантажена або перекрита.

Алгоритм Єна успішно згенерував по два альтернативні маршрути для кожного сценарію. У таблиці 5.5 зведено їхні основні характеристики.

Таблиця 5.5 – Альтернативні маршрути для трьох сценаріїв

Сценарій	Маршрут	Фізична довжина, км	Штраф, км	Загальна вартість, км	Рисунок
Харків – Львів	2	1058	0	915	5.4
Харків – Львів	3	1078	0	738	5.5
Харків – Одеса	2	823	0	680	5.6
Харків – Одеса	3	823	0	623	5.7
Київ – Дніпро	2	540	0	505	5.8
Київ – Дніпро	3	613	0	503	5.9

Аналіз показує, що найкоротший за фізичною відстанню маршрут не завжди є найкращим за інтегральною вартістю. Запропонований метод дозволяє логісту ознайомитися з кількома варіантами, врахувати штрафи (хоча в наведених альтернативах вони нульові) та обрати компромісний шлях. Наявність таких альтернатив є критичною при плануванні негабаритних перевезень, коли через раптові перекриття або зміну умов доводиться оперативно перемикатися на резервний маршрут.

5.4 Переваги та недоліки

На основі проведеного тестування (розділи 5.1–5.4) та аналізу програмної реалізації можна виділити сильні та слабкі сторони запропонованого методу [21]. Це допомагає зрозуміти, у яких випадках метод працює добре, а де потрібні доопрацювання.

Переваги:

1. Врахування жорстких інфраструктурних обмежень: на відміну від класичного алгоритму Дейкстри, який будує маршрут виключно на основі

довжини ребер, модифікований алгоритм перевіряє параметри транспортного засобу (висоту, ширину) перед додаванням ребра. У коді це реалізовано рядком:

Лістинг 5.1 – Приклад врахування жорстких обмежень

```
if vehicle_height > edge.max_height or vehicle_width >
edge.max_width:
    continue
```

Якщо, наприклад, вантажівка має висоту 5,0 м, а міст на ділянці Київ–Житомир пропускає лише до 4,8 м – ребро ігнорується. У сценарії Харків–Львів це змушує алгоритм обирати південний обхід (рисунок 5.5(в)), хоча він на 80 км довший. Це головна перевага: маршрут завжди буде фізично здійсненним.

2. Гнучке налаштування пріоритетів через штрафи: штрафні функції дозволяють зробити деякі ділянки «небажаними», але не забороняти їх повністю [19]. Наприклад, у коді є ребра з penalty=10 (Дніпро–Запоріжжя) та penalty=25 (Вінниця–Київ). Завдяки цьому алгоритм може уникати центрів міст, доріг з поганим покриттям або ділянок з інтенсивним рухом, не виключаючи їх з розгляду. Користувач бачить реальну довжину та окремий рядок «Штраф», що дозволяє свідомо обирати маршрут (рисунки 5.1–5.3 у бічній панелі).

3. Побудова альтернативних маршрутів (алгоритм Єна): стандартна Дейкстра повертає лише один шлях. Алгоритм Єна (реалізований у функції yen_k_shortest_paths) знаходить до трьох різних маршрутів. Це критично для логістики: якщо основний маршрут перекриють на ремонт, водій має запасний варіант. У тесті Харків–Львів альтернативи через Суми та південний обхід дають змогу вибрати компроміс між довжиною та уникненням штрафів.

4. Збереження ефективності базового алгоритму: модифікації не змінюють основну логіку Дейкстри – додаються лише перевірки if та додавання penalty. Складність залишається O((V+E) log V) з використанням

купи. Для графа з ≈50 вершинами та 100 ребрами час пошуку трьох маршрутів становить менше 0,5 секунди, що цілком прийнятно для вебзастосунку.

5. Можливість легко розширювати набір обмежень: у класах Vertex та Edge закладено лише висоту та ширину, але структура дозволяє додати будь-які інші поля: max_length, max_weight, max_axle_load, time_window_start, time_window_end тощо. Умова жорсткої перевірки доповниться новим рядком, а штрафні функції можна зробити багатокомпонентними. Це робить метод універсальним для різних типів транспорту.

6. Наочна візуалізація на карті: застосунок використовує Google Maps API, що дозволяє будувати ламані лінії через проміжні вершини. Користувач одразу бачить, де проходить маршрут, які міста він охоплює. Альтернативи виділяються різними кольорами, що спрощує порівняння. Крім того, у бічній панелі відображається короткий опис «Харків → Полтава → Київ → ...», що зручно для звітності.

7. Легка адаптація до нових регіонів: граф повністю описаний у файлі graph_data.py. Щоб додати нове місто або дорогу, достатньо додати вершину та кілька ребер зі своїми параметрами. Це дозволяє використовувати розроблений метод не лише для України, а й для будь-якої іншої країни – потрібно лише оновити координати, відстані та обмеження.

Недоліки та обмеження:

1. Використання агрегованої моделі графа: у роботі застосовано спрощену модель, де вершинами є лише обласні центри та кілька проміжних точок, а ребра – прямі лінії між ними. Це призводить до похибок: реальна дорога може бути довшою через повороти, але в моделі ми враховуємо лише пряму відстань (хоча довжина задається коректно, геометрія на карті спотворена). Наприклад, лінія Київ–Полтава на карті виглядає як пряма, хоча траса М03 петляє. Це не помилка, а свідоме спрощення для зменшення кількості вершин. Для промислової системи краще використовувати дискретизовану модель (рисунок 2.2).

2. Обмежений набір жорстких обмежень: наразі реалізовано лише

перевірку висоти та ширини. Обмеження за масою, довжиною автопоїзда, радіусом повороту, часовими вікнами та екологічними зонами не враховані. У реальних перевезеннях ці параметри часто критичні. Наприклад, довгомірний автопоїзд може не вписатися в кільцеву розв'язку, але наш алгоритм цього не перевірить. Розширення потребує додавання полів до класу Edge та модифікації умови в modified_dijkstra.

3. Статичність даних: усі обмеження та ваги завантажуються один раз і не змінюються під час роботи. Це означає, що дорожні роботи, тимчасові перекриття, зміна швидкісного режиму або поява нових обмежень не відстежуються. Класичний алгоритм Дейкстри не підтримує динамічне оновлення графа; для цього потрібен окремий механізм сповіщень та перерахунку маршруту.

4. Альтернативні маршрути можуть бути неефективними: алгоритм Єна будує альтернативи шляхом блокування ребер. Це може призводити до появи маршрутів, які повертаються назад (наприклад, Полтава → Харків → Суми) або мають петлі. У поточній реалізації ми сортуємо шляхи за вартістю, але не фільтруємо їх за «логічністю». У тесті Харків–Львів альтернатива через Суми вийшла на 68 км довшою, що прийнятно, але в деяких конфігураціях графа можливі абсурдні варіанти (наприклад, Київ – Чернігів – Київ – Житомир). Це вимагає додаткової постобробки.

5. Відсутність автоматичного підбору штрафів: штрафи задаються вручну при створенні ребра (наприклад, penalty=10 для мосту). Немає автоматичного розрахунку на основі реальних даних (інтенсивність руху, якість покриття, ймовірність затору). Користувач або адміністратор системи повинен сам визначати ці значення, що може бути суб'єктивним. У майбутньому можна було б інтегрувати статистику з навігаційних сервісів.

6. Невелика кількість альтернатив (k=3): алгоритм повертає максимум три маршрути. Для складних логістичних задач може знадобитися 5–10 варіантів, особливо якщо є багато різних обмежень. Збільшення k збільшує час пошуку (теоретично $O(k \cdot V \cdot (E+V \log V))$) і може призвести до дублювання

схожих шляхів. У поточному застосунку k=3 обрано як компроміс між швидкістю та корисністю.

7. Відсутність підтримки багатокритеріальної оптимізації: вартість маршруту обчислюється як length + penalty. Це фактично два критерії, зведені в один. Якщо потрібно враховувати час, вартість палива та штрафи окремо, доведеться застосовувати методи Парето-оптимізації або зважену суму з підбором коефіцієнтів. У роботі це не реалізовано.

8. Залежність від точності геокодера: для перетворення адреси у вершину використовується Google Maps Geocoding API. Якщо користувач введе неіснуючу адресу або з помилкою, застосунок не зможе знайти відповідну вершину. Крім того, геокодер може повернути координати, найближча вершина до яких буде зовсім іншим містом (наприклад, замість села – сусіднє місто). Це обмеження можна зменшити, додавши ручне уточнення або використовуючи офлайн-базу даних.

Порівняння з аналогами:

Для об'єктивної оцінки варто порівняти розроблений метод з типовими підходами, таблиця 5.6.

Таблиця 5.6 – Порівняння з аналогами

Підхід	Облік габаритів	Альтернативи	Динаміка	Штрафи	Відкритість
Google Maps (режим вантажівки)	частково (тільки заборона в'їзду)	так	так	ні	закритий
Спеціалізовані логістичні системи (Trimble, PTV)	так (повний набір)	так	частково	ні	закритий
Класичний Дейкстра	ні	ні	ні	ні	відкритий

Як видно, метод програє динамічним системам у реагуванні на зміни,

але виграє у відкритості та можливості кастомізації. Для невеликих логістичних фірм або дослідницьких проектів цілком достатньо.

Шляхи подолання недоліків [20]:

– перейти на дискретизовану модель (розбити дороги на сегменти по 10–20 км), що підвищить точність геометрії;

– додати в клас Edge поля max_weight, max_length, time_window та розширити перевірки;

– реалізувати фоновий сервіс для оновлення графа з OpenStreetMap (наприклад, раз на добу);

– ввести фільтр «розумних» альтернатив, який відкидає маршрути із самоперетинами або надто довгим відхиленням

– автоматизувати розрахунок штрафів на основі даних про затори (з тієї ж Google Maps API);

– збільшити k до 5 або зробити параметр налаштовуваним

Незважаючи на зазначені недоліки, запропонований метод успішно виконує поставлену задачу – будує альтернативні маршрути для негабаритних перевезень з урахуванням висоти, ширини та штрафів. Він є доброю основою для подальшого вдосконалення в рамках реальної логістичної платформи.

ВИСНОВКИ

У межах магістерської кваліфікаційної роботи було проведено дослідження методів і побудови альтернативних маршрутів для негабаритних перевезень на основі модифікованого алгоритму Дейкстри та контекстного аналізу транспортних обмежень. Також було створено програмний застосунок, який реалізує запропонований підхід та дозволяє будувати маршрути з урахуванням параметрів транспортного засобу.

У процесі виконання роботи були виконані такі етапи:

– проведено аналіз предметної області негабаритних перевезень;

– досліджено сучасні методи маршрутизації та алгоритми пошуку найкоротших шляхів;

– виконано аналіз класичного алгоритму Дейкстри та визначено його обмеження у задачах негабаритних перевезень;

– розроблено модель транспортної мережі у вигляді зваженого графа;

– реалізовано механізм врахування жорстких транспортних обмежень;

– реалізовано систему м’яких обмежень на основі штрафних функцій;

– модифіковано алгоритм Дейкстри для роботи з транспортними обмеженнями;

– реалізовано побудову альтернативних маршрутів за допомогою алгоритму Єна;

– спроєктовано архітектуру програмного застосунку;

– реалізовано серверну частину системи засобами Python та Flask;

– реалізовано візуалізацію маршрутів та взаємодію з користувачем;

– проведено тестування системи на різних сценаріях побудови маршрутів;

– виконано аналіз отриманих результатів та порівняння з класичними методами маршрутизації.

Результати роботи показали, що використання модифікованого алгоритму Дейкстри у поєднанні з контекстним аналізом транспортних

обмежень дозволяє ефективно будувати допустимі маршрути для негабаритного транспорту. Врахування жорстких обмежень забезпечує виключення непридатних ділянок дороги, а система штрафів дозволяє мінімізувати використання небажаних маршрутів.

Додатковою перевагою запропонованого підходу є можливість побудови декількох альтернативних маршрутів. Це підвищує гнучкість системи та дозволяє швидко адаптуватися до змін дорожньої ситуації або появи нових обмежень.

Розроблений програмний застосунок може бути використаний як основа для створення більш складних логістичних систем у сфері негабаритних перевезень. Подальший розвиток роботи може включати інтеграцію з реальними картографічними сервісами, підтримку динамічних транспортних обмежень, врахування дорожнього трафіку в режимі реального часу та використання більш детальних моделей транспортної мережі.

При підготовці кваліфікаційної роботи була підготовлена та опублікована наукова робота: «Алгоритмічні підходи до пошуку альтернативних маршрутів у транспортних мережах. Модифікація алгоритму Дейкстри», що відбулася 27 березня 2026 року в м. Луцьк (Україна).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dijkstra E. W. A note on two problems in connexion with graphs. Numerische Mathematik. 1959. Vol. 1. P. 269–271.

2. Bellman R. On a Routing Problem. Quarterly of Applied Mathematics. 1958. Vol. 16, No. 1. P. 87–90.

3. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 3rd ed. Cambridge : MIT Press, 2009. 1312 p.

4. Hart P. E., Nilsson N. J., Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics. 1968. Vol. 4, No. 2. P. 100–107.

5. Python Software Foundation. heapq – Heap queue algorithm. Python 3.12.3 Documentation. – URL: https://docs.python.org/3/library/heapq.html (дата звернення: 15.05.2026).

6. Ren P., Qin G., Dong J., Li B., Zheng X. Improved Dijkstra algorithm with traffic rule constraints. Journal of Computer Applications. 2015. Vol. 35, No. 9. P. 2503–2507.

7. Diestel R. Graph Theory. 5th ed. Berlin : Springer, 2017. 432 p.

8. Парубець О. М. Моделювання мережевих структур на транспорті з використанням елементів теорії графів. Глобальні та національні проблеми економіки. 2015. № 3. С. 380–384.

9. Rodrigue J.-P. The Geography of Transport Systems. 5th ed. New York : Routledge, 2020. 480 p.

10. ISO 14825:2011. Intelligent transport systems – Geographic Data Files (GDF) – GDF5.0. Geneva : ISO, 2011. 110 p.

11. Yen J. Y. Finding the K Shortest Loopless Paths in a Network. Management Science. 1971. Vol. 17, No. 11. P. 712–716.

12. SciPy Developers. scipy.sparse.csgraph.yen – Yen’s K-Shortest Paths algorithm. – URL: https://scipy.github.io/devdocs/reference/generated/scipy.sparse.csgraph.yen.html (дата звернення: 15.05.2026).

13. Pallets. Flask Documentation (3.1.x). – URL:

https://flask.palletsprojects.com/ (дата звернення: 15.05.2026).

14. Google for Developers. Google Maps Platform Documentation – Web Services. – URL: https://developers.google.com/maps/documentation (дата звернення: 15.05.2026).

15. Бронза Є. С. Дослідження та розробка методів маршрутизації транспортних засобів в системі мережевого рітейлу : кваліфікаційна робота. Харків : ХНУРЕ, 2021. 75 с.

16. OpenStreetMap Contributors. Map Features – Key:access. – URL: https://wiki.openstreetmap.org/wiki/Key:access (дата звернення: 15.05.2026).

17. GIScience Research Group. openrouteservice – Tag Filtering. – URL: https://giscience.github.io/openrouteservice/technical-details/tag-filtering (дата звернення: 15.05.2026).

18. Ahuja R. K., Magnanti T. L., Orlin J. B. Network Flows: Theory, Algorithms, and Applications. Englewood Cliffs : Prentice Hall, 1993. 864 p.

19. Шевченко О. В. Дослідження методів та алгоритмів планування транспортних маршрутів : магістерська кваліфікаційна робота. Миколаїв : ЧНУ ім. Петра Могили, 2023. 78 с.

20. Blum J. A Parameterized View on Transportation Networks : Algorithms, Hierarchy, and Complexity : дис. ... Dr. rer. nat. Konstanz : University of Konstanz, 2023. 152 p.

21. Petrova R. Оцінка поточного стану та перспектив розвитку глобалізаційних процесів в Україні / Харківський національний університет радіоелектроніки. – URL: https://scholar.google.com/citations?view_op=view_citation&hl=ru&user=Cp7XSNoAAAAJ&authuser=1&citation_for_view=Cp7XSNoAAAAJ:8k81kl-MbHgC (дата звернення: 15.05.2026).

22. Серветник Д. С. Алгоритмічні підходи до пошуку альтернативних маршрутів у транспортних мережах. модифікація алгоритму Дейкстри // X International Student Scientific Conference «Globalization of Scientific Knowledge: International Cooperation and Integration of Branches of Science» (March 27, 2026). Lutsk, Ukraine. Pp. URL: https://archive.liga.science/index.php/conference-proceedings/issue/view/inter-27.03.2026.