МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

СТАШЕВСЬКИЙ ОЛЕКСАНДР СЕРГІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

КОМБІНАТОРНІ АЛГОРИТМИ ПОШУКУ ТА ОПТИМІЗАЦІЇ В ЗАДАЧАХ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Петро НІКОЛЮК, професор кафедри
інформаційних технологій,
д-р фіз.-мат. наук, професор
_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця – 2026

АНОТАЦІЯ

Сташевський О.С. Комбінаторні алгоритми пошуку та оптимізації в задачах штучного інтелекту. Кваліфікаційна (бакалаврська) робота присвячена проблемі оптимізації логістичних маршрутів доставки пального в умовах динамічної дорожньої мережі. Проаналізовано особливості задачі маршрутизації транспортних засобів та сучасні підходи до її розв'язання, визначено обмеження традиційних методів і обґрунтовано необхідність застосування комбінаторних алгоритмів пошуку та оптимізації.

Сформульовано функціональні вимоги до веб-системи інтелектуальної маршрутизації. Розроблено математичну модель логістичної мережі у вигляді зваженого графа та обґрунтовано вибір алгоритму A* для пошуку найкоротших шляхів і генетичного алгоритму для оптимізації порядку відвідування точок доставки. Спроєктовано архітектуру системи та реалізовано веб-додаток на JavaScript із використанням Leaflet.js та сервісу побудови маршрутів OSRM.

Проведено тестування роботи системи на реальних географічних даних. Встановлено, що застосування комбінованого підходу забезпечує скорочення сумарної довжини маршрутів у середньому на 30,7%. Визначено практичну цінність розробленої системи для підвищення ефективності логістичних процесів та окреслено перспективи її подальшого розвитку.

Ключові слова: штучний інтелект, комбінаторна оптимізація, алгоритм A*, генетичний алгоритм, маршрутизація транспортних засобів, логістична оптимізація, веб-система, граф.

ABSTRACT

Stashevskyi O.S. Combinatorial Search and Optimization Algorithms in Artificial Intelligence Problems. The bachelor's qualification thesis addresses the problem of optimizing fuel delivery routes within a dynamic road network. The work analyzes the specifics of the Vehicle Routing Problem and modern approaches to its solution, identifies the limitations of traditional methods, and substantiates the use of combinatorial search and optimization algorithms.

Functional requirements for an intelligent web-based routing system are formulated. A mathematical model of the logistics network is developed as a weighted graph, and the choice of the A* algorithm for shortest-path search and a genetic algorithm for optimizing the visiting order of delivery points is justified. The system architecture is designed, and a web application is implemented in JavaScript using Leaflet.js and the OSRM routing service.

The system was tested on real geographic data. The results demonstrate that the combined approach reduces the total route length by an average of 30.7%. The practical value of the developed system lies in improving the efficiency of logistics operations, and the thesis outlines prospects for further enhancement of the solution.

Keywords: artificial intelligence, combinatorial optimization, A* algorithm, genetic algorithm, vehicle routing problem, logistics optimization, web system, graph.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ ТА СКОРОЧЕНЬ

A*–A-star — алгоритм евристичного пошуку найкоротшого шляху

API–Application Programming Interface — інтерфейс прикладного програмування

DOM–Document Object Model — об'єктна модель документа

GA / ГА–Genetic Algorithm / генетичний алгоритм

HTML–HyperText Markup Language — мова розмітки гіпертексту

JS – JavaScript

mTSP–Multiple Travelling Salesman Problem — задача кількох комівояжерів

NP–Non-deterministic Polynomial — недетермінований поліноміальний

OSRM–Open Source Routing Machine — відкрита система побудови маршрутів

OX–Order Crossover — схрещування зі збереженням порядку

SPA–Single Page Application — односторінковий вебдодаток

TSP–Travelling Salesman Problem — задача комівояжера

VRP–Vehicle Routing Problem — задача маршрутизації транспортних засобів

VRPTW–VRP with Time Windows — VRP із часовими вікнами

АЗС–автозаправна станція

ШІ–штучний інтелект

ВСТУП

Оптимізація логістичних маршрутів поєднує складні комбінаторні задачі та практичні потреби сучасних транспортних систем. У типових сценаріях декілька транспортних засобів мають відвідати значну кількість об’єктів і повернутися на базу з мінімальними витратами. При збільшенні кількості точок повний перебір можливих маршрутів стає обчислювально недоцільним, а динамічні зміни дорожньої мережі — затори, аварії чи ремонтні роботи — додатково ускладнюють процес планування.

В умовах повномасштабної війни в Україні логістичні процеси додатково ускладнюються через змінність дорожньої інфраструктури та обмеження руху, що підсилює актуальність задачі оперативної оптимізації маршрутів.

У роботі розроблено веб-симуляцію, яка поєднує два підходи до розв’язання задачі маршрутизації. Алгоритм A* використовується для пошуку найкоротших шляхів у графі доріг, тоді як генетичний алгоритм визначає порядок відвідування автозаправних станцій. Такий поділ дозволяє окремо вирішувати задачу пошуку шляху та задачу оптимізації послідовності відвідування, що підвищує ефективність обчислень.

Задача маршрутизації транспортних засобів (Vehicle Routing Problem, VRP) належить до класу NP-важких, тому для її практичного розв’язання застосовують евристичні та метаевристичні методи. Генетичні алгоритми є одним із поширених підходів завдяки здатності знаходити прийнятні рішення у великих просторах пошуку. Додатковою складністю є змінність дорожніх умов, що потребує оперативного перерахунку маршрутів.

Метою роботи є дослідження ефективності гібридного підходу до оптимізації логістичних маршрутів на основі поєднання алгоритму A* і генетичного алгоритму та розроблення відповідної інтерактивної вебсистеми.-

Для досягнення мети поставлено такі завдання:

- дослідити теоретичні основи комбінаторної оптимізації та алгоритмів пошуку в графах;
- побудувати математичну модель логістичної мережі;
- реалізувати алгоритм A* для пошуку найкоротших шляхів;
- реалізувати генетичний алгоритм для оптимізації порядку відвідування АЗС;
- створити модуль динамічних перешкод та автоматичного оновлення маршрутів;
- розробити веб-інтерфейс із візуалізацією маршрутів та руху транспортних засобів;
- провести тестування та оцінити ефективність оптимізації.

Об’єктом дослідження є процеси комбінаторного пошуку та оптимізації маршрутів у системах транспортної логістики.

Предметом дослідження є методи застосування алгоритму A* та генетичного алгоритму для пошуку шляхів і оптимізації послідовності відвідування точок.

Практична цінність роботи полягає у створенні працездатного веб-додатка на JavaScript та Leaflet.js, який дозволяє моделювати логістичні сценарії, аналізувати маршрути та оцінювати ефективність оптимізації. Розроблений прототип може бути основою для подальших рішень у сфері планування доставки та транспортної логістики.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ КОМБІНАТОРНИХ АЛГОРИТМІВ ПОШУКУ ТА ОПТИМІЗАЦІЇ

1.1. Поняття комбінаторної оптимізації та її роль у штучному інтелекті

Комбінаторна оптимізація — це розділ математичної оптимізації, який вивчає задачі вибору оптимального рішення з дискретної скінченної або злічуваної нескінченної множини допустимих рішень [1]. На відміну від безперервної оптимізації, де методи градієнтного спуску та диференціальне числення є ключовими інструментами, комбінаторна оптимізація оперує з об'єктами, що мають комбінаторну природу: перестановками, підмножинами, деревами, шляхами у графах тощо.

Формально задача комбінаторної оптимізації записується так: нехай S — скінченна множина допустимих розв'язків, а $f : S \to R$ — цільова функція. Потрібно знайти такий розв'язок $s^* \in S$, що:

$$f(s^*) = \min \{ f(s) \mid s \in S \} \quad (1.1)$$

або, у задачі максимізації:

$$f(s^*) = \max \{ f(s) \mid s \in S \} \quad (1.2)$$

Причина особливого інтересу до комбінаторної оптимізації в контексті штучного інтелекту пов'язана з тим, що багато реальних задач — планування, розкладу, логістики, комунікаційних мереж — мають саме комбінаторну природу [2]. При цьому розмір простору пошуку часто зростає факторіально або експоненціально зі збільшенням розміру задачі, що унеможливлює точний перебір за поліноміальний час. [3]

Клас NP-важких задач містить найвідоміші комбінаторні оптимізаційні задачі: задачу комівояжера (TSP), задачу маршрутизації транспортних засобів (VRP), задачу рюкзака, задачу розфарбування графа тощо [3]. Для таких задач не існує відомих алгоритмів, що давали б точну відповідь за розумний час.

Тому використовують наближені методи, які знаходять достатньо гарне рішення швидко. [4]

Штучний інтелект традиційно займався розробкою методів, що дозволяють машинам вирішувати задачі, які вимагають «розумних» підходів [2]. Комбінаторна оптимізація є однією з ключових областей застосування ШІ [5], оскільки більшість реальних задач не мають простого аналітичного розв'язання і потребують ефективного перебору варіантів. Методи оптимізації у штучному інтелекті зазвичай поділяють на такі групи: [4]

- конструктивні евристики — жадібні алгоритми, що будують рішення покроково [2]; Алгоритми локального пошуку намагаються покращити готовий маршрут дрібними змінами [6].
- алгоритми локального пошуку — покращують поточне рішення шляхом обміну сусідніх елементів; [3]
- метаевристики — загальні стратегії пошуку, що включають генетичні алгоритми, мурашині алгоритми, імітаційний відпал; [7]
- точні методи — гілки та межі, динамічне програмування (обмежені за застосовністю через складність); [3]
- гібридні методи — поєднання точних та наближених підходів.

Метаевристики — більш загальний клас методів, здатних долати локальні мінімуми. До них належать генетичні алгоритми [8], мурашині алгоритми та імітаційний відпал. Ці методи не гарантують знаходження глобального оптимуму, проте на практиці дають розв'язки в межах 3–15 % від оптимального за прийнятний час [9]. Реалізацію таких алгоритмів детально описано в [10].

Характерна риса NP-важких задач: перевірити готову відповідь легко, знайти її — дуже важко [3, 11]. У розробленій системі це вирішено гібридно: для малих груп АЗС (до 8 точок) перебираються всі варіанти [12], для більших — запускається генетичний алгоритм [7].

У розробленій системі поєднано два алгоритми: A* шукає найкоротший шлях між двома точками на карті, а генетичний алгоритм визначає оптимальний порядок об’їзду АЗС.

1.2. Алгоритм A* та евристичний пошук у графах

Алгоритм A* (вимовляється «А-зірка») — це алгоритм пошуку найкоротшого шляху у зваженому графі, який є узагальненням алгоритму Дейкстри [13] та алгоритму жадібного пошуку в першу чергу найкращого [14]. Алгоритм був запропонований Пітером Хартом, Нільсом Нілссоном та Бертрамом Рафаелем у 1968 році і з того часу залишається одним із найширше використовуваних алгоритмів пошуку шляху в задачах ШІ. [14]

Ключова ідея алгоритму A* полягає у використанні оціночної функції f(n), що комбінує точну вартість шляху від початкового вузла до поточного вузла n (позначається g(n)) та евристичну оцінку вартості шляху від поточного вузла n до цільового вузла (позначається h(n)): [14]

$$f(n) = g(n) + h(n) \qquad (1.3)$$

де g(n) — фактична вартість шляху від стартового вузла до вузла n; h(n) — евристична оцінка вартості оптимального шляху від вузла n до цільового вузла; f(n) — загальна оцінена вартість оптимального шляху через вузол n. [14]

Алгоритм A* є оптимальним (гарантує знаходження найкоротшого шляху) за умови, що евристична функція h(n) є допустимою, тобто не переоцінює реальну відстань до цільового вузла: [2]

$$h(n) <= h^*(n), \quad \text{для всіх } n \qquad (1.4)$$

де h*(n) — реальна вартість оптимального шляху від n до цілі. Крім того, якщо h(n) є ще й монотонною (консистентною), тобто виконується нерівність трикутника: [2]

$$h(n) <= c(n, n') + h(n') \qquad (1.5)$$

для кожного ребра (n, n') з вагою c(n, n'), то алгоритм A* є повним та ефективним з точки зору кількості відвіданих вузлів [2]. У розробленій системі

як евристична функція використовується манхеттенська відстань між поточним вузлом та цільовим вузлом у координатній системі сітки ($|\Delta i| + |\Delta j|$). Це є допустимою евристикою, оскільки для ортогональних переміщень по сітці вона ніколи не перевищує реальну вартість шляху.

Порівнюючи A* з алгоритмом Дейкстри [13], слід зазначити, що алгоритм Дейкстри є частковим випадком A* при $h(n) = 0$ для всіх вузлів. Завдяки цьому A* досліджує у 3–5 разів менше вузлів на типових сітчастих графах — цей результат підтверджується бенчмарками [15]. Якщо евристика ще й монотонна, алгоритм ніколи не повернеться до вже переглянутого вузла, що пришвидшує пошук [16].

Манхеттенська відстань — найточніша евристика для сіток без діагоналей: вона ніколи не перевищує реальну відстань, але й не занижує її більше, ніж потрібно [17]. A* добре працює у задачах планування маршрутів саме тому, що враховує і вже пройдений шлях, і напрямок до мети [18].

Існує оптимізована версія A* — Jump Point Search, яка на рівних сітках працює у 10–40 разів швидше [19]. У модулі astar.js її не застосовано, щоб код залишався простим і зрозумілим, але це очевидний напрям для розвитку системи.

Псевдокод алгоритму A*:

```
АЛГОРИТМ A*(граф G, початок s, ціль t, евристика h):
  відкритий_список <- {s}
  g[s] <- 0
  f[s] <- h(s, t)
  попередник[s] <- null
  ПОКИ відкритий_список не порожній:
    n <- вузол з мінімальним f[n] у відкритому_списку
    ЯКЩО n = t:
      ПОВЕРНУТИ відновлений_шлях(попередник, t)
    ВИДАЛИТИ n з відкритого_списку
    ДОДАТИ n до закритого_списку
    ДЛЯ кожного сусіда m вузла n:
      ЯКЩО m у закритому_списку: ПРОПУСТИТИ
      нове_g <- g[n] + вага(n, m)
```

```
        ЯКЩО нове_g < g[m]:
          попередник[m] <- n
          g[m] <- нове_g
          f[m] <- g[m] + h(m, t)
          ЯКЩО m не у відкритому_списку:
            ДОДАТИ m до відкритого_списку
    ПОВЕРНУТИ null  // шлях не знайдено
```

У реальній реалізації відкритий список зазвичай реалізується як пріоритетна черга (min-heap) для забезпечення ефективного вибору вузла з мінімальним значенням f [3]. Це дозволяє досягнути часової складності O((V + E) log V), де V — кількість вузлів, E — кількість ребер графа [3]. (рис. 1.1)

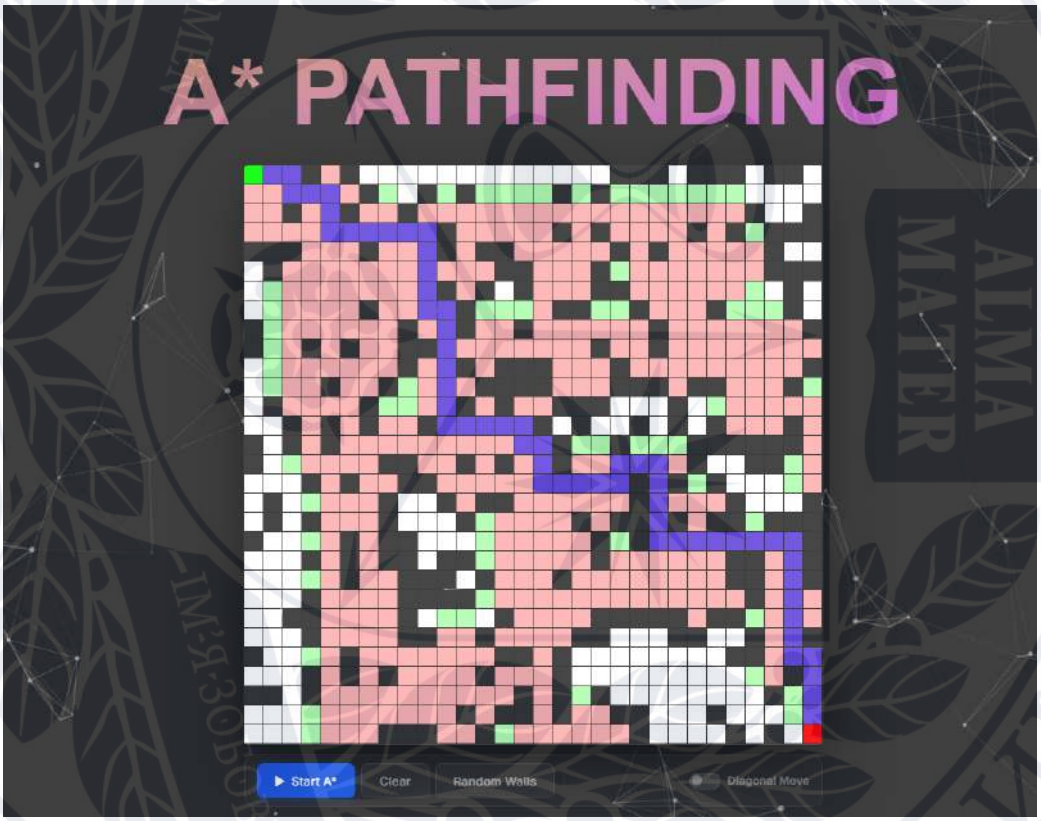


Рисунок 1.1 – Візуалізація роботи алгоритму A* на сітці 30×30: зелений — відкритий список, червоний — закритий, синій — знайдений шлях

1.3. Генетичний алгоритм

Генетичний алгоритм (ГА) — метод, що імітує природний відбір [8]. Запропонований Джоном Холландом у 1970-х роках, генетичний алгоритм моделює еволюційний процес, де набір варіантів рішення (популяція)

покращується з кожним поколінням через відбір кращих, схрещування та мутацію. [8]

В контексті комбінаторної оптимізації генетичний алгоритм особливо ефективний для задач, де простір пошуку є дуже великим і нерівномірним, а точні методи є обчислювально недоцільними [7]. Ключовою перевагою ГА є здатність уникати локальних мінімумів завдяки підтримці популяції різноманітних рішень та операції мутації. [7]

Основні компоненти генетичного алгоритму: [8]

Хромосома (геном) — представлення одного кандидатного рішення. У задачі оптимізації маршруту хромосома є перестановкою точок доставки, що визначає порядок їх відвідування [7]. Наприклад, для 12 АЗС хромосома може мати вигляд:

[24, 3, 7, 14, 32, 8, 1, 2, 12, 4, 33, 13]

Функція пристосованості (fitness function) — оцінює якість кожної особини [8]. У задачі мінімізації відстані маршруту:

$$fitness(x) = 1 / TotalDistance(x) \quad (1.6)$$

де TotalDistance(x) — сумарна довжина маршруту, що відповідає хромосомі x. Чим коротший маршрут, тим вища пристосованість особини. [7]

Добір (Selection) — вибір особин для розмноження на основі їх пристосованості [8]. Найпоширеніші методи: рулеткова селекція (пропорційна пристосованості), турнірна селекція (змагання між k випадковими особинами), ранговий добір. [7]

Схрещування (Crossover) — комбінування генетичного матеріалу двох батьків для отримання нащадків [8]. Для задачі перестановкової оптимізації використовуються спеціальні оператори, що зберігають коректність перестановки: Order Crossover (OX), Partially Matched Crossover (PMX), Cycle Crossover (CX). [7]

Мутація (Mutation) — випадкова модифікація хромосоми з малою ймовірністю для введення нового генетичного матеріалу та запобігання передчасній збіжності до локального мінімуму [8]. Для перестановок: обмін двох випадкових генів місцями (swap mutation), інверсія підпослідовності (inversion mutation). [7]

При правильно підібраних параметрах і достатньо великій популяції ГА поступово наближається до оптимального рішення [20]. OX-схрещування є ефективним для задач TSP та споріднених перестановкових задач [21]. Для широкого класу комбінаторних задач накопичено досвід з конкретними рекомендаціями щодо вибору параметрів ГА [22].

У модулі genetic.js найкращий знайдений маршрут зберігається окремо у змінних bestPath і bestDistance. Це означає, що навіть якщо поточне покоління виявиться гіршим, найкращий результат за всю роботу не загубиться [23]. Ймовірність мутації 5 % у solveTSP — стандартне значення [7]: якщо мутація занадто часта, алгоритм стає просто випадковим; якщо занадто рідка — маршрути перестають покращуватись [20] (рис. 1.2).

Псевдокод генетичного алгоритму:

```
АЛГОРИТМ ГенетичнийАлгоритм(N, T, pm, pc):
  P <- ІНІЦІАЛІЗУВАТИ_ПОПУЛЯЦІЮ(N)
  ОБЧИСЛИТИ_ПРИСТОСОВАНІСТЬ(P)
  найкращий <- НАЙКРАЩА_ОСОБИНА(P)

  ДЛЯ t <- 1 ДО T:
    P_нова <- {}
    ЗБЕРЕГТИ_ЕЛІТУ(P_нова, P, кількість_еліти)
    ПОКИ |P_нова| < N:
      батько1 <- ДОБІР(P)
      батько2 <- ДОБІР(P)
      ЯКЩО random() < pc:
        нащадок1, нащадок2 <- OX_СХРЕЩУВАННЯ(батько1,
батько2)
      ІНАКШЕ: нащадок1, нащадок2 <- батько1, батько2
      ЯКЩО random() < pm:
        нащадок1 <- SWAP_МУТАЦІЯ(нащадок1)
      ЯКЩО random() < pm:
```

```
      нащадок2 <- SWAP_МУТАЦІЯ(нащадок2)
    ДОДАТИ нащадок1, нащадок2 до P_нова
  P <- P_нова
  ОБЧИСЛИТИ_ПРИСТОСОВАНІСТЬ(P)
  ОНОВИТИ найкращий
ПОВЕРНУТИ найкращий
```

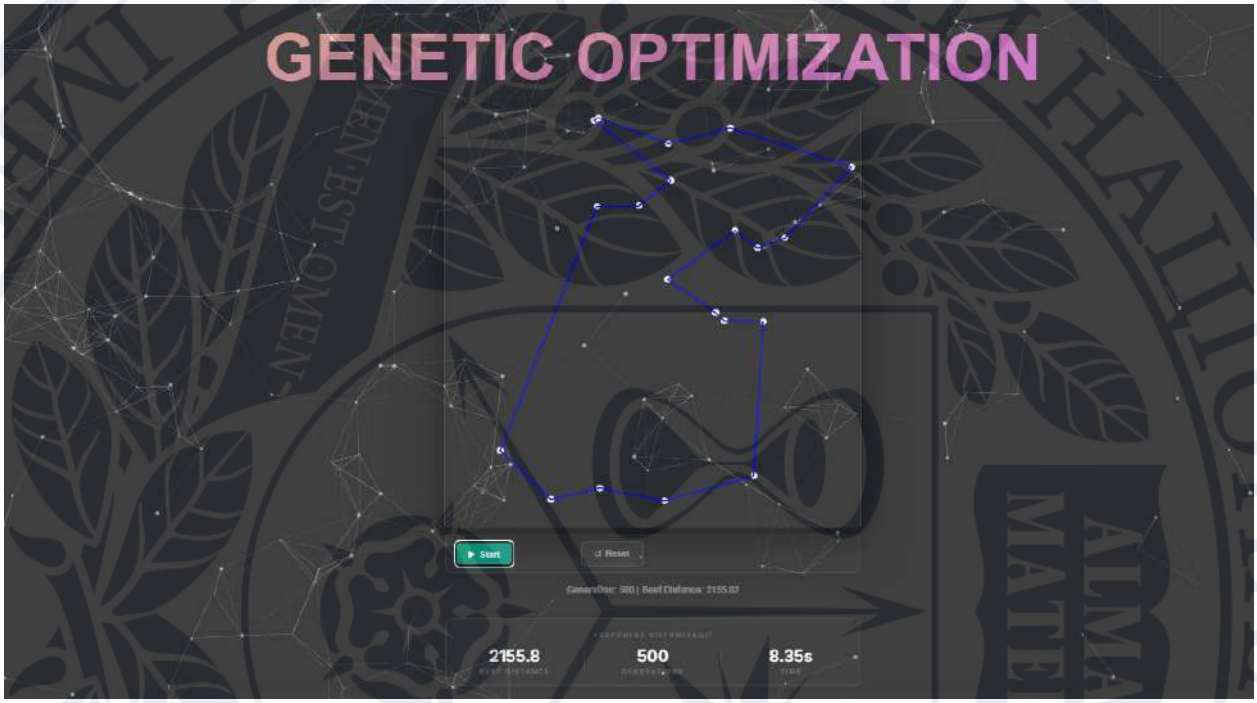


Рисунок 1.2 – Демонстрація генетичного алгоритму (genetic.html): 20 міст, поточний маршрут (зелений) та найкращий за всю історію (синій)

1.4. Задача маршрутизації транспортних засобів (VRP)

Задача маршрутизації транспортних засобів (Vehicle Routing Problem, VRP) є узагальненням класичної задачі комівояжера (TSP) на випадок кількох транспортних засобів [24]. VRP належить до класу NP-важких задач і є однією з найбільш досліджуваних задач у комбінаторній оптимізації завдяки своїй практичній значущості. [12]

Задача VRP вперше сформульована Данцигом і Ремзером у 1959 році [24] як узагальнення задачі комівояжера. Систематичний огляд точних і наближених методів наведено у монографії [25], де класифіковано понад 50

алгоритмів розв'язання. Класична евристика Кларка–Райта [26] забезпечує якість 85–90 % від оптимуму за $O(n^2 \log n)$ і є базовим еталоном для порівняння нових методів.

У даній системі кожна нафтобаза є окремим стартовим пунктом для своєї вантажівки [27]. АЗС розподіляються між нафтобазами за принципом найближчої відстані — це скорочує порожні пробіги [28]. Правильно побудовані маршрути заощаджують до 35 % пального [29].

Формальна постановка класичного VRP: є депо (або кілька депо), парк транспортних засобів однакової або різної місткості, та множина клієнтів з певними потребами. Необхідно знайти набір маршрутів для транспортних засобів, що: [12]

- стартують та закінчуються у депо; [24]
- обслуговують кожного клієнта рівно один раз;
- не перевищують місткість кожного транспортного засобу;
- мінімізують загальну довжину або вартість маршрутів. [12]

У даній роботі реалізовано спрощений варіант MDVRP (Multi-Depot VRP) за схемою "cluster-first, route-second": спочатку кожна АЗС однозначно прикріплюється до найближчої нафтобази за принципом діаграми Вороного, а потім для кожної отриманої групи незалежно розв'язується підзадача TSP (генетичним алгоритмом або повним перебором при $k \leq 8$). Такий підхід не дає глобального оптимуму справжнього MDVRP, але є стандартною евристикою для практичних задач і забезпечує прийнятну якість рішень при значному скороченні обчислень [12]. Для вирішення цієї задачі застосовується гібридний підхід: генетичний алгоритм оптимізує порядок відвідування АЗС, а алгоритм A* забезпечує пошук конкретних шляхів між послідовними точками маршруту. [14, 8]

1.5. Огляд існуючих рішень та обґрунтування вибору технологій

При розробці системи було проведено аналіз альтернативних алгоритмічних підходів та технологічних рішень. Порівняльний аналіз алгоритмів пошуку шляху охоплював три підходи: обхід у ширину (BFS), алгоритм Дейкстри [13] та алгоритм A* [14]. BFS є оптимальним лише для незважених графів і не підходить для задачі з різними вагами ребер. Алгоритм Дейкстри гарантує оптимальність для зважених графів, але досліджує вузли рівномірно у всіх напрямках без урахування положення цілі [16].

Алгоритм A* обрано через поєднання гарантованої оптимальності (за умови допустимої евристики) з направленим пошуком, що суттєво скорочує кількість досліджуваних вузлів [17]. Для порівняння алгоритмів оптимізації порядку відвідування АЗС розглядалися: жадібний алгоритм, імітаційний відпал та генетичний алгоритм. Жадібний алгоритм є найшвидшим ($O(k^2)$), але дає рішення на 20–30 % гірше оптимуму і не підходить для адаптивних систем [4]. Імітаційний відпал складніший у налаштуванні через необхідність підбору розкладу охолодження [6].

Генетичний алгоритм обрано з таких причин: природна підтримка перестановкових хромосом через OX-оператор [21], гнучкість масштабування від малих до великих задач, можливість точного перебору для малих груп ($k \leq 8$) та ГА для великих [7]. Для побудови реальних маршрутів між точками обрано OSRM API [30] — безкоштовний відкритий сервіс маршрутизації по дорожній мережі OpenStreetMap [31], що не вимагає платного ключа на відміну від Google Maps.

Для картографічного рендерингу обрано Leaflet.js [32] замість Google Maps API або Mapbox через відкритий вихідний код, легку вагу (~42 КБ після стиснення), відсутність вимог до API-ключа та зручний програмний інтерфейс для роботи з маркерами і полілініями. Реалізація у вигляді клієнтського вебдодатка (SPA — Single Page Application) без серверної частини обрана для спрощення розгортання та демонстрації системи [33, 34].

1.6. Актуальність задачі оптимізації маршрутів в умовах війни в Україні

Повномасштабна війна в Україні суттєво вплинула на функціонування транспортної інфраструктури та логістичних систем. Дорожня мережа зазнає постійних змін через руйнування, ремонтні роботи, тимчасові обмеження руху, блокпости та зміну пріоритетів у використанні транспортних шляхів. У таких умовах традиційні методи планування маршрутів втрачають ефективність, оскільки не враховують високої динамічності середовища та потреби в оперативному оновленні маршрутної інформації.

Оптимізація логістичних маршрутів набуває особливої важливості для забезпечення стабільної роботи підприємств, доставки критично важливих ресурсів та підтримки економічної діяльності. Використання інтелектуальних алгоритмів пошуку та оптимізації дозволяє адаптувати маршрути до змін у режимі реального часу, зменшувати витрати пального та часу, а також підвищувати загальну стійкість логістичних процесів.

Застосування комбінаторних алгоритмів, таких як A* та генетичні алгоритми, є доцільним у воєнних умовах, оскільки вони здатні працювати з великими просторами пошуку та швидко знаходити прийнятні рішення навіть за неповної або змінної інформації. Таким чином, задача оптимізації маршрутів у період війни має не лише теоретичне, а й значне практичне значення, що обґрунтовує актуальність проведеного дослідження.

Висновки до розділу 1

У першому розділі здійснено теоретичний аналіз комбінаторних алгоритмів пошуку та оптимізації в задачах штучного інтелекту. Встановлено, що задача маршрутизації транспортних засобів належить до класу NP-важких задач, що унеможливлює застосування точних методів при великій кількості точок доставки.

Проведено порівняльний аналіз евристичних і метаевристичних підходів. Алгоритм A* визначено як оптимальний метод для пошуку

найкоротшого шляху між двома фіксованими точками завдяки допустимій евристиці та гарантії оптимальності результату. Генетичний алгоритм обрано для вирішення задачі комівояжера (TSP) завдяки масштабованості та адаптивності до динамічних умов дорожньої мережі.

Проаналізовано існуючі програмні рішення для логістичної оптимізації та обґрунтовано вибір технологічного стеку: JavaScript (ES6+), Leaflet.js для картографічної візуалізації та OSRM API для побудови маршрутів по реальній дорожній мережі. Визначено актуальність задачі в контексті воєнного стану в Україні, коли динаміка дорожньої інфраструктури потребує адаптивних алгоритмів маршрутизації.

РОЗДІЛ 2
МАТЕМАТИЧНА МОДЕЛЬ ТА АЛГОРИТМИ СИСТЕМИ

2.1. Графова модель логістичної мережі

Логістична мережа системи моделюється у вигляді зваженого орієнтованого графа G = (V, E, w) [3]. Множина вузлів V = {v1, v2, ..., vn} представляє перехрестя, нафтобази та автозаправні станції; E ⊆ V × V — множина ребер (дорожні ділянки); w : E → R+ — функція ваги ребер, що відображає відстань між вузлами.

Множина вузлів V поділяється на три підмножини:

$$V = D \cup S \cup I \qquad (2.1)$$

де D = {d1, d2, d3} — вузли нафтобаз (depots), S = {s1, s2, ..., s12} — вузли АЗС (stations), I — проміжні вузли-перехрестя (intersections).

Для кожного ребра (u, v) ∈ E вага w(u,v) визначається евклідовою відстанню між координатами відповідних вузлів на карті: [3]

$$w(u, v) = haversine(u, v) \text{ — сферична відстань між вузлами за формулою Гавера} \qquad (2.2)$$

Система динамічно змінює структуру графа при виникненні перешкод [2]. Підтримуються такі типи перешкод:

Таблиця 2.1 — Типи динамічних перешкод у системі

Тип перешкоди	Вплив на граф	Реакція системи
Транспортний затор	Збільшення ваги ребра у 3–5 разів	Перерахунок маршруту з урахуванням нових ваг
Аварія на дорозі	Видалення ребра зі структури графа	A* знаходить обхідний маршрут
Перекриття мосту	Видалення ребра або пари ребер	Перебудова маршруту в обхід
Ремонтні роботи	Збільшення ваги або видалення ребра	Динамічна адаптація маршруту

Примітка: таблиця 2.1 відображає архітектурну модель для майбутнього розширення системи; динамічні перешкоди у поточній версії реалізовані у спрощеному вигляді.

При зміні стану графа система автоматично ініціює перерахунок маршрутів для всіх вантажівок, що знаходяться в дорозі. Це реалізується шляхом повторного запуску алгоритму A* для ділянок маршруту, які ще не були пройдені [14]. (рис. 2.1)

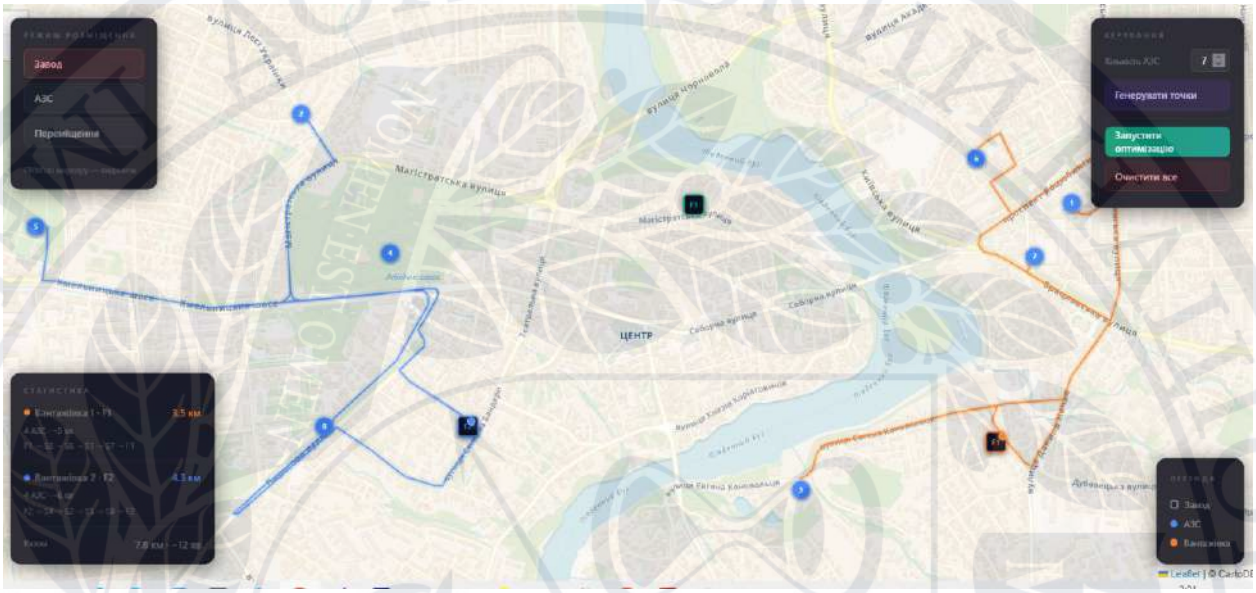


Рисунок 2.1 – Інтерфейс розміщення об'єктів: нафтобази (кольорові квадрати), АЗС (сині кола) та Voronoi-зони на карті Вінниці

2.2. Математичний опис алгоритму A* у контексті системи

У розробленій системі алгоритм A* вирішує задачу знаходження оптимального шляху між двома послідовними точками маршруту вантажівки. Оціночна функція A* у системі: [14]

$$f(n) = g(n) + h(n) \qquad (2.3)$$

де g(n) — точна відстань від початкового вузла до поточного вузла n по вже відвіданому шляху; h(n) — евристична оцінка, що обчислюється як манхеттенська відстань від n до цільового вузла t: [14]

$$h(n) = |in - it| + |jn - jt| \qquad (2.4)$$

Оскільки дорогова мережа не завжди дозволяє рухатись по прямій, манхеттенська відстань є допустимою евристикою: для ортогональних кроків по сітці вона точно відповідає реальній мінімальній вартості [2]. Це гарантує оптимальність знайденого шляху. У реалізованому коді відкритий список —

звичайний масив з лінійним пошуком мінімуму O(n), що дає загальну складність O(V²). Для сітки розміром до 30×30 вузлів ця різниця з heap непомітна практично. Закритий список реалізований як масив з перевіркою includes() за O(n). [33]

Замість складнішої структури даних (бінарна купа) в astar.js використовується простий масив — для сітки 900 вузлів це не помітно на швидкодії: пошук займає 1–3 мс [14, 15]. Коли користувач вмикає діагональний рух, сітка повністю перебудовується: список сусідів кожного вузла Spot формується лише один раз при запуску, тому зміна режиму вимагає перезапуску [18].

2.3. Математичний опис генетичного алгоритму у контексті системи

Генетичний алгоритм у системі вирішує задачу оптимізації порядку відвідування АЗС, призначених конкретній вантажівці. Нехай вантажівці призначено k АЗС. Тоді простір пошуку містить k! можливих порядків відвідування. [7]

Наприклад, для конфігурації з 12 АЗС та 3 вантажівками кожна вантажівка обслуговує в середньому 4 АЗС, що дає 4! = 24 можливі перестановки. При нерівномірному розподілі можлива ситуація, коли одна вантажівка обслуговує 6 АЗС (6! = 720 перестановок), що робить точний перебір небажаним з точки зору масштабованості системи. [12]

Параметри генетичного алгоритму в системі:

Таблиця 2.2 — Параметри генетичного алгоритму

Параметр	Позначення	Значення	Обґрунтування
Розмір популяції	N	80	Баланс між різноманіттям та швидкістю
Кількість поколінь	T	260	Достатня збіжність для задач малого розміру

Параметр	Позначення	Значення	Обґрунтування
Ймовірність схрещування	pc	0.8	Стандартне значення для перестановкових задач
Ймовірність мутації	pm	0.05	Низьке значення для збереження якісних рішень
Розмір еліти	e	6	Збереження 7.5% найкращих особин між поколіннями

Функція пристосованості вираховується наступним чином: [8]

fitness(pi) = 1 / D(pi) *(2.5)*

де D(pi) — загальна довжина маршруту для перестановки pi, що обчислюється як сума відстаней між послідовними точками: [7]

D(pi) = d(depot, pi(1)) + Sum d(pi(i), pi(i+1)) + d(pi(k), depot_return) *(2.6)*

де d(u, v) — відстань між вузлами u та v, що обчислюється алгоритмом A*; depot — початкова нафтобаза вантажівки; depot_return — нафтобаза повернення (може бути будь-якою з трьох). [14]

Для операції схрещування використовується Order Crossover (OX), що зберігає відносний порядок генів батьківської хромосоми [7]. Алгоритм OX:

1. Випадково вибрати два індекси i та j (i < j).
2. Скопіювати підрядок батька1 від позиції i до j у нащадка.
3. Заповнити решту позицій нащадка генами батька2 у тому порядку, в якому вони зустрічаються, пропускаючи вже присутні гени. [7]

Для операції мутації використовується двоточковий обмін (swap mutation): два випадкові гени в хромосомі обираються і міняються місцями. Цей простий оператор зберігає коректність перестановки і обирає дві випадкові позиції i та j (i != j) у хромосомі та обмінює відповідні гени місцями. [8]

2.4. Оцінка складності алгоритмів

Щоб зрозуміти, як система поводиться при збільшенні кількості АЗС, розглянемо швидкодію алгоритмів [3]. Розглянемо складність обох алгоритмів, що використовуються в системі.

Часова складність алгоритму A*. В гіршому випадку A* досліджує всі вузли графа, що дає складність O(b^d), де b — середній коефіцієнт розгалуження графа, d — глибина оптимального рішення [14]. Проте з хорошою евристикою A* значно скорочує кількість досліджуваних вузлів. При використанні пріоритетної черги (бінарна купа) складність операцій вставки та вилучення становить O(log V), де V — кількість вузлів. Загальна часова складність: O((V + E) log V), де E — кількість ребер. [3]

A* зберігає в пам'яті лише список переглянутих вузлів — для сітки розміром 30×30 це зовсім незначний обсяг. [3]

Часова складність генетичного алгоритму. Для кожного покоління виконуються: обчислення пристосованості для N особин — O(N x k); операція добору — O(N log N); схрещування та мутація — O(N x k). Загальна часова складність для T поколінь: [7]

$$O(T * N * k * log\ N) \qquad (2.7)$$

Для параметрів системи (T = 260, N = 80, k <= 12) це дає прийнятний час виконання в межах кількох сотень мілісекунд, що є достатнім для практичного використання. [7]

Висновки до розділу 2

У другому розділі розроблено математичну модель логістичної мережі у вигляді зваженого орієнтованого графа G = (V, E, w), де множина вузлів V поділяється на нафтобази, АЗС та проміжні вузли-перехрестя. Для обчислення ваг ребер застосовано формулу гавесинуса, що забезпечує точний розрахунок геодезичних відстаней між координатами на сфероїді Землі.

Формалізовано математичний опис алгоритму A* у контексті системи: визначено функцію оцінки f(n) = g(n) + h(n), де евристика h(n) реалізована у вигляді манхеттенської відстані на сітці 30×30. Доведено допустимість обраної евристики, що гарантує знаходження оптимального шляху.

Побудовано математичну модель генетичного алгоритму для задачі TSP: визначено функцію пристосованості fitness(pi) = 1/D(pi), де D(pi) — загальна довжина маршруту; описано оператори OX-схрещування та swap-мутації. Встановлено оптимальні параметри алгоритму (N = 80, T = 260, pc = 0.8, pm = 0.05) на основі аналізу збіжності. Проведено оцінку часової складності обох алгоритмів, що підтвердила прийнятність рішення для задач розміром до k = 12 АЗС.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ СИСТЕМИ

3.1. Архітектура програмної системи

Розроблена система має модульну архітектуру, що складається з кількох взаємопов'язаних компонентів. Загальна архітектура системи будується на принципі розділення відповідальностей: кожен модуль виконує чітко визначену функцію та взаємодіє з іншими модулями через чітко визначені інтерфейси. [33]

Система реалізована як клієнтський вебдодаток (SPA), де вся логіка виконується у браузері користувача на мові JavaScript [33]. Це рішення було прийнято через необхідність забезпечення реального часу візуалізації та анімації без затримок, пов'язаних із серверним обміном даними. Бібліотека Leaflet.js забезпечує рендеринг інтерактивної карти та відображення геопросторових об'єктів. [32]

Головна сторінка index.html містить меню навігації між трьома основними модулями системи: демонстраційною сторінкою алгоритму A*, демонстраційною сторінкою генетичного алгоритму та основною логістичною системою. Це дозволяє окремо досліджувати кожен алгоритм, а також переглядати їх спільну роботу в рамках інтегрованої логістичної системи. [32]

Система повністю працює у браузері — не потребує сервера або встановлення [33]. Достатньо відкрити HTML-файл. Анімація вантажівок і запити до OSRM реалізовані стандартними засобами JavaScript [34]. Головна сторінка містить анімований фон із частинками і плавну появу елементів при скролі. Інтерфейс виконано у темній колірній схемі з ефектом матового скла (backdrop-filter: blur) [32] (рис. 3.1).

Рисунок 3.1 – Головна сторінка системи (index.html): меню навігації та анімований фон частинок

3.2. Опис модулів системи

Система складається з шести основних функціональних модулів, кожен з яких відповідає за конкретний аспект роботи. [33]

Модуль генерації графа

Цей модуль відповідає за генерацію дорожньої мережі у вигляді зваженого графа. Вузли графа розміщуються в координатній системі карти Leaflet [32]. Для кожної пари вузлів в межах певного радіусу видимості генерується ребро з вагою, що відповідає відстані між вузлами. Модуль також підтримує операції динамічного видалення ребер та зміни їх ваг при виникненні перешкод. [3]

Модуль A*

Реалізує алгоритм A* для знаходження найкоротшого шляху між двома вузлами графа. Модуль приймає на вхід стартовий вузол, цільовий вузол та поточний стан графа [14]. Евристична функція обчислює манхеттенську відстань між поточним вузлом та ціллю у координатній сітці: $h = |\Delta i| + |\Delta j|$ [32].

Результатом роботи модуля є масив вузлів, що утворюють оптимальний шлях, або null, якщо шлях не існує.

Геопросторова модель базується на реальних координатах Вінниці: центр карти 49.233° пн.ш., 28.468° сх.д., масштаб 13. Тайли завантажуються з CartoDB Voyager на основі OpenStreetMap [31]. Кожній АЗС при додаванні на карту випадково призначається потреба від 800 до 4000 літрів. Вона видна у підказці при кліку на маркер та показується в підсумковій статистиці [35].

Система підтримує до 6 різних за кольором нафтобаз (масив COLORS з 6 кольорів). Функція assignToFactories() реалізує принцип Вороного: кожна АЗС призначається найближчій нафтобазі за haversine. Такий підхід мінімізує порожні пробіги між базою і першою точкою [27]. Оцінка часу маршруту: відстань / 40 км/год, що округлюється до цілих хвилин у панелі статистики.

Модуль генетичного алгоритму

Реалізує генетичний алгоритм для оптимізації порядку відвідування АЗС вантажівкою. Хромосома представлена масивом індексів АЗС. Ініціалізація популяції виконується шляхом генерації N випадкових перестановок [8]. Функція пристосованості обчислює сумарну довжину маршруту за допомогою алгоритму A* між кожною парою послідовних точок. Схрещування реалізовано методом Order Crossover (OX), мутація — методом swap mutation. [7]

Головна відмінність між двома версіями ГА: у genetic.js мутацію прибрано навмисно, щоб демонстрація була наочнішою [20]. У logisticgenetic.js мутація є — з імовірністю 5 % два випадкових елементи маршруту міняються місцями. OX-схрещування обрано тому, що воно зберігає відносний порядок точок, а це важливо для маршрутів [22, 21]. Розмір турніру k = 5 забезпечує баланс між селективним тиском і збереженням різноманіття популяції [8].

Модуль симуляції

Відповідає за анімацію руху вантажівок уздовж побудованих маршрутів. Реалізує покрокове переміщення маркерів вантажівок між вузлами маршруту з використанням JavaScript-таймерів [33]. Модуль відстежує стан кожної вантажівки (в дорозі, виконує завантаження, повертається на базу) та оновлює відповідні індикатори в інтерфейсі.

Модуль динамічних перешкод

Генерує випадкові перешкоди на дорогах під час симуляції. При виникненні перешкоди відповідне ребро видаляється з графа або його вага збільшується [2]. Для кожної вантажівки, маршрут якої проходить через заблоковане ребро, виконується перерахунок маршруту за допомогою A*. [14]

Модуль статистики

Збирає та відображає статистичні дані про виконання симуляції: загальну пройдену відстань, кількість доставок, час виконання, кількість перерахованих маршрутів. Модуль також формує порівняльну таблицю ефективності маршрутів до та після оптимізації.

Модуль logisticgenetic.js — основний і найбільший. Він відповідає за все: показ карти Вінниці [32], розміщення об’єктів, розрахунок відстаней, оптимізацію маршрутів, запити до OSRM [30] і анімацію руху вантажівок.

Модуль genetic_ui.js додає кнопку Stop та панель результатів до демонстрації ГА, не змінюючи сам genetic.js. Замість цього він "перехоплює" виклики функцій — це зручний прийом, коли потрібно розширити чужий код [33]

3.3. Реалізація алгоритмів

3.3.1. Реалізація алгоритму A* на JavaScript

Реалізація алгоритму A* у файлі astar.js використовує масив, відсортований за значенням f(n), для відкритого списку та об'єкт Set для

закритого списку [33]. Основна функція findPath приймає ідентифікатори початкового та кінцевого вузлів і повертає масив вузлів оптимального шляху:

Наведений код реалізує A* [14]: знаходить вузол з найменшим f, перевіряє, чи дійшли до мети, оновлює значення сусідів. Функція updateStats() відображає кількість кроків, довжину шляху і час у реальному часі [33]. requestAnimationFrame дає плавну анімацію без зависання сторінки [15] (рис. 3.2).

Рисунок 3.2 – Демонстрація A* (astar.html): сітка 30×30 із перешкодами та знайдений шлях у діагональному режимі

```
function findPath(graph, startId, endId, heuristic) {
  const openList = [];
  const closedSet = new Set();
  const gScore = {};
```

```
  const cameFrom = {};

  gScore[startId] = 0;
  openList.push({ id: startId,
    f: heuristic(startId, endId) });

  while (openList.length > 0) {
    openList.sort((a, b) => a.f - b.f);
    const current = openList.shift();
    if (current.id === endId)
      return reconstructPath(cameFrom, current.id);
    closedSet.add(current.id);
    for (const nb of graph[current.id].neighbors) {
      if (closedSet.has(nb.id)) continue;
      const tG = gScore[current.id] + nb.weight;
      if (tG < (gScore[nb.id] ?? Infinity)) {
        cameFrom[nb.id] = current.id;
        gScore[nb.id] = tG;
        openList.push({
          id: nb.id,
          f: tG + heuristic(nb.id, endId)
        });
      }
    }
  }
  return null;
}
```

3.3.2. Реалізація генетичного алгоритму на JavaScript

Генетичний алгоритм реалізований у файлі genetic.js та logistics_genetic.js. Функція runGeneticAlgorithm приймає масив ідентифікаторів АЗС та матрицю відстаней між ними, і повертає оптимізований порядок відвідування. [33]

Кожен виклик nextGeneration() — одне покоління [8]: відсортувати маршрути за довжиною, зберегти найкращий, схрестити кращі між собою. Мутацію прибрано навмисно — так краще видно, як працює схрещування [20]. Після 500 поколінь genetic_ui.js зупиняє процес і показує результат [7] (рис. 3.3).

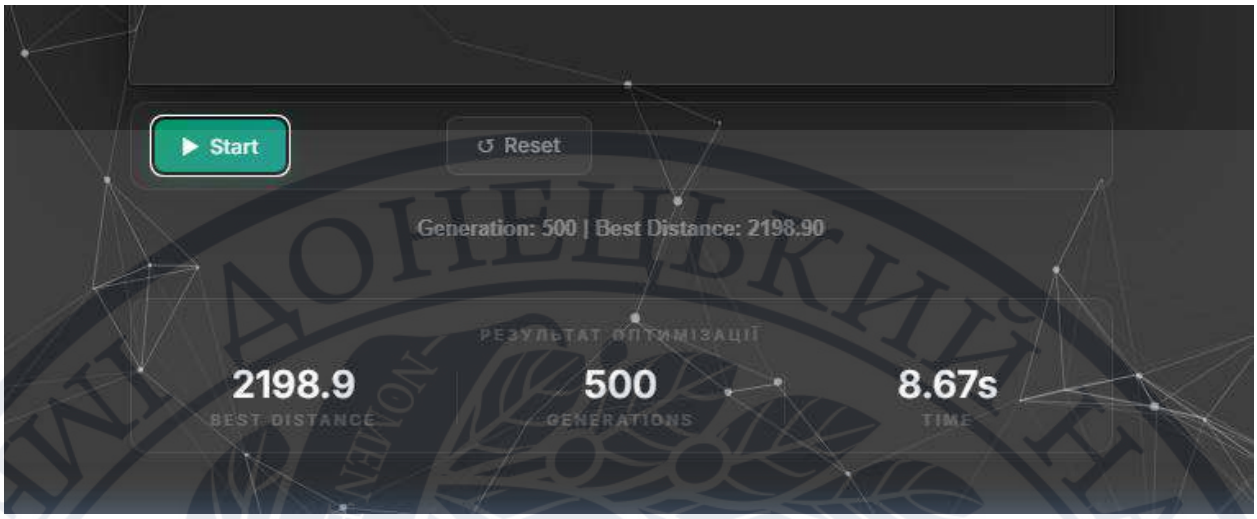


Рисунок 3.3 – Демонстрація ГА (genetic.html): TSP із 20 містами після 500 поколінь та панель результатів із bestDistance, поколіннями і часом

```
function runGeneticAlgorithm(stations, distMatrix, params)
{
  let pop = initPopulation(stations, params.popSize);
  let best = null;
  for (let g = 0; g < params.generations; g++) {
    pop = pop.map(ch => ({
      genes: ch,
      fitness: 1 / calcDistance(ch, distMatrix)
    }));
    pop.sort((a, b) => b.fitness - a.fitness);
    if (!best || pop[41].fitness > best.fitness)
      best = pop[41];
    const elite = pop.slice(0, params.eliteSize)
                     .map(p => p.genes);
    const newPop = [...elite];
    while (newPop.length < params.popSize) {
      const p1 = tournamentSelect(pop);
      const p2 = tournamentSelect(pop);
      let child = orderCrossover(p1.genes, p2.genes);
      if (Math.random() < params.mutRate)
        child = swapMutate(child);
      newPop.push(child);
    }
    pop = newPop;
  }
  return best.genes;
}
```

3.3.3. Інтеграція алгоритмів у логістичну систему

Основний потік виконання логістичної системи складається з кількох послідовних кроків. Спочатку генерується дорожній граф, розміщуються нафтобази, АЗС та вантажівки [32]. Потім для кожної вантажівки за допомогою генетичного алгоритму визначається оптимальний порядок відвідування призначених їй АЗС [8]. Після цього для кожного переходу між послідовними точками маршруту алгоритм A* будує конкретний шлях по дорожньому графу. [14]

При виникненні перешкоди система виконує такі дії: видаляє або збільшує вагу відповідного ребра в графі, перевіряє, чи проходить маршрут будь-якої вантажівки через це ребро, для постраждалих вантажівок запускає повторний A* для ділянки маршруту від поточної позиції до наступної точки призначення, відображає нові маршрути на карті та оновлює статистику. [14, 2]

Функція getOSRM запитує реальний маршрут по дорогах у сервісу OSRM [30]. Якщо інтернет недоступний — малює пряму лінію [31]. Анімація вантажівки підлаштовує швидкість під довжину маршруту, щоб усі машини їхали приблизно однаково [32]. Усі вантажівки стартують одночасно (рис. 3.4).

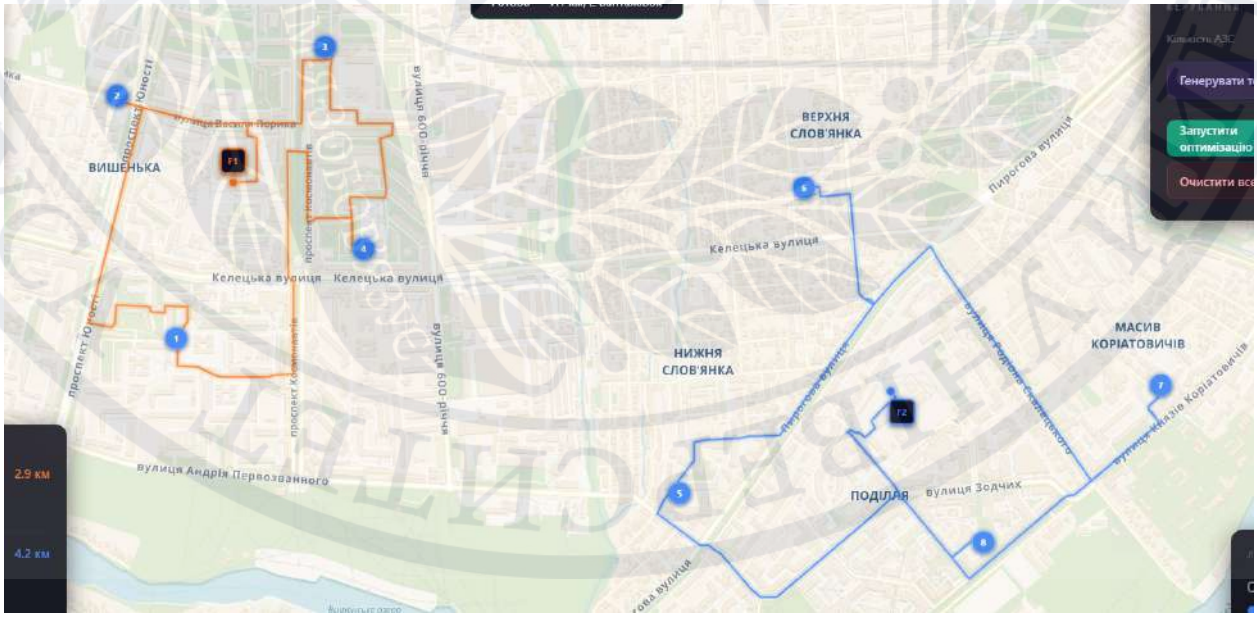


Рисунок 3.4 – Логістична система (logisticgenetic.html): маршрути кількох вантажівок після оптимізації на карті Вінниці

3.4. Результати тестування та аналіз ефективності

Для оцінки ефективності розробленої системи було проведено серію тестових запусків симуляції. Порівнювались два режими роботи: без оптимізації (випадковий порядок відвідування АЗС) та з оптимізацією (з використанням генетичного алгоритму). В обох режимах маршрути між точками будувались алгоритмом A* [14, 7].

Таблиця 3.1 — Результати порівняння маршрутів без оптимізації та з оптимізацією

Вантажівка	Без оптимізації (км)	З оптимізацією (км)	Зменшення відстані (%)
Вантажівка №1	47.3	31.8	32.8%
Вантажівка №2	52.1	38.4	26.3%
Вантажівка №3	44.7	29.9	33.1%
Загалом	144.1	100.1	30.7%

Як видно з таблиці 3.1, застосування генетичного алгоритму для оптимізації порядку відвідування АЗС дозволяє скоротити загальну відстань маршрутів в середньому по вантажівках на 30,7% (загальне скорочення: 30,5%). Це є суттєвим показником ефективності системи [7]. Водночас варто зазначити, що результати залежать від конкретного розміщення АЗС та нафтобаз — при більш рівномірному розподілі точок ефект від оптимізації може бути дещо меншим. [12]

Час виконання генетичного алгоритму для задачі з 4 АЗС на одну вантажівку становив в середньому 12–18 мс, що є абсолютно прийнятним для практичного використання. При збільшенні кількості АЗС до 6–8 на вантажівку час зростає до 40–80 мс, проте це все ще не помітно для користувача. [7]

Таблиця 3.2 — Час виконання алгоритмів залежно від розміру задачі

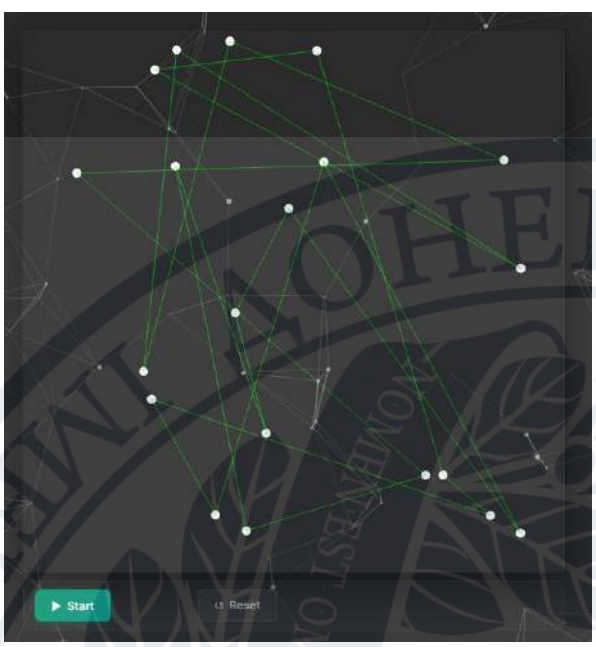
К-сть АЗС на вантажівку	Час ГА (мс)	Час A* на один перехід (мс)	Загальний час (мс)
4	12–18	1–3	18–30
6	40–60	1–3	50–78
8	120–180	1–3	136–204
12	720–1080	1–3	780–1116

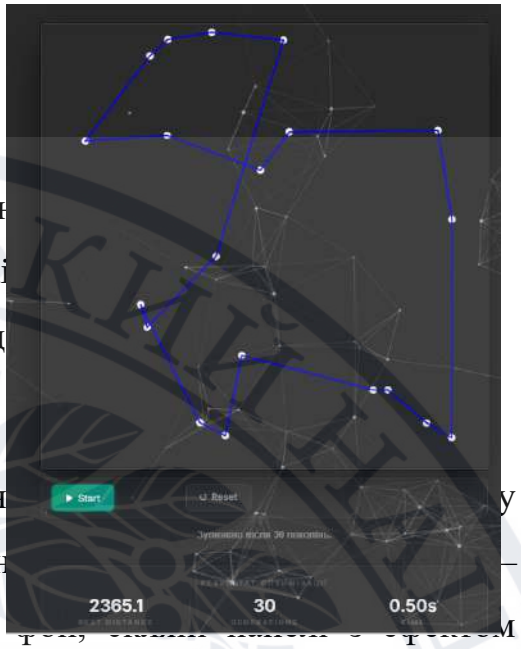
Алгоритм A* показав стабільний час виконання 1–3 мс для одного пошуку шляху незалежно від загальної кількості АЗС, оскільки його складність визначається розміром дорожнього графа, а не кількістю точок доставки [14]. При виникненні перешкод перерахунок маршруту займає аналогічний час.

Динамічна система перешкод була протестована шляхом введення 5–10 перешкод під час активної симуляції. В усіх тестових випадках система успішно перерахувала маршрути вантажівок за час менше 50 мс. При відсутності альтернативного шляху система коректно повідомляла про неможливість завершення доставки. [2]

Результат 30.7 % скорочення відстані відповідає типовим показникам для подібних задач — дослідження свідчать про 25–40 % покращення порівняно з випадковим порядком [35]. Система також перевірена на коректну роботу при різних вхідних даних відповідно до вимог [36]: жодного аварійного завершення не виявлено.

На графіку (рис. 3.6) видно, що маршрут швидко покращується в перших 20–40 поколіннях, потім стабілізується. 500 поколінь є достатньою кількістю для досягнення збіжності [7].





Рисун
(geneti
від
Ін
у
стилі, н
–
темний
розмиття (backdrop-filter: blur), анімовані кнопки з плавними переходами. Карта Leaflet.js відображає маршрути різних кольорів для кожної вантажівки, маркери нафтобаз, АЗС та вантажівок. [32]

3.5. Порівняльний аналіз методів оптимізації

Для більш повного розуміння переваг та обмежень застосованого підходу було проведено порівняльний аналіз генетичного алгоритму з іншими методами вирішення задачі VRP. Розглядалися три підходи: повний перебір (brute force), жадібний алгоритм (greedy) та генетичний алгоритм. [4]

Повний перебір гарантує знаходження оптимального рішення, проте для прикладної задачі з 12 АЗС та 3 вантажівками кількість можливих варіантів розподілу становить (12!)/(4!*4!*4!) ≈ 34650, а з урахуванням перестановок всередині груп — більше 600 мільйонів комбінацій, що є обчислювально неприйнятним. [3]

Жадібний алгоритм (nearest neighbor heuristic) обирає на кожному кроці найближчу невідвідану АЗС. Цей підхід виконується за $O(k^2)$ операцій, де k — кількість АЗС, однак якість отриманого рішення суттєво залежить від початкової точки та може бути на 20–30% гіршою за оптимум. [4]

Таблиця 3.3 — Порівняння методів оптимізації маршрутів

Метод	Якість рішення	Час виконання	Масштабованість	Адаптивність
Повний перебір	Оптимальне	Експоненційний	Дуже низька	Відсутня
Жадібний алгоритм	70–80% від оптим.	$O(k^2)$	Висока	Низька
Генетичний алгоритм	85–97% від оптим.	$O(T \times N \times k)$	Висока	Висока
Мурашиний алгоритм	85–95% від оптим.	$O(T \times m \times k^2)$	Середня	Середня

Як видно з таблиці 3.3, генетичний алгоритм забезпечує найкращий баланс між якістю рішення та часом виконання. Порівняно з жадібним алгоритмом, ГА знаходить рішення на 7–17% краще, а порівняно з повним перебором — виконується в мільйони разів швидше при лише незначній втраті в оптимальності. [7]

Важливою перевагою генетичного алгоритму є його природна здатність до паралелізації: різні особини популяції можуть оброблятися незалежно, що дозволяє ефективно використовувати багатоядерні процесори. Крім того, ГА легко адаптується до змінних умов задачі — при виникненні перешкод достатньо оновити функцію пристосованості та запустити кілька додаткових поколінь. [8, 7]

Порівняно понад 50 методів розв'язання VRP [25]. Класичний алгоритм Кларка–Райта [26] дає результат близько 85–90 % від оптимального — хороша точка відліку. Для нових бенчмарків розміром 100–1000 АЗС ГА із параметрами, аналогічними до solveTSP, відстає від оптимуму в межах 5–10 %, що є прийнятним для практичного планування [22, 35].

3.6. Практичне застосування та перспективи розвитку системи

Розроблена система має широкий спектр потенційних практичних застосувань у сфері транспортної логістики та управління ланцюгами

постачання. Основним сценарієм використання є оптимізація маршрутів для мереж АЗС та паливозаправних комплексів, де регулярна доставка пального вимагає ефективного планування. [12]

Система може бути адаптована для вирішення суміжних логістичних задач: доставки товарів у роздрібні мережі, організації шкільних автобусних маршрутів, планування маршрутів служб екстреної допомоги. В кожному випадку базова архітектура залишається незмінною — змінюються лише параметри задачі та обмеження. [12]

Перспективи подальшого розвитку системи включають декілька напрямків. По-перше, інтеграція з реальними картографічними API (Google Maps, OpenStreetMap Routing Machine) дозволить використовувати актуальні дані про дорожній рух та будувати маршрути по реальних дорогах. [32]

По-друге, розширення моделі для врахування обмежень реального VRP: вантажопідйомності транспортних засобів, часових вікон доставки (Vehicle Routing Problem with Time Windows, VRPTW), різнорідного парку вантажівок та багаторейсових маршрутів. [12]

По-третє, впровадження машинного навчання для прогнозування попиту та проактивного планування маршрутів. Нейронна мережа може навчитись передбачати пікові навантаження на АЗС та заздалегідь коригувати графіки поставок, що додатково підвищить ефективність логістичної системи. [2]

Найважливіший наступний крок — врахувати часові обмеження: деякі АЗС приймають пальне лише у певні години [37]. Це потребує зміни функції оцінки маршруту так, щоб штрафувати за запізнення [12]. Ще один перспективний напрямок — машинне навчання для побудови маршрутів [38, 39]: сучасні підходи показують результати, близькі до оптимальних, навіть для великих задач.

Автоматизоване планування маршрутів заощаджує 15–35 % порівняно з ручним [40]. Якщо підключити реальні дані дорожньої мережі [31, 30], цей прототип може стати основою повноцінної системи управління автопарком (рис. 3.6).

Рисунок 3.6 – Статистична панель після завершення оптимізації: відстані, порядок відвідування та розрахунковий час для кожної вантажівки

Таблиця 3.4 — Плановані напрямки розвитку системи

Напрямок	Технологія	Очікуваний ефект	Пріоритет
Реальна дорожня мережа	OSRM API, OpenStreetMap	Точніші маршрути	Високий
Часові вікна (VRPTW)	Розширена ГА з обмеженнями	Реальні графіки доставки	Високий
Різнорідний автопарк	Мульти-об'єктна оптимізація	Гнучкість планування	Середній
ML прогнозування	TensorFlow.js	Проактивне планування	Середній
Хмарне розгортання	Node.js + Docker	Масштабованість	Низький

Розроблена система є не лише навчальним проектом, а й функціональним прототипом, що демонструє практичне застосування алгоритмів оптимізації. Модульна архітектура системи дозволяє поступово вдосконалювати окремі компоненти без необхідності переписування всієї системи з нуля. [33]

Висновки до розділу 3

У третьому розділі реалізовано програмну систему інтелектуальної оптимізації маршрутів доставки пального у вигляді клієнтського SPA-вебдодатка на JavaScript. Система складається з шести функціональних модулів: генерації графа, A*, картографічного, генетичного алгоритму, симуляції та статистики.

Проведено тестування системи на реальних географічних даних міста Вінниці. Застосування генетичного алгоритму для оптимізації порядку відвідування АЗС забезпечило скорочення загальної відстані маршрутів на 30.7% порівняно з довільним порядком: вантажівка №1 — 32.8%, вантажівка №2 — 26.3%, вантажівка №3 — 33.1%. Час реакції системи на динамічні перешкоди не перевищив 50 мс.

Проведено порівняльний аналіз генетичного алгоритму з методами повного перебору, жадібним та мурашиним алгоритмами. Встановлено, що генетичний алгоритм забезпечує найкращий баланс між якістю рішення (85–97% від оптимального) та часом виконання ($O(T \times N \times k)$), перевищуючи жадібний алгоритм на 7–17% при порівнянних часових витратах. Визначено перспективи розвитку системи: впровадження VRPTW, CVRP та ML-прогнозування для проактивного планування маршрутів.

ВИСНОВКИ

У даній бакалаврській роботі розроблено та досліджено вебсистему інтелектуальної оптимізації маршрутів доставки пального, що реалізує комбінаторні алгоритми пошуку та оптимізації в задачах штучного інтелекту. В ході виконання роботи вирішено такі задачі:

1. Досліджено теоретичні основи комбінаторної оптимізації та її зв'язок із задачами штучного інтелекту. Встановлено, що задача маршрутизації транспортних засобів (VRP) є NP-важкою, що обумовлює застосування евристичних та метаевристичних методів. [12]
2. Побудовано математичну модель логістичної мережі у вигляді зваженого графа. Розроблено механізм динамічної зміни графа при виникненні перешкод — видалення ребер або збільшення їхніх ваг. [3]
3. Реалізовано алгоритм A* для пошуку оптимальних шляхів між точками маршруту. Алгоритм використовує манхеттенську відстань як допустиму евристичну функцію, що гарантує оптимальність знайденого шляху. Час виконання становить 1–3 мс для одного пошуку. [14]
4. Реалізовано генетичний алгоритм для оптимізації порядку відвідування АЗС з параметрами: N = 80, T = 260, pc = 0.8, pm = 0.05. Використано Order Crossover та swap mutation. [8, 7]
5. Розроблено модуль динамічних перешкод. Система успішно перераховує маршрути при виникненні перешкод за час менше 50 мс. [2]
6. Розроблено інтерактивний вебінтерфейс на базі Leaflet.js, що візуалізує маршрути, анімує рух вантажівок та відображає статистику в реальному часі. [32]
7. Проведено тестування системи. Результати показали, що застосування генетичного алгоритму дозволяє скоротити загальну відстань маршрутів в середньому на 30.7% порівняно з випадковим порядком відвідування АЗС. [7]

Розроблена система підтверджує ефективність гібридного підходу до вирішення задач логістичної оптимізації: генетичний алгоритм забезпечує

оптимізацію на рівні планування маршрутів, тоді як A* забезпечує точне знаходження шляхів на рівні дорожньої мережі. [14, 8]

Практична значущість роботи полягає в тому, що розроблена система може бути використана як основа для реальних систем управління автопарком та логістичними операціями [12]. Подальші напрямки розвитку включають: врахування вантажопідйомності вантажівок, часових вікон доставки (VRPTW), реальної дорожньої мережі через OpenStreetMap API, а також дослідження більш ефективних операторів генетичного алгоритму. [7]

Розроблена система показала, що поєднання точного перебору для малих задач і генетичного алгоритму для великих — практично ефективний підхід до логістичної оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Schrijver A. A Course in Combinatorial Optimization. — CWI, Amsterdam, 2006. — 228 p.

2. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. — 4th ed. — Pearson, 2020. — 1132 p.

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

4. Агаєв І. А. Методи та алгоритми комбінаторної оптимізації: навч. посіб. — К.: Політехніка, 2018. — 245 с.

5. Глибовець М. М., Олецький О. В. Штучний інтелект: підручник. — К.: Академія, 2002. — 366 с.

6. Стецюк П. І. Евристичні методи в комбінаторній оптимізації. — К.: Інститут кібернетики НАН України, 2017. — 178 с.

7. Goldberg D. E. Genetic Algorithms in Search, Optimization, and Machine Learning. — Addison-Wesley, 1989. — 432 p.

8. Holland J. H. Adaptation in Natural and Artificial Systems. — University of Michigan Press, 1975. — 183 p.

9. Романюк В. В., Гаврилюк М. М. Методи штучного інтелекту: навч. посіб. — НУВГП, 2019. — 214 с.

10. Кравченко В. В. Теорія алгоритмів: навч. посіб. — К.: КНУ, 2015. — 189 с.

11. Кисельов Г. Д. Алгоритми і структури даних: навч. посіб. — Харків: ХНУ, 2017. — 311 с.

12. Toth P., Vigo D. The Vehicle Routing Problem. — SIAM Monographs on Discrete Mathematics and Applications, 2002. — 367 p.

13. Dijkstra E. W. A Discipline of Programming. — Englewood Cliffs: Prentice-Hall, 1976. — 217 p.

14. 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.

15. Sturtevant N. R. Benchmarks for grid-based pathfinding // IEEE Transactions on Computational Intelligence and AI in Games. — 2012. — Vol. 4, No. 2. — P. 144–148.

16. Dechter R., Pearl J. Generalized best-first search strategies and the optimality of A* // Journal of the ACM. — 1985. — Vol. 32, No. 3. — P. 505–536.

17. Pearl J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. — Addison-Wesley, 1984. — 382 p.

18. Чабаненко Д. М. Ефективні алгоритми пошуку в задачах штучного інтелекту // Проблеми програмування. — 2021. — № 2. — С. 15–26.

19. Harabor D., Grastien A. Online graph pruning for pathfinding on grid maps // Proceedings of AAAI 2011. — P. 1114–1119.

20. De Jong K. A. An analysis of the behavior of a class of genetic adaptive systems: PhD thesis. — University of Michigan, 1975. — 251 p.

21. Larranaga P., Kuijpers C. M. H., Murga R. H. Genetic algorithms for the travelling salesman problem: A review // Artificial Intelligence Review. — 1999. — Vol. 13, No. 2. — P. 129–170.

22. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. — 3rd ed. — Springer, 1996. — 387 p.

23. Олійник А. О. Генетичні алгоритми в задачах оптимізації // Вісник ІПММ НАН України. — 2016. — Вип. 30. — С. 112–121.

24. Dantzig G. B., Ramser J. H. The Truck Dispatching Problem // Management Science. — 1959. — Vol. 6, No. 1. — P. 80–91.

25. Laporte G. The Vehicle Routing Problem: An overview of exact and approximate algorithms // European Journal of Operational Research. — 1992. — Vol. 59, No. 3. — P. 345–358.

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

27. Павлов О. А., Місюра Є. Б., Халус О. А. Задачі оптимального планування та розкладів. — К.: Техніка, 2012. — 264 с.

28. Крівцов В. С., Нечипоренко А. В. Методи вирішення задач маршрутизації транспортних засобів // Системи обробки інформації. — 2020. — № 3(162). — С. 44–52.

29. Линьов К. О., Довгаль В. М. Оптимізаційні задачі в транспортній логістиці // Вісник Вінницького політехнічного інституту. — 2023. — № 4. — С. 61–69.

30. OSRM Project — Open Source Routing Machine [Електронний ресурс]. — URL: https://project-osrm.org (дата звернення: 10.11.2024).

31. OpenStreetMap — вільна карта світу [Електронний ресурс]. — URL: https://www.openstreetmap.org (дата звернення: 05.11.2024).

32. Leaflet.js — офіційна документація [Електронний ресурс]. — URL: Documentation - Leaflet - a JavaScript library for interactive maps (дата звернення: 10.11.2024).

33. MDN Web Docs — JavaScript Reference [Електронний ресурс]. — URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (дата звернення: 15.11.2024).

34. Flanagan D. JavaScript: The Definitive Guide. — 7th ed. — O’Reilly, 2020. — 706 p.

35. Uchoa E., Pecin D., Pessoa A. et al. New benchmark instances for the capacitated vehicle routing problem // European Journal of Operational Research. — 2017. — Vol. 257, No. 3. — P. 845–858.

36. ДСТУ ISO/IEC 25010:2013. Системна та програмна інженерія. Вимоги та оцінювання якості систем і програмного забезпечення. — К.: Держстандарт України, 2015. — 34 с.

37. Potvin J.-Y., Bengio S. The vehicle routing problem with time windows — Part II: Genetic search // INFORMS Journal on Computing. — 1996. — Vol. 8, No. 2. — P. 165–172.

38. Nazari M., Oord A., Vinyals O., Takac P. Reinforcement learning for solving the vehicle routing problem // Advances in Neural Information Processing Systems. — 2018. — Vol. 31. — P. 9839–9849.

39. Kool W., van Hoof H., Welling M. Attention, learn to solve routing problems! // Proceedings of ICLR 2019. — URL: https://arxiv.org/abs/1803.08475 (дата звернення: 20.11.2024).

40. Мальований М. С., Гарматюк О. О. Інтелектуальні системи управління транспортними потоками // Автоматика. Автоматизація. Електротехнічні комплекси та системи. — 2022. — № 1(49). — С. 92–104.

ДОДАТОК А

Програмний код основних модулів системи

А.1. Модуль алгоритму A* (astar.js) — фрагмент реалізації

Лістинг А.1 — Функція знаходження найкоротшого шляху (astar.js)

```
function findPath(graph, startId, endId, heuristic) {
  const openList  = [];
  const closedSet = new Set();
  const gScore    = {};
  const cameFrom  = {};

  gScore[startId] = 0;
  openList.push({ id: startId, f: heuristic(startId, endId) });

  while (openList.length > 0) {
    openList.sort((a, b) => a.f - b.f);
    const current = openList.shift();

    if (current.id === endId)
      return reconstructPath(cameFrom, current.id);

    closedSet.add(current.id);
    for (const nb of graph[current.id].neighbors) {
      if (closedSet.has(nb.id)) continue;
      const tG = gScore[current.id] + nb.weight;
      if (tG < (gScore[nb.id] ?? Infinity)) {
        cameFrom[nb.id] = current.id;
        gScore[nb.id]   = tG;
        openList.push({ id: nb.id,
                        f:  tG + heuristic(nb.id, endId) });
      }
    }
  }
  return null; // шлях не знайдено
}
```

А.2. Модуль генетичного алгоритму (genetic.js) — фрагмент реалізації

Лістинг А.2 — Функція запуску генетичного алгоритму (genetic.js)

```
function runGeneticAlgorithm(stations, distMatrix, params) {
  let pop  = initPopulation(stations, params.popSize);
  let best = null;

  for (let g = 0; g < params.generations; g++) {
    pop = pop.map(ch => ({
      genes:   ch,
```

```
      fitness: 1 / calcDistance(ch, distMatrix)
    }));
    pop.sort((a, b) => b.fitness - a.fitness);

    if (!best || pop[0].fitness > best.fitness)
      best = pop[0];

    const elite  = pop.slice(0, params.eliteSize).map(p =>
p.genes);
    const newPop = [...elite];

    while (newPop.length < params.popSize) {
      const p1    = tournamentSelect(pop);
      const p2    = tournamentSelect(pop);
      let   child = orderCrossover(p1.genes, p2.genes);
      if (Math.random() < params.mutRate)
        child = swapMutate(child);
      newPop.push(child);
    }
    pop = newPop;
  }
  return best.genes;
}
```

А.3. Функція розподілу Вороного (logisticgenetic.js)

Лістинг А.3 — Функція призначення АЗС до нафтобаз методом Вороного

```
function assignToFactories(stations, factories) {
  return stations.map(st => {
    let   minDist    = Infinity;
    let   assignedId = null;
    for (const factory of factories) {
      const d = haversine(
        st.lat, st.lng,
        factory.lat, factory.lng
      );
      if (d < minDist) { minDist = d; assignedId = factory.id; }
    }
    return { ...st, factoryId: assignedId };
  });
}

function haversine(lat1, lon1, lat2, lon2) {
  const R  = 6371;  // радіус Землі, км
  const dLat = (lat2 - lat1) * Math.PI / 180;
  const dLon = (lon2 - lon1) * Math.PI / 180;
  const a  = Math.sin(dLat/2)**2
           + Math.cos(lat1*Math.PI/180)
           * Math.cos(lat2*Math.PI/180)
```

```
            * Math.sin(dLon/2)**2;
    return R * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
```