

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет систем управління літальних апаратів

Кафедра комп'ютерних наук та інформаційних технологій

**Пояснювальна записка
до кваліфікаційної роботи**

(тип кваліфікаційної роботи)

магістра

(освітній ступінь)

на тему: «Розробка веб-додатку для оптимізації маршрутів вантажоперевезень з застосуванням генетичних алгоритмів»

XAI.302.367.22O.122. 9652061 ПЗ

Виконав: здобувач б курсу групи № 367

(код та найменування)

Спеціальність 122 «Комп'ютерні науки»

(код і найменування напряму підготовки)

Освітня програма Комп'ютеризація обробки інформації та управління

(найменування)

Гринюк М.О.

(прізвище та ініціали здобувача (ки))

Керівник: Яшина О. С.

(прізвище та ініціали)

Рецензент: Литвинов А. Л.

(прізвище та ініціали)

Харків – 2022

**Міністерство освіти і науки України
Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»**

Факультет систем управління літальних апаратів
 Кафедра комп'ютерних наук та інформаційних технологій
 Рівень вищої освіти другий (магістерський)
 Спеціальність 122 «Комп'ютерні науки»
 Освітня програма Комп'ютерізація обробки інформації та управління

ЗАТВЕРДЖУЮ
Завідувач кафедри
Олег ФЕДОРОВИЧ
(підпис) (ім'я та прізвище)
«01» листопада 2022 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гринюку Максиму Олександровичу

(прізвище, ім'я та по батькові)

1. Тема кваліфікаційної роботи «Розробка веб-додатку для оптимізації маршрутів вантажоперевезень з застосуванням генетичних алгоритмів»
 керівник кваліфікаційної роботи Яшина Олена Сергіївна, к.т.н., доц.каф 302,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом Університету №_1602-уч. від «17» листопада 2022 року
2. Термін подання студентом кваліфікаційної роботи 15.12.2022
3. Вихідні дані до проекту (роботи) – математична модель транспортної задачі, технології ASP.Net Core, фреймворк Angular, мови програмування C#, JavaScript.
4. Зміст пояснівальної записки (перелік завдань, які потрібно розв'язати)
Аналіз транспортних задач та огляд існуючих рішень; формалізація та математичне моделювання предметної області; застосування генетичних алгоритмів до розв'язання задачі комівояжера; проектування програмного забезпечення оптимізації вантажоперевезень; програмна реалізація; виконання експериментальних досліджень; визначення собівартості і тривалості роботи при розробці програмного забезпечення.
5. Перелік графічного матеріалу: Постановка задачі; метод розв'язання задачі; особливості генетичного алгоритму для задач комівояжера; опис програмної реалізації; результати дослідження та чисельних експериментів; висновки;

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
з основної частини	Яшина О.С., доц. каф. 302		
з спеціального розділу	Яшина О.С., доц. каф. 302		
з економіки та управління	Малєєва Ю.А., доц. каф. 302		

Нормоконтроль _____ **Олександр ПРОХОРОВ** «15» грудня 2022 р.
 (підпис) (ініціали та прізвище)

7. Дата видачі завдання «31» жовтня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	1.09.2022 – 14.09.2022	
2	Опис математичної моделі	15.10.2022 – 17.10.2022	
3	Розробка алгоритмічної моделі та методики	18.10.2022 – 14.11.2022	
4	Тестування програмного продукту	15.11.2022 – 17.11.2022	
5	Розрахунок основних результатів	18.12.2022 – 30.12.2022	
6	Оформлення розрахунково–пояснювальної записки	01.09.2022 – 11.12.2022	
7	Розробка презентації	05.10.2022 – 05.11.2022	
8	Передзахист кваліфікаційної роботи магістра	19.12.2022	
9	Виправлення помилок та корегування зауважень	20.12.2022 – 22.12.2022	
10	Захист кваліфікаційної роботи магістра	24.12.2022	

Студент Гришко Максим ГРИНЮК
 (підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи Олена ЯШИНА
 (підпис) (прізвище та ініціали)

РЕФЕРАТ

Розробка веб-додатку для оптимізації маршрутів вантажоперевезень з застосуванням генетичних алгоритмів. – Гринюк Максим Олександрович, кваліфікаційна робота магістра, Харків, Національний аерокосмічний університет ім. М.Є. Жуковського «ХАІ», 86 сторінок, кількість рисунків 34, кількість таблиць 14, кількість джерел літератури 17.

КЛЮЧОВІ СЛОВА: гетеничний алгоритм, селекція, схрещування, хромосома, кросинговер, мутація, популяція, пристосованість, еволюційний алгоритм, комівояжер.

ОБ'ЄКТ ДОСЛІДЖЕННЯ: оптимізація маршрутів в транспортних задачах.

ПРЕДМЕТ ДОСЛІДЖЕННЯ: методи та засоби розв'язання задачі комівояжера з застосуванням генетичних алгоритмів.

МЕТА: оптимізація маршруту доставки товарів шляхом удосконалення математичного, алгоритмічного та програмного забезпечення розв'язання задачі комівояжера.

ЗАДАЧІ: 1) аналіз транспортних задач та огляд існуючих рішень; 2) формалізація та математичне моделювання предметної області; 3) застосування генетичних алгоритмів до розв'язання задачі комівояжера; 4) проектування програмного забезпечення оптимізації вантажоперевезень; 5) програмна реалізація; 6) виконання експериментальних досліджень; 7) визначення собівартості і тривалості роботи при розробці програмного забезпечення.

МЕТОДИ ДОСЛІДЖЕННЯ: математичне моделювання, завдання комівояжера, генетичні алгоритми, метод імітації відпалу.

ОТРИМАНІ РЕЗУЛЬТАТИ: розроблений веб-додаток для оптимізації маршрутів доставки вантажів.

ABSTRACT

Development of a web application for optimizing freight routes using genetic algorithms. – Maksym Hryniuk, Master's Thesis, Kharkiv, National Aerospace University "Kharkiv Aviation Institute", amount of pages 86, amount of figures 34, amount of tables 14, amount of sources of literature 17.

KEY WORDS: genetic algorithm, selection, crossing, chromosome, crossing-over, mutation, population, attachment, evolutionary algorithm, traveler.

OBJECT OF RESEARCH: optimization of routes in transport tasks.

SUBJECT OF RESEARCH: Methods and techniques for deriving the traveling traveling problem from the development of genetic algorithms.

PURPOSE OF RESEARCH: optimization of the route of delivery of goods by way of improving the mathematical, algorithmic and software security of developing the tasks of a traveler.

TASKS OF RESEARCH: 1) Analysis of transport tasks and an overview of the main decisions; 2) Formalization and mathematical modeling of the subject area;

3) development of genetic algorithms before the solution of the traveling traveling problem; 4) designing software for optimizing the transportation; 5) software implementation; 6) performing experimental research; 7) designation of compatibility and trivality of work in the development of software security.

METHODS OF RESEARCH: mathematical modeling, head of a traveler, genetic algorithms, the method of imitation of annealing.

RESULTS OF RESEARCH: developed web application for optimizing cargo delivery routes.

ЗМІСТ

ВСТУП.....	7
1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	9
1.1 Аналіз та проблеми	9
1.2 Способи розв'язання	10
1.3 Постановка задачі.....	16
1.4 Висновки за розділом.....	17
2 ФОРМАЛІЗАЦІЯ АБО КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ	18
2.1 Математична модель.....	18
2.2 Основні поняття.....	20
2.3 Схрещування та формування нового покоління	21
2.4 Висновки за розділом.....	22
3 АЛГОРИТМІЗАЦІЯ	23
3.1 Математична модель завдання комівояжера	23
3.2 Загальна характеристика завдання	23
3.3 Подання об'єктів	30
3.4 Особливості застосування	31
3.5 Метод гілок та кордонів.....	36
3.6 Висновки за розділом.....	40
4 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	41
4.1 Клієнтська частина.....	41
4.2 Серверна частина.....	46
4.3 Висновки за розділом.....	49
5 ОПИС РЕЖИМІВ РОБОТИ ТА ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ	50
5.1 Тестування генетичного алгоритму	50
5.2 Висновок за розділом.....	54
6 ЕКОНОМІЧНА ЧАСТИНА	55
6.1 Мета розділу	55
6.2 Визначення тривалості та розрахунок собівартості виконання проекту (перший варіант)	55
6.3 Визначення тривалості та розрахунок собівартості виконання проекту (другий варіант)	60
6.4 Висновки за розділом.....	64
ВИСНОВКИ	65
ПЕРЕЛІК ПОСИЛАНЬ	66
ДОДАТОК А ЛИСТИНГ ПРОГРАМИ ГЕНЕТИЧНИЙ АЛГОРИТМ НА С#.....	68
ДОДАТОК Б ГРАФІЧНІ МАТЕРІАЛИ	76

ВСТУП

Актуальність теми дослідження. Одним із основних бізнес-процесів компанії, що займається оптовою торгівлею продовольчими товарами, є постачання товарів замовникам. Як замовники виступають як дрібні торгові мережі, що не мають власних виробничих потужностей та транспортної служби доставки, так і різноманітні приватні магазини, що належать до классу малого бізнесу. Тому обліковий склад пунктів доставки щодня є унікальним і може містити до сотні пунктів у різних кінцях міста. У таких умовах визначити оптимальний маршрут проходження автомобіля доставки виявляється дуже важко, і навіть досвідчений водій, який добре знає вулично-дорожню мережу міста, може вирішити цю завдання далеко не найоптимальнішим чином. Тому актуальним завданням є розробка інструменту з урахуванням математичної моделі побудови оптимального маршруту. Причому, оскільки на маршруті прямування кожен пункт доставки має бути відвіданий лише один раз, дане завдання доречно звести до відомої задачі комівояжера.

Ефективність та якість функціонування транспортних комплексів значною мірою залежать від раціональної координації роботи різних видів транспорту, оптимального перерозподілу між ними обсягів перевезень та формування на цій основі необхідних управлінських рішень.

Метою дослідження оптимізація маршруту доставки товарів шляхом удосконалення математичного, алгоритмічного та програмного забезпечення розв'язання задачі комівояжера.

Основні завдання дослідження:

- 1) аналіз транспортних задач та огляд існуючих рішень;
- 2) формалізація та математичне моделювання предметної області;
- 3) застосування генетичних алгоритмів до розв'язання задачі комівояжера;
- 4) проектування програмного забезпечення оптимізації вантажоперевезень;
- 5) програмна реалізація;
- 6) виконання експериментальних досліджень;
- 7) визначення собівартості і тривалості роботи при розробці програмного забезпечення.

Об'єктом дослідження є процеси оптимізації маршрутів в транспортних задачах.

Предметом дослідження є методи та засоби розв'язання задачі комівояжера з застосуванням генетичних алгоритмів.

Практичне значення отриманих результатів проведеного дослідження полягає в автоматизації управлінських рішень, зменшення витрачення ресурсів для логістичних завдань.

Структура і обсяг кваліфікаційної роботи магістра. Кваліфікаційна робота магістра складається з вступу, 6 розділів, висновків і додатків. Повний обсяг кваліфікаційної роботи магістра становить 86 сторінок, в тому числі: 34 малюнка; 14 таблиць по тексту; 3 додатка; 17 джерел літератури.

1 АНАЛІЗ ПРОБЛЕМИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз та проблеми

Суть завдання комівояжера полягає в наступному. Є N пунктів спрямування, відстані між якими відомі. Мандрівник (або комівояжер) повинен відвідати кожен з базового набору пунктів спрямування по одному разу та повернутися до вихідної точки. Відома вартість (час) переміщення з одного пунктів спрямування до іншого. Необхідно скласти маршрут, щоб сума витрачених коштів (часу) була мінімальною. Цілком очевидно, що завдання може бути вирішene перебором усіх варіантів об'їзду та вибором оптимального. Але проблема в тому, що кількість можливих маршрутів дуже швидко зростає зі зростанням n (она дорівнює $n!$ кількості способів упорядкування пунктів). Наприклад для 100 пунктів кількість варіантів буде представлятися 158-значним числом. Наразі не існує алгоритму рішення, що має статичну складність (тобто вимагає порядку n^a операцій для деякого a , будь-який алгоритм буде гіршим. Все це робить завдання комівояжера безнадійною для послідовного виконання операцій, якщо хоч скільки-небудь велике. У такому разі слід відмовитися від спроб відшукати точне вирішення завдання комівояжера і зосередитися на пошуку наближеного – нехай не оптимального, але хоча б близького до нього. З огляду на велику практичну важливість завдання корисними будуть і наближені рішення. Тому найбільш оптимальні методи для використання будуть – моделювання тих або інших біологічних та природних процесів які вирішують необхідну задачу.

Пошуковий простір для завдання комівояжера – безліч N пунктів. Будь-яка комбінація з N пунктів, які не повторюються, є рішенням. Оптимальне рішення – така комбінація, вартість якої мінімальна. Існують суворі обмеження на послідовність, і кількість пунктів спрямування може бути дуже великим. Завдання комівояжера було віднесене до NP-повним завданням, т. е. повним перебором за поліноміальний час. Алгоритмічна складність симетричної задачі комівояжера з n пунктів спрямування становить $\frac{(n-1)!}{2}$. Відповідно, вже для 30 пунктів спрямування пошук оптимального шляху є складним завданням, практично нерозв'язним методом повного перебору і спонукає розвиток різних нових методів, у тому числі нейромереж та генетичних алгоритмів. Кожен варіант рішення – це числовий рядок, де на j -му місці стоїть номер j -го по порядку обходу міста. Таким чином, у цій задачі N параметрів (N – число пунктів спрямування), причому не всі комбінації значень допустимі.

Задача має вигляд наступним: є кілька автотранспорту, один склад (депо) та кілька клієнтів. Для кожного транспортного засобу потрібно скласти маршрут,

протягом якого транспортний засіб відвідує клієнтів (наприклад, з метою доставки будь-якого вантажу). На маршрут кожного транспортного засобу накладається ряд обмежень. Основні обмеження представлені формулами:

$$\sum_{k \in V} \sum_{j \in N} X_{ij}^k = 1, \forall i \in C, \quad (1.1)$$

$$\sum_{i \in C} d_i \sum_{j \in N} X_{ij}^k \leq q, \forall k \in V \quad (1.2)$$

$$a_i \leq S_i^k \leq b_i, \forall i \in N, \forall k \in V \quad (1.3)$$

Обмеження (1.1) вважає, що кожен клієнт обслуговується лише одним транспортним засобом та лише один раз. Змінні X_{ij}^k приймають значення $\{0, 1\}$, де 1 означає, що автомобіль рухається від вершини i до вершини j , 0 – зворотне. Верхнім індексом k позначається відповідний кур'єр ($k \in V$ де V – кількість ідентичних автомобілів вантажопідйомністю q).

Обмеження (1.2) визначає, що транспортний засіб не може обслугити більше клієнтів, ніж дозволяє його вантажопідйомність d_i , $i \in C$ – попит відповідного клієнта, C – безліч клієнтів. Обмеження (1.3) – це обмеження за часом; прибуття автомобіля до клієнта має бути в межах тимчасового вікна, де S_i^k – час прибуття відповідного автомобіля до певного клієнта, а $[a_i, b_i]$ – проміжок часу, так зване тимчасове вікно, протягом якого має бути обслужений клієнт. Для даної задачі формулюються наступні цілі (цільові функції): первинна мета – мінімізувати загальну кількість транспортних засобів, необхідні обслуговування всіх клієнтів; вторинні – мінімізувати загальний час обслуговування всіх клієнтів та загальну відстань, пройдену всіма транспортними засобами. Точні методи, засновані на методах спрямованого перебору, не дозволяють ефективно вирішувати ці завдання великої розмірності. Евристичні та мета-евристичні методи, яким відносяться генетичні алгоритми, дають у цьому випадку більш прийнятні результати.

1.2 Способи розв’язання

Метод градієнтного спуску. Один із поширених способів рішення задачі комівояжера заснований на методі градієнтного спуску. *Відношення близькості* – це спосіб визначити для двох елементів множини, чи є вони близькими (у якомусь сенсі). Для числових множин, для множин точок на площині чи просторі близькими можна вважати два числа (две точки), відстань між якими не перевищує деякого маленького числа ε . Для безлічі S_n близькими зручно вважати дві перестановки, що відрізняються на одну транспозицію, тобто перестановка двох елементів множини, що виходять

один з одного. Наприклад, перестановки $(2,4,1,3)$ і $(2,3,1,4)$ близькі в цьому сенсі, оскільки відрізняються перестановою елементів з номерами 2 і 4. Можна визначити й суворіше ставлення близькості, у якому близькі перестановки відрізняються сусіднію транспозицією, коли рокіровка зачіпає елементи множини із сусідніми номерами. Тоді вказані вище перестановки близькими не будуть, але близькими виявляться $(2,4,1,3)$ і $(2, 4, 3, 1)$.

Суть методу градієнтного спуску відбита у його назві і полягає в наступному. Будується послідовність $\{x_1, x_2, x_3, \dots\} \subset X$, у якій початковий елемент x_0 вибирається довільно (можливо, випадковим чином), а кожен наступний є одним із сусідів попереднього, причому саме тим із сусідів, для якого значення функції U буде найменшим. Побудова послідовності завершується тоді, коли послідовність значень цільової функції $\{U(x_0), U(x_1), U(x_2), U(x_3), \dots\}$ перестане бути монотонно спадаючою. Останній елемент побудованої послідовності називають точкою локального мінімуму. Це така точка, у яких значення U суворо менше, ніж у всіх сусідніх із нею. У слові «локальний» укладено головний недолік описаного методу. Локальних мінімумів у функції U може бути багато, і кожному з них відповідає, як правило, своє локально мінімальне значення цільової функції.

При цьому спочатку вибираються деякі випадкові значення параметрів, а потім ці значення поступово змінюють, досягаючи найбільшої швидкості зростання цільової функції. Досягши локального максимуму, такий алгоритм зупиняється, тому для пошуку глобального оптимуму будуть потрібні додаткові зусилля. Градієнтні методи працюють дуже швидко, але не гарантують оптимальність знайденого рішення. Вони ідеальні для застосування у так званих унімодальних задачах, де цільова функція має єдиний локальний максимум. (Він же глобальний). Легко бачити, що задача комівояжера не є унімодальною. Типове практичне завдання, як правило, мультимодальне та багатовимірне. Для таких завдань немає жодного універсального методу, який дозволяв би досить швидко знайти точне рішення. Однак, комбінуючи перебірний та градієнтний методи, можна сподіватися отримати хоча б наближене рішення, точність якого зростатиме при збільшенні часу розрахунку.

Метод імітації відпалу. Ще один із наближених методів вирішення таких завдань оптимізації – метод імітації відпалу. *Відпал* – вже згадуваний процес поступового остигання речовини, у якому молекули і натомість теплового руху, що все сповільнюється, збираються в найбільш енергетично вигідні конфігурації. Термін «відпал» прийшов із металургії. Справа в тому, що метал у більш енергетично вигідному стані одночасно твердіший і міцніший: потрібна більша зовнішня дія, велика механічна робота над шматком металу, щоб порушити вигідну конфігурацію молекул, «підняти» цю конфігурацію над самим дном енергетичної ями. Метод

імітації відпалу є модифікацією імовірнісного методу градієнтного спуску. Відмінність полягає в поведінці алгоритму, коли $U(x) \leq U(\tilde{x})$, де x – черговий елемент послідовності, а \tilde{x} - його сусід, выбраний навмання. Імовірнісний метод градієнтного спуску відкидав такого сусіда безумовно, а метод імітації відпалу допускає додавання такого «поганого» сусіда до послідовності, щоправда, з певною ймовірністю p , яка залежить від того, наскільки поганий сусід погіршив цільову функцію. Візьмемо різницю $\Delta U = U(\tilde{x}) - U(x)$ і представимо $p = e^{-\frac{\Delta U}{\theta}}$, де $e = 2.718281828459045..$, а θ – деяке позитивне число, яке називається температурою. На рис. 1 «Імовірність мутації для методу імітації відпалу» показано залежність ймовірності мутації від величини ΔU при різних значеннях температури θ . Високим температурам відповідають графіки, колір яких ближче до червоного, низьким – до синього. Як і належить, значення ймовірності укладено у відрізку. При негативних ΔU ймовірність дорівнює 1, що відповідає випадку «хорошої» мутації. Приклад наведений на рисунку 1.1.

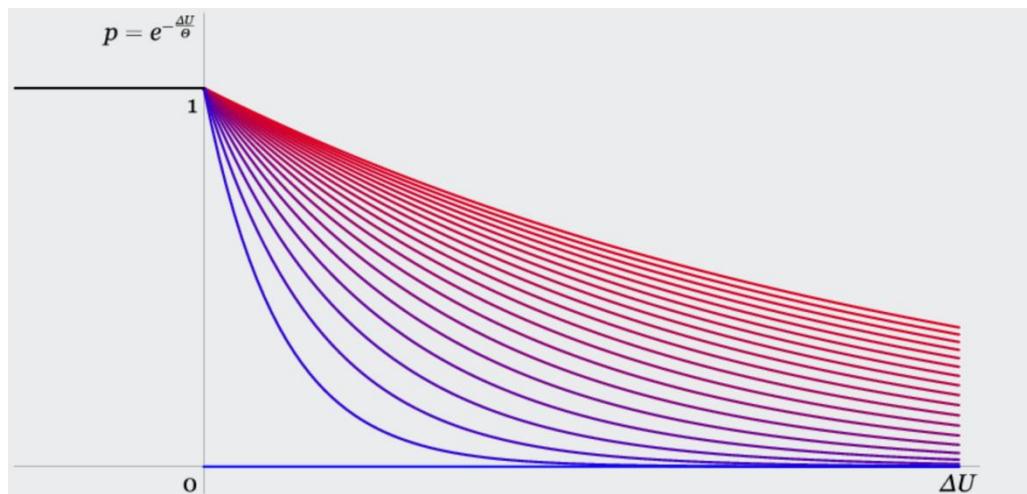


Рисунок 1.1 – Графік імовірності мутації для методу імітації відпалу

Жадібний алгоритм (англ. Greedy algorithm) – алгоритм, який полягає у прийнятті локально оптимальних рішень на кожному етапі, припускаючи, що кінцеве рішення також виявиться оптимальним. Відомо, що й структура завдання задається матроїдом, тоді застосування жадібного алгоритму видасть глобальний оптимум. При виборі чергового міста бере найближче не відвідане раніше місто. На жаль, якщо спочатку шлях виходить коротким, то в кінці часто виникає дуже великий приріст його довжини. У жадібному алгоритмі важливо яке місто вибирається як стартового. Щоб покращити результат, будемо по черзі робити стартовим кожне місто, обираючи той варіант, який дає найкоротший шлях. На початку наступної ітерації допоміжний масив заповнюється номерами міст: $[0,1,2,3,\dots,N-1]$ і перше місце

ставиться місто $j=0,1,2,\dots,N-1$. І тому він змінюється місцями з нульовим елементом масиву. Потім шукається найближче до нього місто і ставиться на перше місце (знов за допомогою перестановки елементів масиву). Ці дії продовжуються, доки сформується повний шлях.

Жадібний алгоритм (з перебором стартового міста) має кубічний час роботи $T \sim n^3$. Приклад наведений на рисунку 1.2.

```

Salesman.prototype.greedy = function ()
{
    this.minLen = Infinity;
    for(var j=0; j < this.N; j++){
        for(var i=0; i < this.N; i++)
            this.way[i] = i; // все города
        Salesman.swap(this.way, 0, j); // j-ый город ставим первым

        var last = j, lf = 1, d = 0; // last - последний посещённый город
        while(lf < this.N){ // пока не перебрали оставшиеся города
            var minD = this.dists[last][this.way[lf]]; // ищем ближайший к last город
            for(var i=lf+1; i < this.N; i++) // ищем ближайший к last город
                if(this.dists[last][this.way[i]] < minD){
                    minD = this.dists[last][this.way[i]];
                    Salesman.swap(this.way, lf, i); // ближайший переносим в начало
                }
            d += minD; // суммируем общую длину пути
            last = this.way[lf++]; // берём "первый" как ближайший
        }
        d += this.dists[last][j]; // финальный отрезок
        if(this.minLen > d){ // если длина пути лучшая,
            this.minLen = d; // запоминаем её и путь:
            this.minWay = Salesman.copy(this.minWay, this.way);
        }
    }
}

```

Рисунок 1.2 – Фрагмент коду жадібного алгоритму

Генетичний алгоритм є саме такий комбінований метод. Механізми скрещування та мутації в певному сенсі реалізують перебірну частину методу, а вибір кращих рішень – градієнтний спуск. По суті розглядається безперервне багато параметричне завдання безперервної оптимізації, де $f(x)$ – максимізована (цільова) скалярна багато параметрична функція, яка може бути не визначена поза допустимою областю, а всередині допустимої області мати кілька глобальних екстремумів; D – область пошуку; x – вектор, що кодує розв'язання задачі:

$$\max f(x), x \in D, D = \{x = (x_1, x_2, \dots, x_n) | x_i \in [a_i, b_i], i = \overline{1, N}\} \quad (1.4)$$

Передбачається, що про функцію $f(x)$ відомо лише те, що вона визначена в будь-якій точці області D . Ніяка додаткова інформація про характер функції та її властивості не враховується у процесі пошуку. Виходячи з припущення про можливу багатоекстремальність $f(x)$ оптимальне рішення може бути не єдиним (1.1).

Перевагою генетичного алгоритму полягає у знаходженні множини субоптимальних рішень за відносно короткий час. Під розв'язанням задачі розуміємо маршрут, який формально представляє вектор $x = (x_1, x_2, \dots, x_n)$. Оптимальним маршрутом вважатимемо вектор x^* , при якому цільова функція $\int(x)$ набуває максимального значення. Виходячи з припущення про можливу багатоекстремальність $\int(x)$, оптимальне рішення може бути не єдиним.

Маршрутом називатимемо послідовність неповторних міст (котрий має свій порядковий номер) Основна перевага порядкового уявлення в тому, що нього застосовний практично будь-який оператор кросовера. Будь-які два маршрути в порядковому виставі, обрізані на будь-якій позиції і склеєні разом, породять два нащадки, кожен з яких буде правильним маршрутом.

Класична постановка завдання на пошук оптимального маршруту полягає в мінімізації цільової функції, побудованої за критерієм витрачених коштів або часу на основі знання вартості/часу переміщення з одного пункту до іншого. У нашому випадку як основний критерій доцільно використовувати розрахунковий час проїзду з одного пунктів спрямування в інший. При цьому розрахунок здійснюється на підставі даних про стан вулично-дорожньої мережі, у т. ч. погодних умов та інтенсивності транспортних потоків. Вираз для розрахунку цільової функції:

$$F^* = \sum_{i=1}^{N-1} T_{ri}, F^* \rightarrow \min, \quad (1.5)$$

де T_{ri} – розрахунковий час проїзду між i -м та $(i+1)$ -м пунктів спрямування. N – число пунктів спрямування на маршруті спрямування. Оскільки в задачі розвезення товарів існує лише один початковий/кінцевий пункт, підвищення надійності даного генетичного апарату доцільно кожному поколінню відстежувати достовірність початкового та кінцевого пунктів спрямування та при пошкодженні проводити їх відновлення. Для реалізації ГА використовується класична схема з наступними параметрами (рисунок 1.3):

- 1) псевдовипадкова генерація початкової популяції (з витримкою обмежень, що накладаються заданими параметрами);
- 2) природний відбір методом турнірного розіграшу з високою розмірністю турнірної групи;
- 3) парний оператор кросинговеру;
- 4) використання аутбридингу для утворення батьківських пар;
- 5) випадковий вибір одноточкових та двоточкового кросинговеру та мінливості;

- 6) ремонт зіпсованих хромосом (що не відповідають заданим параметрам) за допомогою спрямованих мутацій;
- 7) формування нового покоління шляхом елітного відбору.

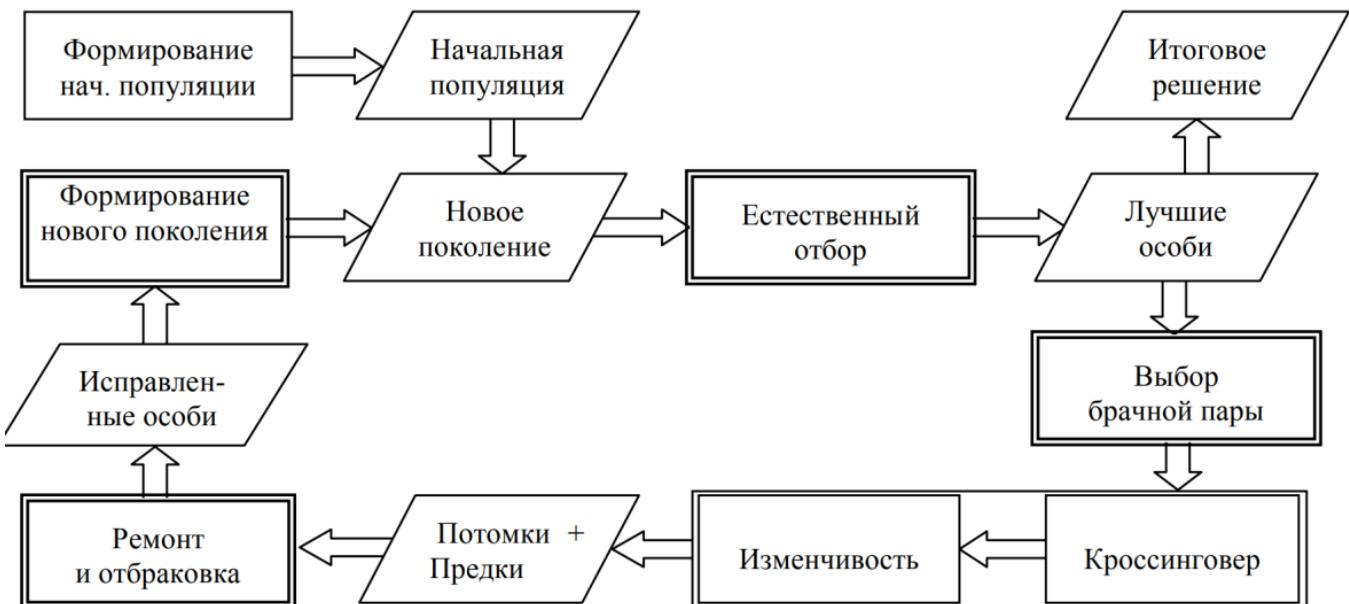


Рисунок. 1.3 – Схема генетичного алгоритму

Робота програми представлена сторінкою, що дозволяє імітувати надходження вихідних даних за розрахунковим часом проїзду у вигляді матриці суміжності та провести пошук набору з десяти оптимізованих рішень (маршрутів прямування) для автомобіля. Вибір остаточного варіанта маршруту надається оператору. Остаточне рішення у вигляді послідовності пунктів спрямування направляється водієві. Приклад матриці суміжності для 14 пунктів спрямування, що належать до вулично-дорожньої мережі, включаючи вихідний та кінцевий пункт, представлений у таблиці на рисунку 1.4, де пункт спрямування №0 – це база транспортного засобу, що здійснює розвезення товарів до пунктів роздрібного продажу. Слід зазначити, що матриця не є симетричною, тому що в силу особливостей вулично-дорожньої мережі не завжди можливий проїзд одним і тим самим шляхом від одного пустка спрямування до іншого.

Початкові параметри генетичного алгоритму:

- 1) потужність популяції, розрахована за формулою (2) – 100 особин;
- 2) число поколінь прийнято рівним 20;
- 3) ймовірність кросинговеру – 0,9;
- 4) ймовірність операторів мінливості – 0.05.

№ ПС	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	5	14	14	12	16	14	14	19	9	12	6	7	8	12
1	5	0	8	8	7	11	9	9	18	4	8	2	7	8	11
2	13	8	0	0	6	8	6	7	19	4	15	11	16	17	19
3	13	8	0	0	6	8	6	7	19	4	15	11	16	17	19
4	12	7	6	6	0	6	4	9	20	3	15	9	14	15	17
5	14	9	6	6	5	0	1	12	20	4	11	11	14	14	17
6	12	7	5	5	3	6	0	12	23	3	12	10	12	13	15
7	15	10	8	8	10	10	12	0	14	12	16	19	21	22	24
8	19	20	19	19	20	17	19	14	0	22	12	21	16	17	19
9	9	4	5	5	3	6	5	12	22	0	12	6	11	12	15
10	12	9	16	16	15	10	12	17	12	13	0	11	9	10	12
11	5	4	12	12	11	15	13	20	18	8	11	0	6	7	11
12	5	6	15	15	13	13	15	20	17	10	12	6	0	1	9
13	6	7	16	16	14	13	15	20	17	11	9	7	1	0	9
14	11	12	20	20	19	18	19	24	20	16	12	12	7	6	0

Рисунок 1.4 – Матриця суміжності

1.3 Постановка задачі

Метою роботи є оптимізація маршруту доставки товарів шляхом удосконалення математичного, алгоритмічного та програмного забезпечення розв'язання задачі комівояжера. Для досягнення поставленої мети необхідно виконати завдання:

- 1) аналіз транспортних задач та огляд існуючих рішень;
- 2) формалізацію та математичне моделювання предметної області;
- 3) застосування генетичних алгоритмів до розв'язання задачі комівояжера;
- 4) проектування програмного забезпечення оптимізації вантажоперевезень;
- 5) програмна реалізація;
- 6) виконання експериментальних досліджень;
- 7) визначення собівартості і тривалості роботи при розробці програмного забезпечення.

Розв'язання транспортного завдання за допомогою генетичних алгоритмів можна описати так: є автотранспорт та кур'єр, один склад депо та кілька точок доставки. Для кожної точки необхідно скласти маршрут, протягом якого транспорт відвідує ряд точки доставки (клієнтів). На маршрут кожного транспортного засобу накладається ряд обмежень:

- 1) кожного клієнта кур'єр може відвідати лише один раз;
- 2) кур'єр повинен повернутися в депо після останнього клієнта.

Очікуваний результат: програмне забезпечення, яке реалізує генетичний алгоритм та видає найкоротший маршрут. Це дозволить мінімізувати загальний час

обслуговування всіх клієнтів та загальну відстань, пройдену по всім пунктам доставки. Точні методи, засновані на методах спрямованого перебору, не дозволяють ефективно вирішувати ці завдання великої розмірності. Евристичні та мета евристичні методи, яким відносяться генетичні алгоритми, дають у цьому випадку більш прийнятні результати.

1.4 Висновки за розділом

Дана загальна характеристика завдання комівояжера, розглянули проблеми існуючих рішень, дали обґрутований вибір, визначили постановку задач.

2 ФОРМАЛІЗАЦІЯ АБО КОНЦЕПТУАЛЬНЕ ПРОЕКТУВАННЯ

2.1 Математична модель

Змоделюємо загальну задачу оптимізації с використання генетичного алгоритму наступним чином:

$$\max \{f(S_k) | S_k \in \{0,1\}^n; k = 1, \dots, n\}, \quad (2.1)$$

де n – розмір популяції. Первинна популяція є формований випадково простір потенційних рішень існуючої задачі. Генетичний алгоритм за допомогою операторів буде перетворювати цю популяцію до тих пір, поки не буде, досягнуто критерію зупинки, або буде досягнуто максимальної кількості поколінь. Індивіди кожного покоління відбираються з їх рівня пристосованості. Функцією пристосованості є цільова функція поставленого завдання. Потім гени індивідів попарно схрещуються та виходять індивіди нового покоління, далі нові індивіди піддаються мутації. Отримане таким чином рішення додається до популяції і відкидається значення, для якого цільова функція має найменший показник.

Представимо роботу генетичного алгоритму наступними кроками:

1. Вибираємо вихідну популяцію

$$S_k(0) = \{S_{k1}, S_{k2}, \dots, S_{kn}\} \quad (2.2)$$

де N – Довжина ланцюга. Вважатимемо, що:

$$f^* = \max\{f(\varphi_k) | S_k \in S_k(t)\}, t = 0 \quad (2.3)$$

2. До досягнення критерію зупинки необхідно виконувати наступні шляхи:

- 2.1) обираємо батьків S_{k1}, S_{k2} із загальної популяції $S_k(t)$;
- 2.2) шляхом схрещування отримуємо особину S_k ;
- 2.3) за допомогою S_{k1}, S_{k2} , модифікуємо S_k , тобто, робимо мутацію.

3. Перевіряємо, якщо

$$f^* < f(S_k) \text{ то } f^* := f(S_k) \quad (2.4)$$

то

$$t := t + 1 \quad (2.5)$$

На першому етапі потрібно коректно сформувати вихідну популяцію. Індивіди, які складаються з певного набору хромосом, представлених у вигляді бітових рядків або векторів.

4) Далі генетичний алгоритм декодує значення та отримує безліч значень, що задовольняють субоптимальному значенню поставленого завдання. Отже, генетичний алгоритм, власне, є множино-вероятностної моделлю. Що дозволяє генерувати безліч рішень задачі, що лежать у певній околиці від істинного рішення. Цей факт дозволяє працювати у тих галузях, де точка екстремуму завдання може бути чітко задана.

Розглянемо обмін хромосомами між батьківськими індивідами популяції з метою визначення значення пристосованості. Для цього розглянемо двох батьків та їх набори хромосом, так:

$$S_1 = (S_{11}, S_{12}, \dots, S_{1n}) \text{ i } S_2 = (S_{21}, S_{22}, \dots, S_{2n}) \quad (2.6)$$

Крапка розриву хромосом задана випадково. Потім батьківські індивіди діляться щодо випадково заданої точки обмінюються хромосомами. В результаті виникають два нащадки з наступними наборами хромосом:

$$(S_{11}, \dots, S_{1m}, S_{2m+1}, \dots, S_{2N}) \text{ i } (S_{21}, \dots, S_{2m}, S_{1m+1}, \dots, S_{1N}) \quad (2.7)$$

Тобто ділянки після точки розриву схрещуються між хромосомами батьків. Далі відбувається мутація індивідів нового покоління. Мутація здійснюється наступним чином: у кожному рядку один із геном змінюється на протилежний. Перший із створених нащадків зберігатиме символи хромосом батьків S_1 або S_2 , довільні для будь-якої позиції символів. Другий нащадок отримає символи різниці. Наприклад, нехай нащадки мають такий вигляд:

$$S_1 = (S_{11}, S_{12}, S_{13}, S_{14} \dots S_{2n}) \text{ i } S_2 = (S_{21}, S_{22}, S_{23}, S_{24}, \dots S_{1n}) \quad (2.8)$$

Таким чином, генетичні алгоритми мають здатні виробляти паралельний пошук генів у всіх комбінаціях. Відповідно час пошуку прямо пропорційно до обсягу популяції.

2.2 Основні поняття

Процес починається з набору особин, що називається населенням. Кожна особина – це вирішення проблеми, що була поставлена. Особина характеризується набором параметрів (змінних), які називають генами. Гени об'єднані в один рядок і формують хромосому – розв'язання задачі. У генетичному алгоритмі набір генів особини представлений у вигляді бінарного рядка. Закодована комбінація генів називається хромосомою.

Функція сили – функція сили визначає, наскільки сильна окрема особина. Сила визначається як здатність особи конкурувати з іншими особами за заданою метрикою. Функція надає кожній особині рівень сили. Імовірність те, що особина буде обрано добутку наступної популяції, ґрунтуючись лише на рівні сили особини.

Відбір – ідея відбору полягає в тому, щоб відібрати найсильніших особин та передати їх гени наступному поколінню особин. Н пар особин (батьки) відбирається виходячи з їх сили.

Схрещування – це основна частина генетичного алгоритму. Для кожної пари батьків. Випадково вибирається крапка у бінарному рядку хромосоми, до якої особини обмінюються генами. Після цього модифіковані особини називаються потомством.

Точка схрещування – нащадок створюється через процес обміну генами батьків до випадково заданої позиції у рядку. Після обміну генами між батьками потомство додається до нової популяції.

Мутація – якась частина генів буде малоймовірною. Щоб підтримувати різноманітність популяції, окремо прописується процес мутації нових особин.

Селекція – (відбір) необхідна, щоб вибрati більш пристосованих особин для схрещування. Існує безліч варіантів селекції, опишемо найвідоміші з них.

Рулеткова селекція – у даному варіанті селекції ймовірність i -ї особини взяти участь у схрещуванні p_i пропорційна значенню її пристосованості f_i і дорівнює $p_i = \frac{f_i}{\sum_j f_j}$. Процес відбору особин для схрещування нагадує гру в рулетку. Рулетковий круг ділиться на сектори, причому площа i -го сектора пропорційна значенню p_i . Після цього n разів «обертається» рулетка, де n – розмір популяції, і сектору, у якому зупиняється рулетка, визначається особина, вибрана для схрещування.

Селекція усіченням – При відборі усічення після обчислення значень пристосованості для схрещування вибираються ln кращих особин, де l – "поріг відсікання", $0 < l < 1$, n – розмір популяції. Чим менше значення k , тим більше тиск селекції, тобто менше шанси на виживання у погано пристосованих особин. Як правило, вибирають l в інтервалі від 0.3 до 0.7.

Алгоритм завершує роботу, коли населення зійшлася, тобто не виробляє потомство, яке значно відрізняється від попереднього покоління. Коли алгоритм зійшовся, виході виходить набір оптимальних рішень заданої проблеми.

2.3 Схрещування та формування нового покоління

Відібрані внаслідок селекції особини (звані батьківськими) схрещуються та дають потомство. Хромосоми нащадків формуються в у процесі обміну генетичною інформацією (із застосуванням оператора кросовера) між батьківськими особами. Створені в такий спосіб нащадки становлять популяцію наступного покоління. Нижче будуть описані основні схрещування для цілісного та речового кодування. Будемо розглядати випадок, коли з безлічі батьківських особин випадково вибираються 2 особини і схрещуються з ймовірністю РС, в результаті чого створюються 2 нащадки. Цей процес повторюється до того часу, доки створено n нащадків. Імовірність схрещування РС є одним з ключових параметрів генетичного алгоритму та в більшості випадків її значення знаходиться у діапазоні від 0,6 до 1.

Руйнівна здатність кросовера – оператори кросовера характеризуються здатністю до руйнування батьківських хромосом. Кросовер для цілісного кодування вважається більш руйнівним, якщо в результаті його застосування відстань по Хеммінгу між хромосомами нащадків і хромосомами батьків, що вийшли, велика. Інакше кажучи, здатність цілочисленного кросовера до руйнування залежить від цього, наскільки сильно він «перемішує» (рекомбінує) вміст батьківських хромосом. Так, 1-точковий кросовер вважається слаборуйнівним, а однорідний кросовер у більшості випадків є сильноруйнівним оператором. Відповідно, 2-точковий кросовер по руйнівної здатності займає проміжну позицію по відношенню до 1-точкового та однорідного кросоверів. Для кросовера для речовинного кодування здатність до руйнування визначається тим, наскільки велика відстань у просторі пошуку між точками, що відповідають хромосомам батьків та нащадків. Таким чином, руйнівний ефект 2-точкового кросовера залежить від вмісту батьківських хромосом. Руйнівна здатність арифметичного кросовера залежить від значення параметра λ , наприклад, при $\lambda \rightarrow 1$ та $\lambda \rightarrow 0$, здатність до руйнування буде низькою. Для BLX- α кросовера руйнівна здатність залежить як від значення α , так і від різниці значень відповідних генів батьківських особин. Зазначимо, що одночасно зі здатністю до руйнування говорять також про здатність до створення кросовером нових особин. Тим самим підкреслюється, що, руйнуючи батьківські хромосоми особин, кросовер може створити абсолютно нові хромосоми, які раніше не зустрічалися в процесі еволюційного пошуку.

Формування нового покоління – як згадувалося вище, внаслідок схрещування створюються нащадки, які формують популяцію наступного покоління. Зазначимо, що оновлена таким чином популяція не обов'язково повинна включати самих лише особин-нащадків. Якщо частка оновлюваних особин дорівнює T , то нове покоління потрапляє T_n Tn нащадків, n – розмір популяції, а $(1 - T)n$ особи у новій популяції є найбільш пристосованими батьківськими особинами (так звані елітні особини). Параметр T називають розрив поколінь. Використання елітних особин дозволяє збільшити швидкість збіжності генетичного алгоритму

Мутація – оператор мутації використовується для внесення випадкових змін хромосоми особин. Це дозволяє «вибиратися» з локальних екстремумів. і тим самим, ефективніше досліджувати простір пошуку. Так само як і для оператора кросовера існує ймовірність застосування мутації P_m . Розглянемо різні оператори мутації залежно від способу уявлення генетичної інформації

2.4 Висновки за розділом

Дані основні поняття генетичного алгоритму та створена математична та концептуальна моделі генетичного алгоритму в задачі комівояжера.

3 АЛГОРИТМІЗАЦІЯ

3.1 Математична модель завдання комівояжера

Сформульована задача – завдання ціличислове. Розглядається n міст, пов'язаних дорожньою мережею. Нехай $x_{ij} = 1$, якщо мандрівник переїжджає з i -ого міста до j -ого і $x_{ij} = 0$, якщо j -е місто не відвідується. Умовно запровадимо $(n+1)$ -е місто, поєднане з 1-м містом, тобто. відстані від $(n+1)$ -го міста до будь-якого іншого, відмінного від первого, дорівнюють відстаням від первого міста. При цьому, якщо з первого міста можна лише вийти, то в $(n+1)$ місто можна лише прийти. Введемо додаткові цілі змінні, що рівні номеру відвідування цього міста на шляху. $u_l=0$, $u_{n+1}=n$. Для того, щоб уникнути замкнутих шляхів, вийти з первого міста і повернутися до $(n+1)$ введемо додаткові обмеження, що зв'язують змінні x_{ij} та змінні u_i (u_i цілі негативні числа).

$$\min \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (3.1)$$

$$\sum_{i=1}^n x_{ij} = 1, \quad 1 \leq j \leq n, \quad (3.2)$$

$$\sum_{j=1}^n x_{ij} = 1, \quad 1 \leq i \leq n, \quad (3.3)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \geq 1, \quad S \neq \emptyset, S \subseteq \{1, \dots, n\}, \quad (3.4)$$

$$x_{ij} \in \{0,1\}, \quad 1 \leq i, j \leq n \quad (3.5)$$

Верхня подвійна сума – цільова функція завдання, на яку найменше значення слід шукати. Співвідношення нижче-обмеження, що накладаються на змінні. Одноразові суми – вказують на вимогу для заняття одиничними елементами плану-рішення X єдиної позиції в кожному рядку ($1 \leq i \leq n$) та в кожному стовпці ($1 \leq j \leq n$).

3.2 Загальна характеристика завдання

Комівояжер – бродячий торговець відвідує населені пункти (n), пов'язані розгалуженою дорожньою мережею, проїзд якою оплачується окремо між i -м, j -м пунктами мережі. Наслідуючи уздовж маршруту (туру), побувати необхідно в кожному з n пунктів (одноразово) і повернутися звідки вийшов. Це – формуллювання замкнутого завдання, можна не повернатися до пункту відправлення, і завдання стає незамкненим. Симетричне завдання. Симетрична проблема комівояжера (TSP - traveling salesman problem) виникає, коли вартість ($C_{ij} = C_{ji}$) в обидва кінці між i -м, j -м пунктами однаакова. Вибір маршруту диктується витратами, які мінімізуються

торговцем. Асиметрична проблема комівояжера (ATSP) допускає несиметричність матриці $C_{ji} \neq C_{ij}$. У ще більш загальному випадку, шляхи між деякими містами можуть бути відсутніми, а щоб вони не вибиралися ним вписують у матриці $C_{ji} = \infty$ нескінченну довжину). Завдання з частковим упорядкуванням (SOP = sequential ordering problem), що вимагає, щоб певне місто i було відвідане до міста j (таких умов може бути кілька). Пошук циклу Гамільтона (HCP = narniltonian cycle problem) – Виявлення в довільному графі замкнутих шляхів, що проходять через кожну вершину точно один раз. У TSP (симетричній задачі комівояжера) шлях замкнений і стартувати можна з будь-якого міста (i в будь-який бік). Для n міст існує $(n - 1)!/2$ різних шляхів. Факторіал росте дуже швидко: $n! \sim nn$ і простір у якому шукається оптимальне рішення виявляється величезним. Наприклад, для 15 міст існує 43 мільярди маршрутів та для 18 міст вже 177 трильйонів.

Модельний повнозв'язковий n -вершинний орієнтований граф (орграф) задачі є таким, що між кожною (i, j) парою вершин існує 2 дуги з різною вартістю проїзду та $C_{ij} \neq C_{ji}$ (односторонній рух). Таким чином, розв'язання задачі комівояжера полягає у відшуканні на нашему орграфі маршруту, що проходить одноразово через усі (n) вершини, при найменшій його вартості. Всі такі існуючі ормаршрути в орграфах називають Гамільтоновими циклами, які задаються одноцикловими перестановками на безлічі вершин графа, а для n міст завжди існує $\frac{1}{2}(n - 1)!$ різних маршрутів.

Гамільтоновим циклом називається ормаршрут орієнтованого графа, що включає по одному разу кожну його вершину, виключаючи вихідну.

Підстановна матриця – матриця, що відповідає перестановці, цифра позиції якої відповідає рядку, а порядковий номер позиції стовпцю. Такі матриці називають діагональними.

Одноциклою називається перестановка – елемент ступеня n симетричної групи S_n , якому відповідає один цикл.

Простий приклад для 4-х міст. На цьому прикладі покажемо та опишемо основи методу гілок та кордонів (МГК), не вирішуючи його. Рішення легко отримає самостійно сам читач після прочитання статті. У всіх контрольних положеннях МГК у прикладі є відповіді для самоперевірки. Спочатку покажемо сутність процесу рішення ЗК з усіма елементами методу гілок і кордонів (МГК), з поняттями і позначеннями, що використовуються, але без деталей МГК. На прикладі “Рисунок 3.1 – Повне дерево пошуку оптимального рішення задачі комівояжера” Позначені: корінь дерева пошуку оптимального рішення: у корені R безліч усіх допустимих рішень $R=\{(1234),(1243),(1432),(1423),(1324),(1342)\}$; корінь містить усі 6 рішень – одноциклових перестановок симетричної групи S_4 ; інші вузли дерева позначені символами X та Y ; X завжди використовується як ім'я поточного вузла, що

обробляється; дочірні вузли X (результати розгалуження) позитивний Y і негативний Y – вузол (з підкresленням зверху чи знизу), у якому алгоритм відмовляється здійснювати розгалуження (у ньому цьому кроці маршрут не нарощується новими гілкою і вузлом). Приклад дерева пошуку оптимального рішення задачі комівояжера наведений на рисунку 3.1.

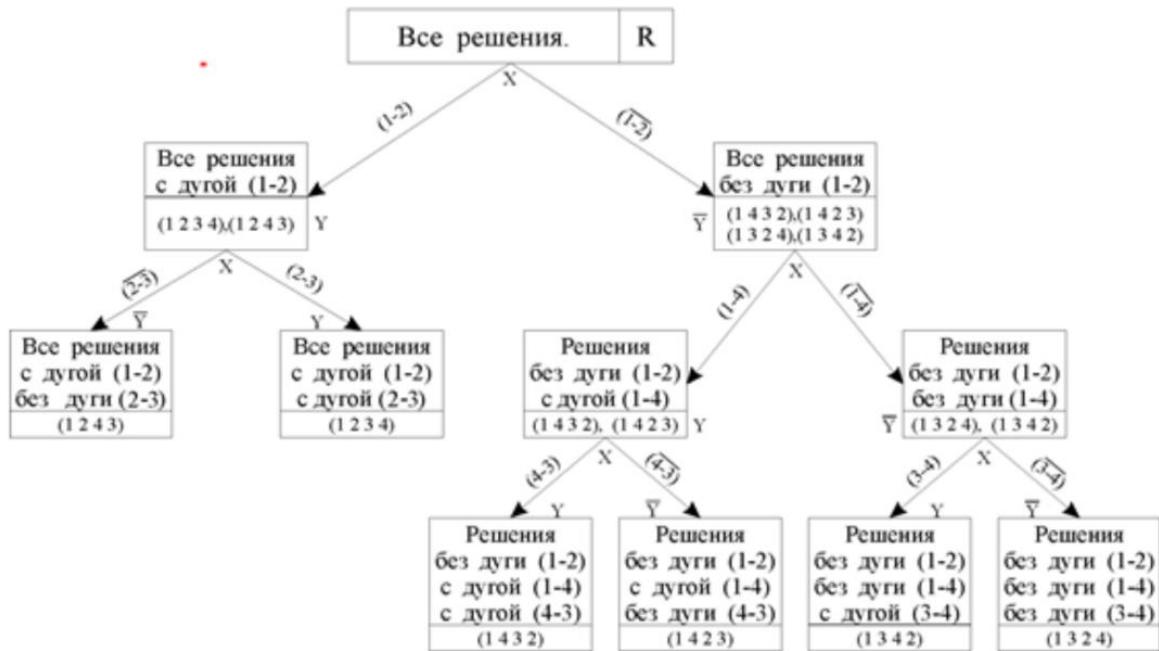


Рисунок 3.1 – Повне дерево пошуку оптимального рішення задачі комівояжера

Продемонструємо перебіг рішення на невеликому прикладі простим перебором маршрутів. Цей приклад покликаний ілюструвати загальний принцип та поняття МГК. Деталі будуть подані нижче з вичерпною подробицею. З цією метою сформуємо зведену таблицю для дорожньої мережі із 4-х вузлів права клітина таблиці. Приклад наведений на рисунку 3.2.

Формується безліч $S \{(i, j)\}$ клітин наведеної матриці з нульовими значеннями. Починаємо будувати маршрут (малюнок 1) з вершини 1, включаючи в нього гілку (1-2), негативну гілку (1-2) забороняємо включати до маршруту присвоєнням елементу $C_{12} = \infty$ великої вартості. При цьому безліч рішень в корені дерева розпадається спочатку на два підмножини Y і \bar{Y} , потім кожне з отриманих підмножин розчленовується на дрібніші і процес дроблення підмножин рішень може продовжуватися, поки в кожному з підмножин залишиться по одному рішенню. Але найчастіше до цього не доходить. У процес втручаються інші процедури та розвиток процесу відбувається в інших напрямках. Іноді розпочатий маршрут доводиться

кидати не завершивши, і повертачися до раніше перерваного та зупиненого. Річ у тім, що з нарощуванні маршруту відстежується його якість. Якщо якість маршруту можна покращити, повертаючись до перерваного раніше маршруту, то продовжувати ущербний (неякісний) маршрут не варто. Зауважимо, що розв'язання задачі моделюються перестановками п вершин (міст), тобто. елементами симетричної алгебраїчної групи S_n ступеня n . Приклад наведено у таблиці 3.1.

исходная матрица	приведение строк	приведение столбцов	граф путей задачи
$C = 2$ $\begin{array}{cccc} & 1 & 2 & 3 & 4 \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \left(\begin{array}{ccccc} \infty & 5 & 16 & 14 \\ 13 & \infty & 6 & 9 \\ 10 & 12 & \infty & 11 \\ 8 & 15 & 7 & \infty \end{array} \right) \end{array}$	$H=5+6+10+7+1=29$ $\begin{array}{ccccc} 1 & 2 & 3 & 4 & h_i \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ h_j \end{matrix} & \left(\begin{array}{ccccc} \infty & 0 & 11 & 9 & 5 \\ 7 & \infty & 0 & 3 & 6 \\ 0 & 2 & \infty & 1 & 10 \\ 1 & 8 & 0 & \infty & 7 \\ 0 & 0 & 0 & 1 & n \end{array} \right) \end{array}$	$\begin{array}{ccccc} 1 & 2 & 3 & 4 & h_i \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ h_j \end{matrix} & \left(\begin{array}{ccccc} \infty & 0 & 11 & 7 & 5 \\ 7 & \infty & 0 & 2 & 6 \\ 0 & 2 & \infty & 0 & 10 \\ 1 & 8 & 0 & \infty & 7 \\ 0 & 0 & 0 & 1 & 29 \end{array} \right) \end{array}$	

Рисунок 3.2 – Вихідні дані та приведення матриці вартості поїздок

Не всі перестановки є допустимими рішеннями і, звісно, далеко ще не всі серед допустимих будуть оптимальними. У симетричній групі підстановок реалізується розбиття на класи, що не перетинаються, сполучених елементів. Один такий клас складається з безлічі допустимих рішень – це клас одноциклових підстановок, тобто. підстановок утворюють цикл, що включає всі елементи.

Таблиця 3.1 – Варіанти маршрутів

№ п\п	перетановування рішення Зк	циклічне уявлення	значення цільової функції (ЦФ)	нижня межа ЦФ (НМЦФ) всіх рішень
1	1234	(1)(2)(3)(4)	9	
2	1243	(1)(2)(34)	8	
3	1324	(1)(23)(4)	10	
4	1423	(1)(234)	11	
5	1432	(1)(24)(3)		
6	2134	(12)(3)(4)		
7	2143	(12)(34)	7	

Продовження таблиці 3.1

8	2314	(123)(4)		29
9	2341	(1234)		
10	2413	(1243)	$5+6+11+8=30$	
11	2431	(124)(3)	$5+9+7+10=31$	
12	3124	(132)(4)		
13	3142	(1342)		
14	3214	(13)(2)(4)	$16+11+15+13=55$	
15	3241	(134)(2)	15	
16	3412	(132)(3)	13	
17	3421	(13)(24)		
18	4123	(1324)	$16+12+9+8=45$	
19	4132	(1432)	$14+7+12+13=46$	
20	4213	(142)(3)		
21	4231	(143)(2)	6	
22	4321	(14)(2)(3)		
23	4312	(1423)	$14+15+6+10=45$	
24	4321	(14)(23)	12	

Аналіз множини рішень ЗК. У таблицю 3.1 включено все $n!=4!=24$ можливих перестановок – рішень (Планів X) завдання, вони впорядковані за їх лексикографічним номером. Для кожної перестановки включено її подання циклами (колонка 3). Наприклад, перестановці (12)(34) у рядку 8 відповідають пари дуг $1 \rightarrow 2$ та $1 \leftarrow 2$; $3 \rightarrow 4$ та $3 \leftarrow 4$, які не реалізують маршруту (єдиного циклу). А перестановці (1234) з рядка 10 відповідає ланцюжок $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$, тобто. вийшов замкнутий цикл, до якого включаються всі вершини графа доріг, але одноразово. Вибору дуг цього циклу відповідають елементи матриці: $C_{12} = 5$, $C_{23} = 6$, $C_{34} = 11$, $C_{41} = 8$. Їх сума – значення ЦФ маршруту ЦФ = $5 + 6 + 11 + 8 = 30$. З $n! = 4! = 24$ перестановок номерів вершин графа колій циклічними (одноцикловими) є лише шість $(n-1)! = (4-1)! = 3! = 6$ (іхні номери 10,11,14, 18,19,23). Маршрути в мережі (тури) – допустимі обмеження моделі рішення наведені в таблиці (3-я колонка виділені жирним шрифтом). Серед них можна здійснити вибір кращого туру. Для кожного з шести маршрутів підраховано значення цільової функції (4-я колонка ЦФ: 30,31,55,45,46,45). Серед цих значень найменше дорівнює 30.Хоча воно і перевищує

нижню межу ЦФ рівну 29. Саме це рішення відповідно до дерева пошуку рішень на підставі наведеної матриці вартостей є оптимальним $T_{opt} = 30$.

Для розробки клієнтської частини веб-застосунку і візуалізації алгоритму застосовувався JS фреймворк Angular останньої версії який зосереджений на компонентному програмуванні, бібліотека NgPrime та Google Maps API. Для розробки серверної частини застосовувався ASP.Net Core 6, Rest Web API, C# 8.0. Для сбереження даних застосовувався SQL Server та Entity Framework Core code first.

Послідовність реалізації (рисунок 3.3) генетичного алгоритму зосереджена на клієнтської частині і розбита на частини:

- 1) на початку користувачеві необхідно позначити вхідні параметри на конфігурацію алгоритму, а саме кількість поколінь, популяції, % мутації, та пункти на мапі;
- 2) при кожному додаванні нового пункту додаються координати, ім'я пункту, іконка на мапі та розрахунок довжини шляху від кожного пункту до нового у колекції Dictionary, де зберігається довжина для кожної унікальної послідовності;
- 3) далі є можливість наглядно виобразити послідовний шлях який починається і закінчується в одному і тому ж пункті без повторних напрямків і розраховану суму шляху;
- 4) при запуску алгоритму створюються випадкові унікальні ціле численні послідовності або комбінації масивів за кожну із популяцій (рисунок 3.4);
- 5) розраховуються суми шляхів для кожної із комбінацій;
- 6) далі за кожне покоління застосуємо турнірний відбір для комбінацій для вияву найкращих пристосованих осіб як зворотний масив;
- 7) далі виконуємо схрещування між парними та не парними елементами турнірного масиву (рисунок 3.5):
 - 7.1) генеруємо точку розриву від (2 до Plen-2), де Plen – довжина одного із батьків;
 - 7.2) формування першого потомку – послідовність у першого батька до точки розриву + послідовність у другого батька після точки розриву;
 - 7.3) якщо залишилися не заповнені (довжина батька не равна довжині нащадка) – заповнюються гени у першого з батьків після точки розриву;
 - 7.4) формування другого потомку – послідовність у другого батька до точки розриву + послідовність у першого батька після точки розриву;
 - 7.5) якщо залишилися не заповнені (довжина батька не равна довжині нащадка) – заповнюються гени у другого з батьків після точки розриву;

- 8) далі генерація випадкового числа від 0 до 100, якщо вхідний % менший за випадкове число – проводимо мутацію особі (рисунок 3.6);
- 9) додаємо до спільного масиву результатів;
- 10) після завершення останнього покоління сортируємо масив за зростанням та знаходимо перший елемент як краще вирішення, який і відображаємо;

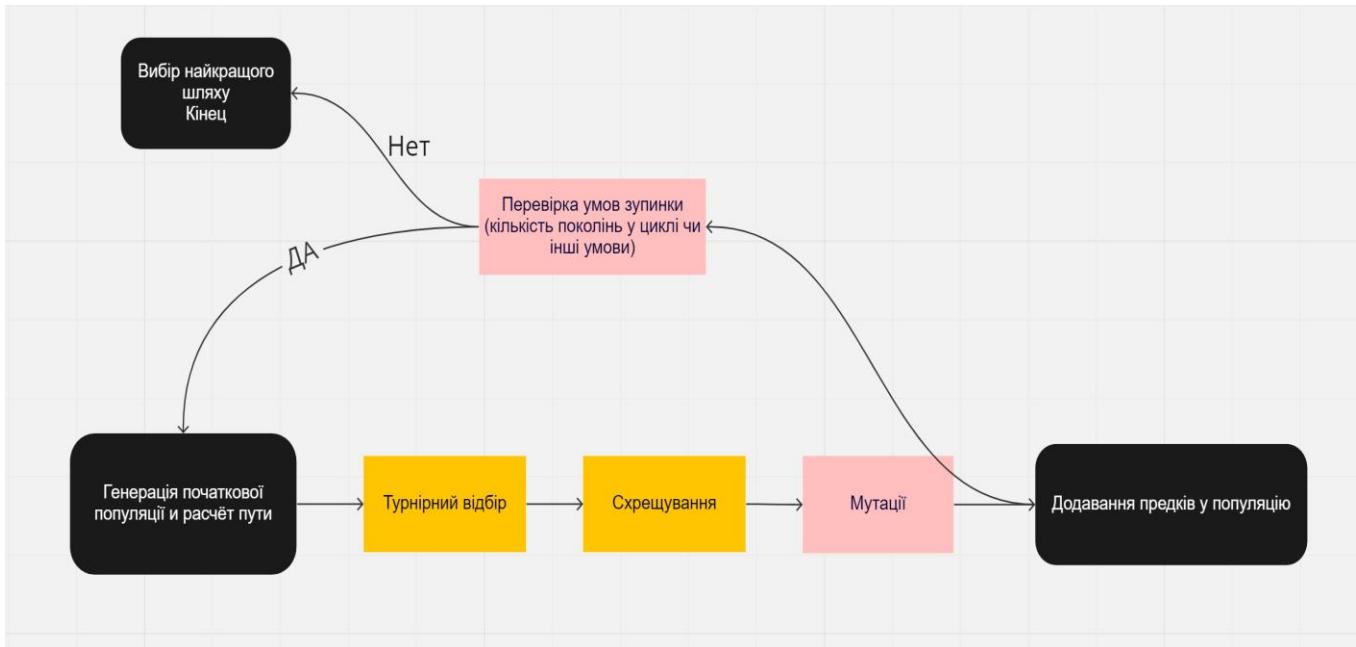


Рисунок 3.3 – Послідовність генетичного алгоритму

0	1	2	3	4	14
0	1	3	2	4	16
0	3	1	4	2	20
0	2	4	3	1	11
0	2	1	3	4	15
0	3	2	1	4	25
0	3	2	4	1	19

Рисунок 3.4 – Приклад популяції унікальних послідовностей

0	3	1	4	2	20
0	2	4	3	1	11
0	3	1	4	2	20
0	2	4	3	1	11
0	3	4	1		
0	3	1	4	2	20
0	2	4	3	1	11
0	3	4	1	2	18

Рисунок 3.5 – Схрещування потомка

0	3	4	1	2	
0	1	4	3	2	19

Рисунок 3.6 – Мутації

3.3 Подання об'єктів

У алгоритмі кожна з особин оцінюється мірою придатності згідно з тим наскільки "добре" відповідне їй рішення задачі. Найбільш пристосовані особини отримують можливість відтворювати потомство за допомогою перехресного схрещування з іншими особинами популяції. Найменш пристосовані особини з меншою ймовірністю зможуть відтворити нащадків. Щоб використовувати алгоритм, потрібно спочатку вибрати відповідну структуру для подання цих рішень. Структура

даних алгоритма (особина) складається з однієї або більше хромосом. Зазвичай хромосома являє собою рядок біт, тому часто використовується термін рядок. Кожна хромосома являє собою об'єднання певного числа подкомпонентов, які називаються генами. Гени є позиціями або локусами хромосоми і приймають значення звані алелями. Біологічний термін генотип відноситься до всього генетичним складом індивідуума, і відповідає структурі в алгоритм. Термін фенотип відноситься до зовнішніми характеристиками індивідуума і відповідає декодувати структури в алгоритмі. Щоб оптимізувати структуру за допомогою алгоритма, потрібно призначити деяку міру якості кожної структури в області пошуку. Це завдання виконує функція придатності. Алгоритм випадковим чином генерує початкову популяцію з L структур. На кожному поколінні алгоритма використовує оператори відбору, кросинговеру, мутації і позиціонування.

3.4 Особливості застосування

Структура хромосоми. У даному випадку застосування двійкових значень генів хромосоми не є виправданим, тому що таким чином ми не скоротимо перебір, а навпаки його розширимо. Тому використовуватимемо гібридний генетичний алгоритм компонування N модулів. Якщо ми закодуємо кожен модуль чотирма послідовними числами (две координати розташування на схемі, висота і ширина модуля), отримаємо хромосому довжиною $4*N$.

Особливості ініціалізації популяції. З метою скорочення часових витрат на знаходження потрібних параметрів модулів, ініціалізуємо особини початкової популяції не зовсім випадковим чином. Обиратимемо випадкову імплементацію деякого модуля з відповідної йому множини імплементацій, а координати модуля ініціалізуємо випадковим чином. На цьому етапі хромосома може закодувати модулі як без поворотів, так і з ними.

Придатність – це значення цільової функції у розв'язуваній задачі оптимізації. Більш «придатні» особини відбираються з поточної популяції, їхня хромосома модифікується (мутує випадковим чином) для формування нового покоління, яке потім використовується в наступній ітерації алгоритму. Зазвичай алгоритм завершується після досягнення заданої кількості ітерацій чи коли населення досягне задовільного рівня «придатності».

Перед тим як алгоритм розпочне роботу, потрібно визначити функцію придатності (фітнес-функцію) та подати можливі рішення (особи) у вигляді двійкових масивів. Потім алгоритм виконує такі дії:

1) ініціалізація – формування початкової популяції випадковим чином, число особин коливається від кількох сотень до кількох тисяч;

2) вибір (селекція) – кожної ітерації відбирається частина поточної популяції на формування наступного покоління. Відбираються особини, котрим значення фітнес-функції досить високе;

3) застосування генетичних операторів – до обраних на попередньому етапі особин застосовуються оператори схрещування та мутації. Для формуванняожної нової особини вибираються два батьки, які створюють нову особину, яка успадковує ознаки своїх батьків. Хоча методи репродукції, засновані на використанні двох батьків, більш відповідають біологічній основі методу, дослідження показали, що використання трьох і більше батьківських особливостей породжує нащадків вищої якості. Через війну середній рівень придатності популяції підвищується;

4) завершення – алгоритм продовжує роботу доти, доки не буде виконано одну з умов зупинки:

4.1) знайдено рішення, що задовольняє деякий мінімум фітнес-функції; досягнуто заданої кількості ітерацій;

4.2) значення фітнес-функції нових особин перестало зростати.

Цільова функція. У широкому значенні цільова функція є математичним виразом деякого критерію якості одного об'єкта (рішення, моделі, процесу і т.д.) у порівнянні з іншим. Приклад такого підходу є, наприклад, середньоквадратичний критерій точності апроксимації. Цільова функція в цьому випадку може бути записана так:

$$q(x_{0oц}, x_{1oц}, \dots, x_{Noц}) = \sum_{i=1}^N (x_i - x_{ioц})^2, \quad (3.6)$$

іншими словами, **мета** – знайти такі оцінки $x_{ioц}$, за яких цільова функція досягає мінімуму. Важливо, що критерій завжди привноситься ззовні і тільки після цього шукається рішення, яке мінімізує або максимізує цільову функцію.

Селекція. Для виконання вибірки хромосом подальшого схрещування ми пропонуємо застосувати селекції двійкового (парного) турніру. Як найпопулярніший метод селекції, вона чудово підходить для реалізації генетичного алгоритму. Таким чином, зі всієї популяції випадковим чином обираються пари хромосом, а зожної пари до проміжного масиву відбирається одна, із кращим (меншим) значенням функції пристосованості. Ця процедура повторюється стільки разів, скільки потрібно

особин для проміжної популяції. Потім в отриманому проміжному масиві застосовуються генетичні оператори.

Селекція рулеткою. Метод рулетки (рисунок 3.7) використовується визначення, які розв'язання завдання чи члени популяції вибираються для розмноження. У основі ідеї методу лежить уявлення популяції як колеса рулетки, де кожної особини є сектор, розмір якого пропорційний значенню її показника пристосованості. На "колісі" вибирається фіксована точка і воно "обертається". Особина, навпроти якої зупиняється точка, вибирається як батьківська. Так само вибирається і другий батько (рисунок 3.7).

$$p_i = \frac{f_i}{\sum_{i=1}^N f_i}, \quad (3.7)$$

де, p_i – ймовірність вибору i особини, f_i – значення функції придатності для i особини, N – кількість особин у популяції. Очевидно, що чим вище показник пристосованості особини, тим ширше відповідний сектор і тим більша ймовірність для особи бути обраною як батьківська.

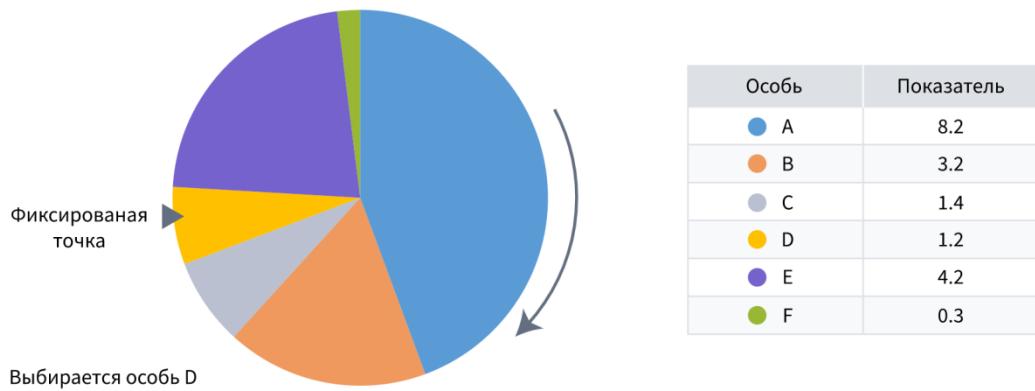


Рисунок 3.7 – Селекція рулеткою

Цей метод можна використовувати лише у завданнях максимізації функції (але не мінімізації). Очевидно, що проблему мінімізації можна легко звести до завдання максимізації функції та назад. У деяких реалізаціях генетичного алгоритму метод рулетки використовується для пошуку мінімуму функції (а не максимуму). Це результат відповідного перетворення, яке виконується програмним шляхом для зручності користувачів, оскільки в більшості прикладних завдань вирішується

проблема мінімізації (наприклад, витрат, відстані, похиби тощо). Проте можливість застосування методу рулетки лише одного класу завдань, тобто. тільки для максимізації (або тільки для мінімізації) можна вважати його безперечним недоліком. Інша слабка сторона цього у тому, що особини з дуже малим значенням функції пристосованості занадто швидко виключаються з популяції, що може призвести до передчасної збіжності генетичного алгоритму. Для запобігання такому ефекту застосовується масштабування функції пристосованості. З урахуванням зазначених недоліків методу рулетки створено та використовуються альтернативні алгоритми селекції. Один із них називається турнірним методом (tournament selection). Уявімо ці методи докладніше.

При турнірної селекції всі особи популяції розбиваються на підгрупи з наступним вибором у кожному їх особини з найкращою пристосованістю. Розрізняються два способи вибору: детермінований вибір (deterministic tournament selection) і випадковий вибір (stochastic tournament selection). Детермінований вибір здійснюється з ймовірністю, що дорівнює 1, а випадковий вибір – з ймовірністю, меншою за 1. Підгрупи можуть мати довільний розмір, але найчастіше популяція поділяється на підгрупи по 2 – 3 особи в кожній.

Турнірний метод придатний вирішення завдань як максимізації, і мінімізації функції. З іншого боку, може бути легко поширеній завдання, пов'язані з багатокритеріальної оптимізацією, тобто. на випадок одночасної оптимізації кількох функцій. У турнірному методі допускається зміна розміру підгруп, куди підрозділяється населення (tournament size). Дослідження підтверджують, що турнірний метод діє ефективніше, ніж метод рулетки.

Оператори схрещування. У алгоритмі застосовані два гіbridних оператори схрещування. Отримані в результаті першого оператора дочірні хромосоми приносять більше розмаїття у популяцію, а у результаті другого – не так радикально еволюціонують. Перший оператор схрещування – оператор одноточкового схрещування OnePointCrossover – подібний до оператора схрещування канонічного генетичного алгоритму, за однією тільки відмінністю, що «точка розрізу», за якою обидві батьківські хромосоми, поділяються на два сегменти, обирається випадковим чином між групами генів, що закодовують окремі модулі, а не всередині групи. Це зумовлено особливостями структури хромосоми у нашому проекті.

За таким само принципом між групами генів відбуваються «розрізи» другого оператора схрещування – оператора двоточкового схрещування. TwoPointCrossover. Він «розрізає» хромосоми по краях сегменту генів, відповідального за один і той самий модуль, і за допомогою цього оператора мутації хромосоми «обмінюються» інформацією про імплементацію цього модуля. Схематичне зображення дії обох

операторів мутації наведене на (згори – OnePointCrossover; знизу – TwoPointCrossover). Приклад наведений на рисунку 3.8.

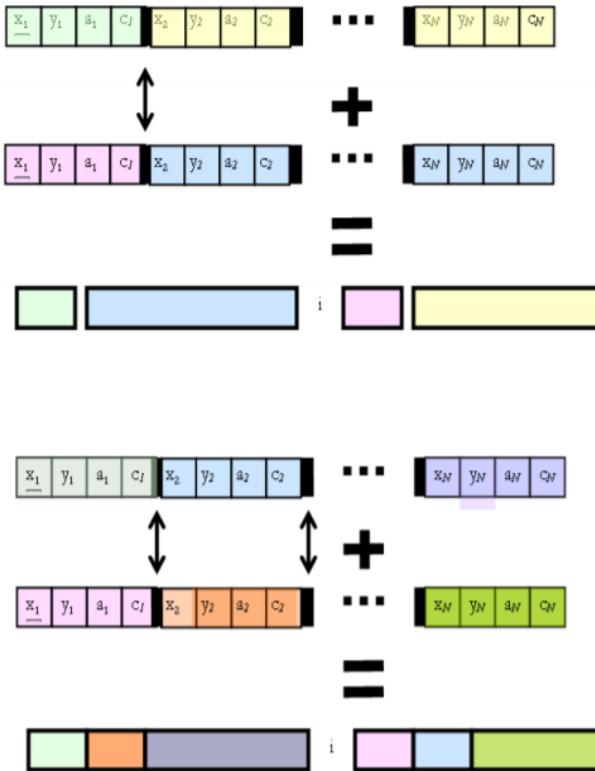


Рисунок 3.8 – Оператори схрещування: вгорі – одноточкове схрещування; внизу – двоточкове схрещування

Оператори мутації. У алгоритмі визначено також два гібридні оператори мутації. У їх роботі ми враховуємо такі особливості постановки задачі як скінченність шмату матеріалу і безпосереднє визначення користувачем множини можливих імплементацій конкретного модуля.

Зміст роботи першого визначеного оператора мутації (MutatePosition) полягає у мутації гена, відповідального за одну з координат обраного випадковим чином модуля. Це відбувається за рахунок фактичного збільшення/зменшення параметрів координат. Якщо при збільшенні параметр координати виходить за задані межі площини матеріалу, ми надаємо йому значення 0 і, таким чином, розміщуємо поточний модуль біля краю площини. Цей оператор забезпечує знаходження оптимального місця розташування відповідного модуля на інтегральній схемі. Зміст другого гібридного оператора мутації MutateImplementation полягає у фактичній заміні поточної імплементації модуля на іншу з відповідної множини імплементацій.

Ця заміна допомагає нашому оптимізаційному алгоритму уникнути «зациклювання» на деякому локальному екстремумі, а також поступово еволюціонувати у напрямку покращення результата. Також важливим є той факт, що цей оператор мутації із деякою ймовірністю реалізує поворот модуля.

Функція пристосованості. Як вже зазначалося вище, хромосоми в імітації процесу природного відбору потребують певної якісної оцінки їх життєздатності, тому для кожної хромосоми з популяції обчислюється функція пристосованості. Ця функція показує міру оптимальності розв'язку, тобто чим краще значення функції для хромосоми, тим краще вона як розв'язок задовольняє умовам оптимізаційної задачі.

Критерії зупинки роботи алгоритму. Для того, щоб алгоритм працював у межах розумного часу, визначаються критерії зупинки роботи алгоритму. По-перше, якщо протягом визначеного кількості поколінь не відбувається покращення найліпше пристосованої хромосоми, алгоритм припиняє роботу. Цей випадок означає, що хромосоми популяції майже однакові і знаходяться у області деякого екстремуму, а схрещування вже майже ніяк не впливає на популяцію, бо дочірні особини або є абсолютноними копіями особин з цієї популяції, або виходять за межі області прийнятних розв'язків і виявляються менш пристосованими, ніж інші особини популяції.

Іншими словами, зупинка відбувається у випадку, коли популяція збіглася – просто знайдений найкращий або близький до нього розв'язок.

Генетичні алгоритми надають можливість швидкої генерації прийнятних розв'язків задач, які неможливо розв'язати іншими традиційними аналітичними методами. Як недоліки генетичних алгоритмів можна виділити:

- 1) відсутність масштабованості.
- 2) евристичний характер методу;
- 3) можливість збіжності алгоритму локального мінімуму.

3.5 Метод гілок та кордонів

Пов'язують із деревом пошуку оптимального рішення, яке будується у процесі обробки вихідних даних завдання. Звідси назви корінь, якому в дереві приписують усі можливі завдання, рішення-гілки, що з'єднують вузли дерева, Використання поняття кордонів та його розрахунок стимулює чи гальмує зростання гілок у такому дереві. Важливу роль грає процедура розбиття на вузли області допустимих рішень (ОДР) вихідного завдання, тобто. на менші непересічні підмножини та їх оцінювання. Інша процедура, названа процедурою розгалуження, реалізує розбиття

на безліч допустимих значень змінної x на під області менших розмірів. Ще один важливий елемент методу – процедура обчислення оцінок, яка полягає у пошуку значень для вирішення задачі. Обчислення нижньої межі є найважливішим, ключовим елементом запропонованої схеми. Таким чином, в основі методу гілок і кордонів лежить ідея послідовного розбиття безлічі допустимих рішень на підмножини (стратегія "поділяй і владарюй") та оцінювання одержуваних при розбитті частин. Кожен крок алгоритму розбиття супроводжується перевіркою умови того, чи конкретне підмножина містить оптимальне рішення чи ні.

Приклад вирішення задачі методом гілок та кордонів. Число міст $n = 5$. Граф дорожньої мережі, що зв'язує міста, повний без петель, орієнтований, несиметричний звичайний, задається матрицею розміру 5×5 , дуги графа зважені; рядки матриці відповідають пунктам відправлення, стовпці – пунктам прибуття. Позначимо початкову (виходну) нижню межу цільової функції символом із значенням $Q^* = \infty$ – НГЦФ; вона служить порівняння з нею поточних оцінок. Як тільки вдається сформувати повний маршрут і обчислити для нього ЦФ, значення $Q^* = \infty$ замінюється на нове, ним стає обчислена ЦФ, з якою далі порівнюються та оцінюються інші маршрути, $H_C = 1+5+6+7+8+3+1 = 31$ – значення (оцінка) НГЦФ отримано як сума констант приведення рядків і стовпців матриці вартості. Вага дуг (вартість проїзду по дузі) задається елементами C_{ij} квадратної матриці. Ця матриця щін $C[5]$ – основний об'єкт перетворень у задачі. Усі діагональні елементи матриці $C[5]$ нічого не винні включатися в Т маршрут (в тур), оскільки відповідні їм дуги – це петлі графа і тому їм приписані дуже великі вартості (∞). Інші значення клітинах заповнені поспіль наступними натуральними числами (у прикладі спірально за годинниковою стрілкою від клітини $C_{11}=1$ до центру матриці). Робота з пошуку маршруту проводиться на постійному графі доріг (з вершинами і дугами) і на дереві, що послідовно вирощується, (з вузлами і гілками) пошуку рішень. Поточному вузлу дерева пошуку рішення на кожному кроці алгоритму присвоюється ім'я X , а вузлам-результатам розгалуження ім'я Y -позитивному, що включається в тур, і ім'я Y -негативному вузлу, що не включається до Т тур (маршрут).

Багатокріковий алгоритм пошуку на дереві рішень

Швидке розв'язання задачі досягається при забезпеченні наявності хоча б одного нуля в кожному рядку і кожному стовпці матриці $C [5]$ при їх "діагональному" розташуванні. В цьому випадку маршрут можна прокласти через клітини з нулями і вартість маршруту буде найменшою. Далі ми побачимо, що таке положення нулів у матриці забезпечується багаторазовим застосуванням процедури приведення матриці [5] та її перетвореннями, що скорочують розмірність до $\text{dim}C [5] = 2 \times 2$. У момент досягнення матрицею розмірності 2×2 вона забезпечує однозначне

визначення вершин, що включаються до маршруту. Розглядаються характеристики (d_i, d_j) дуг графа доріг у рядку d_i і в стовпці d_j матриці, що змінюються $C[n]$ вартостей. Вузли X дерева розгалужуються на "позитивний" та "негативний". Безліч висячих «відкладених» вузлів (листя) у дереві рішень позначається символом, а початковому значенню нижньої межі цільової функції задається $Q^* = \infty$. Кроки алгоритму бувають двох типів: звичайні та з поверненням у відкладений вузол; другий тип реалізує процедуру "back tracking".

На кожному кроці алгоритму розв'язання задачі комівояжера виконуються (визначаються):

- 1) приведення матриці $C[i,j]$ для отримання нульових елементів у кожних її рядку та стовпці;
- 2) формування множини клітин S з нульовими значеннями в матриці та їх характеристик d_i, d_j, \sum_{ij} у формі плоскої таблиці;
- 3) обчислюються $\max \sum_{ij}$ та визначається клітина (k, l) з \max яка не включається до маршруту;
- 4) модифікується матриця: з матриці видаляються k рядок і l -й стовпець клітини (k, l) ;
- 5) обчислюються оцінки НГЦФ для позитивного та негативного вузлів дерева пошуку рішення;
- 6) у маршрут включається елемент (k, l) з кращою (меншою) оцінкою $\omega(k, l)$ НГЦФ;
- 7) порівнюються оцінки між собою вихідна Q^* та обчислена H на поточному кроці;
- 8) обчислюється розмірність $\dim C[n]$ матриці та перевіряється рівність цієї розмірності з $\dim 2 \times 2$;
- 9) формується безліч висячих вузлів дерева рішень з їх оцінками НГЦФ;
- 10) наводиться фрагмент дерева пошуку рішень для наступного кроку.

Ітераційні кроки: Формування туру T починається вибором 1-ї дуги (k, l) графа доріг включення їх у маршрут T , оформленний як послідовність дуг $T = \{(a,b)(c,d)(ef)(hg), \dots\}$.

Процедура приведення вихідної матриці $C[n]$ (отримання нулів у ній). У кожному i -му рядку $C[n]$ знаходимо найменший елемент і віднімаємо його з усіх елементів рядка. Проходимо по всіх рядках матриці. Для такої зміненої матриці в кожному j -му стовпці знову знаходимо найменший елемент і віднімаємо його з усіх елементів кожного j -го стовпця. Проходимо по всіх стовпцях. Знайдені найменші елементи називають константами приведення рядків та стовпців (ліній матриці) та позначають h_i h_i , $i=1(1)2n$. Константою H_c приведення всієї матриці називають суму

$\sum h_i$ констант ліній $C[n]$. Ця сума $H_c (C[5]) = \sum h_i = 1+5+6+7+8+3+1= 31$ є оцінкою НГЦФ всіх розв'язків задачі. Приведення поточних матриць $C[n]$ на кроках алгоритму виконується лише у разі відсутності нулів у її деяких рядках та/або стовпцях, і нові константи визначаються тільки для таких ліній. Приклад наведений на рисунку 3.9. Нижче в таблицях показані карта доріг, дії та їх результати щодо приведення матриці завдання.

Для наведеної матриці рядкові константи h_i приписані в стовпці праворуч, а стовпцеві - у рядку знизу матриці. Константа приведення матриці ($C [5]$) = $\sum h_i = 31$ обчислюється один раз. Приклад наведений на рисунку 3.10.

Тепер нулі в клітинах матриці є у всіх рядках і стовпцях і навіть можливо більш ніж по одному. На жаль позиції, займані нулями, не утворюють діагонального плану X . Інакше рішенням завдання був би якраз цей план і вартість відповідного йому маршруту дорівнювала оцінці НГЦФ вихідної $C[5]$ матриці, тобто. $H = 31 = 1 + 5 + 6 + 7 + 8 + 3 + 1$.



Рисунок 3.9 – Карта доріг, дії

$$C_{[5]} = \begin{array}{|c|c|c|c|c|c|c|} \hline & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & 0 & 1 & 2 & 3 & 1 \\ \hline 2 & 9 & \infty & 10 & 11 & 0 & 5 \\ \hline 3 & 7 & 14 & \infty & 11 & 0 & 6 \\ \hline 4 & 5 & 12 & 11 & \infty & 0 & 7 \\ \hline 5 & 3 & 2 & 1 & 0 & \infty & 8 \\ \hline \end{array} \quad C_{[5]} = \begin{array}{|c|c|c|c|c|c|c|} \hline & 1 & 2 & 3 & 4 & 5 & \\ \hline 1 & \infty & 0 & 0 & 2 & 3 & 1 \\ \hline 2 & 6 & \infty & 9 & 11 & 0 & 5 \\ \hline 3 & 4 & 14 & \infty & 11 & 0 & 6 \\ \hline 4 & 2 & 12 & 10 & \infty & 0 & 7 \\ \hline 5 & 0 & 2 & 0 & 0 & \infty & 8 \\ \hline \end{array}$$

Приведение матрицы по строкам

Приведение матрицы по столбцам

Рисунок 3.10 – Матриця вартості з константами приведення рядками та стовпцями

3.6 Висновки за розділом

На підставі математичної моделі, були дані описи основнів алгоритмів які вирішують завдання комівояжера, їх особливості та реалізації в програмному забезпеченні.

4 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Клієнтська чатина

Для розробки архітектури клієнтської частини використовується Angular та яка поділена на наступні частини.

Components. відокремлена частина функціоналу зі своєю логікою, HTML-шаблоном та CSS-стилями. Клас стає Angular компонентом, якщо його оголошення передує декоратор @Component() з конфігураційним об'єктом. У компонентах знаходиться всі частини інтерфейсу користувача та логіка взаємодії з іншими компонентами (рисунок 4.1).

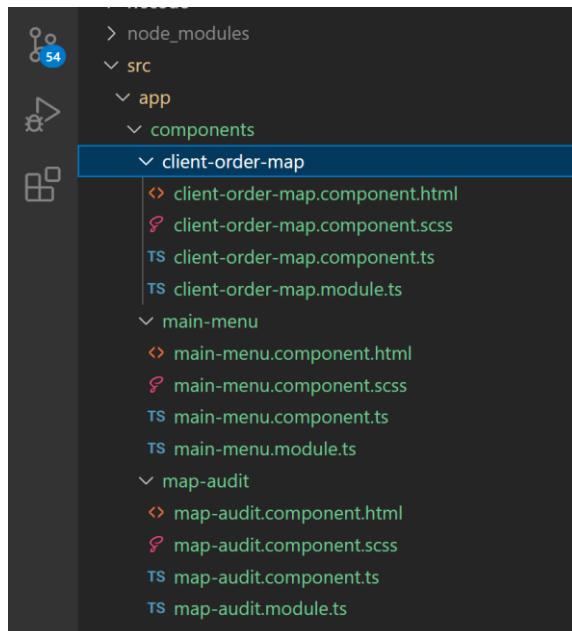


Рисунок 4.1 – Компоненти

Директиви – Angular директиви використовуються для зміни зовнішнього вигляду або поведінки DOM-елемента. Виділяють три типи директив:

- 1) з власним шаблоном, чи інакше компоненти (компоненти є директивами)
- 2) структурні, що змінюють структуру DOM-дерева;
- 3) атрибути, які змінюють зовнішній вигляд або стандартну поведінку елемента DOM-дерева.

Angular модуль – це клас з декоратором @NgModule(), який служить ізольуючою логічною структурою для компонентів, директив, фільтрів і сервісів. Всі ці сутності визначаються і конфігуруються за допомогою @NgModule(). Ключова

роль при створенні Angular модуля у декоратора `@NgModule()`, що приймає об'єкт конфігурації з властивостями:

- 1) imports – масив, де вказується список другорядних модулів, що імпортуються;
- 2) exports – масив компонентів, директив та фільтрів, якими користуються інші модулі, якщо вони імпортують поточний;
- 3) declarations – масив компонентів, директив та фільтрів;
- 4) entryComponents – масив створюваних динамічних компонентів;
- 5) bootstrap – масив, в якому вказується компонент для завантаження;
- 6) providers – масив сервісів.

Angular додаток має модульну архітектуру і складається принаймні з одного головного, або кореневого, модуля. Всі інші відносяться до другорядних (рисунок 4.2).

```

import { CommonModule } from "@angular/common";
import { NgModule } from "@angular/core";
import { FormsModule } from "@angular/forms";
import { BreadcrumbModule } from "primeng/breadcrumb";
import {ButtonModule } from "primeng/button";
import {CheckboxModule } from "primeng/checkbox";
import {DialogModule } from "primeng/dialog";
import {DropdownModule } from "primeng/dropdown";
import {GMapModule } from "primeng/gmap";
import {InputNumberModule } from "primeng/inputnumber";
import {InputTextModule } from "primeng/inputtext";
import {MessageModule } from "primeng/message";
import {MessagesModule } from "primeng/messages";
import {ClientOrderMapComponent } from "./client-order-map.component";
import {CalendarModule} from 'primeng/calendar';
import {OrdersService } from "src/app/services/order-service";
import {CarrierService } from "src/app/services/carrier.service";

@NgModule({
  declarations: [
    ClientOrderMapComponent
  ],
  imports: [
    GMapModule,
    MessagesModule,
    MessageModule,
    CheckboxModule,
    DropdownModule,
    FormsModule,
    InputTextModule,
    ButtonModule,
    InputNumberModule,
    BreadcrumbModule,
    DialogModule,
    CommonModule,
    CalendarModule,
    FormsModule
  ],
  providers: [OrdersService, CarrierService],
  bootstrap: [],
  exports:[ClientOrderMapComponent]
})

export class ClientOrderMapModule { }

```

Рисунок 4.2 – Модулі

Services Module. Модулі Сервісів. Наприклад – `HttpClientModule`. Практично будь-яка клієнтська програма отримує дані від віддаленого сервера. Більшість сучасних API засновані на протоколі HTTP, тому "спілкування" із сервером Angular здійснює через REST-подібні запити. За це відповідає `HttpClientModule`. Після компонент або сервіс (залежно від побудови архітектури) імпортуються сервіс `HttpClient`. Приклад “Рисунок 4.3 – Сервіс”.

```

1 import { Injectable } from "@angular/core";
2 import { HttpClient, HttpHeaders } from "@angular/common/http";
3 import { Observable } from "rxjs";
4 import { Order } from "../models/order";
5 import { DeliveryPath } from "../models/deliverPath";
6
7 @Injectable({
8   providedIn: 'root',
9 })
10
11 export class OrdersService {
12
13   url: string = "https://localhost:7275/order";
14
15   constructor(private httpClient: HttpClient) { }
16
17   public getAllOrders(): Observable<Order[]> {
18     return this.httpClient.get<Order[]>(`${this.url}/GetAllOrders`);
19   }
20
21   public getOrdersByCarrier(company: string): Observable<Order[]> {
22     return this.httpClient.get<Order[]>(`${this.url}/GetOrdersByCarrierCompany?companyName=${company}`);
23   }
24
25   public addOrder(order: Order): Observable<Order> {
26     return this.httpClient.post<Order>(`${this.url}/AddOrder`, order);
27   }
28
29   public deleteOrder(id: string) {
30     return this.httpClient.delete(`${this.url}/RemoveOrderById?savedPath=${id}`);
31   }
32
33   public getDeliveryPathsByCarrier(carrierId: number): Observable<DeliveryPath[]> {
34     return this.httpClient.get<DeliveryPath[]>(`${this.url}/GetDeliveryPathsByCarrier?carrierId=${carrierId}`);
35   }
36 }

```

Рисунок 4.3 – Приклад сервісу

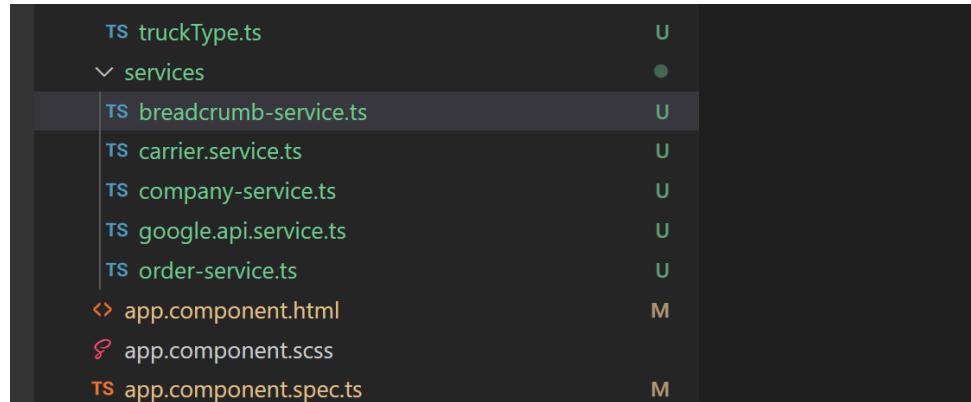


Рисунок 4.4 – Сервіси

Angular сервіс – це звичайний клас, що використовується в контексті Angular для зберігання глобального стану програми або як постачальник даних. Angular послуги можуть бути визначені на рівні програми, модуля або компонента. Angular сервіси можуть бути визначені на рівні програми, модуля або компонента (рисунки 4.4, 4.5).

```

    < models
      TS carrier.ts
      TS company.ts
      TS deliverPath.ts
      TS direction.response.ts
      TS distance.ts
      TS duration.ts
      TS leg.ts
      TS order.ts
      TS point.ts
      TS route.ts
      TS truckType.ts
  
```

```

  6   export class Bread {
  7     private itemsSou
  8
  9     itemsHandler = t
 10
 11     constructor() []
 12
 13     setItems(items: [
 14       this.itemsSou
 15     ]
 16   }
 17
 18 }
 19
  
```

Рисунок 4.5 – Моделі даних

У Angular маршрутизація є перехід від одного уявлення (шаблону) до іншого залежно від заданого URL. Причому навігація може здійснюватись і всередині уявлення. Навігація в додатках Angular відбувається без перезавантаження сторінки. За організацію маршрутизації в Angular відповідає модуль RouterModule бібліотеки @angular/router. URL-адреси організуються в спеціальні модулі та визначаються для кожного окремого модуля програми (рисунок 4.6).

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { RouterModule, Routes } from '@angular/router';
import { MessageService } from 'primeng/api';
import { AppComponent } from './app.component';
import { MapAuditComponent } from './components/map-audit/map-audit.component';
import { MapAuditModule } from './components/map-audit/map-audit.module';
import { BreadcrumbModule } from 'primeng/breadcrumb';
import { MenubarModule } from 'primeng/menubar';
import { HttpClientModule } from '@angular/common/http';
import { MainMenuModule } from './components/main-menu/main-menu.module';
import { MainMenuComponent } from './components/main-menu/main-menu.component';

const appRoutes: Routes = [
  { path: 'main-menu', component: MainMenuComponent },
  { path: 'carrier-land-audit', component: MapAuditComponent },
  { path: 'carrier-air-audit', component: MapAuditComponent }
];

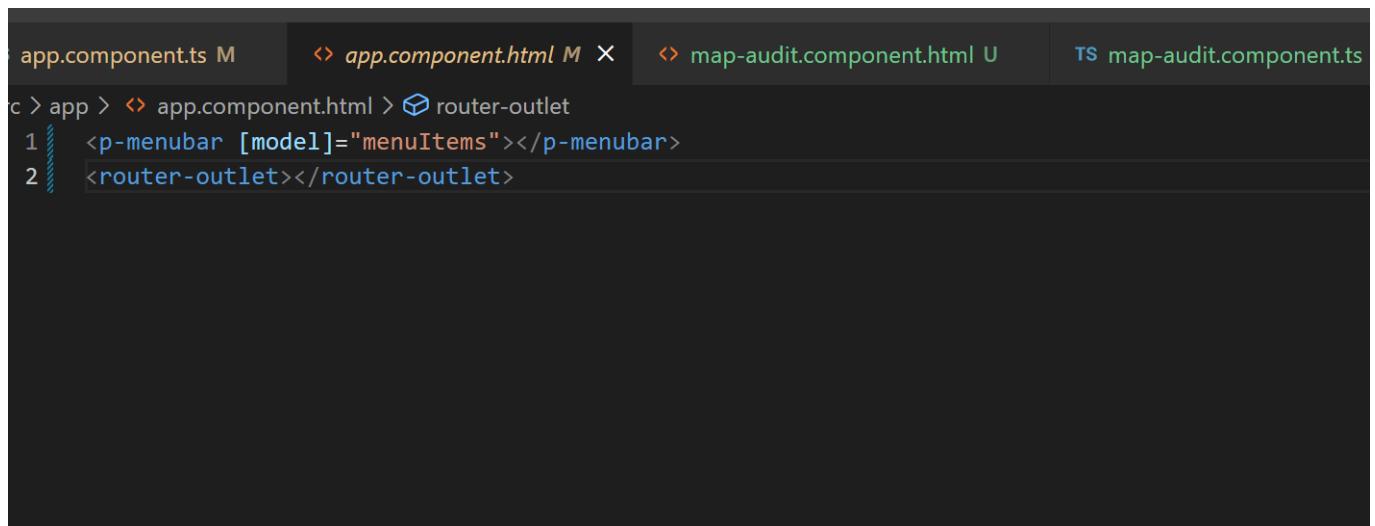
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    BrowserAnimationsModule,
    RouterModule.forRoot(appRoutes),
    MapAuditModule,
    HttpClientModule,
    BreadcrumbModule,
    MenubarModule,
    MainMenuModule
  ],
  providers: [MessageService],
  bootstrap: [AppComponent]
})
export class AppModule { }
  
```

Рисунок 4.6 – Router Module

При визначенні маршруту можна вказати низку властивостей:

- 1) path – найменування маршруту;
- 2) component – компонент для відображення під час переходу на URL, що збігається з path;
- 3) children – одна з додаткових властивостей, що поєднує групу маршрутів щодо поточного;
- 4) data – додаткові дані, наприклад, значення хлібних крихт;
- 5) redirectTo – перенаправляє на вказаний URL при попаданні на маршрут, вказаний у path;
- 6) pathMatch – використовується спільно з redirectTo.

Компоненти-маршрути повинні бути імпортовані в модуль Angular маршрутизації (рисунок 4.7). Визначення маршрутів далі передається як аргумент методу forRoot() модуля RouterModule. Компоненти, на які вказує Angular routing, підвантажуються до місця, де вказана директива <router-outlet></router-outlet>.



```
app.component.ts M app.component.html M map-audit.component.html U map-audit.component.ts
c > app > app.component.html > router-outlet
1 <p-menubar [model]=\"menuItems\"></p-menubar>
2 <router-outlet></router-outlet>
```

Рисунок 4.7 – Директиви маршрутизації

Для переходу по заданих URL використовується директива routerLink, яка може бути вказана не тільки у тега <a>, але й у будь-якого іншого блокового HTML елемента. Спільно з routerLink використовується директива routerLinkActive. Вона приймає назву класу, який буде доданий елементу (у якого вказані директиви) за

активної URL-адреси, на яку вони посилаються. Для продуктивної взаємодії з різними джерелами даних використовувалася бібліотека **RxJS** – є бібліотекою, що дозволяє керувати всіма асинхронними операціями та подіями в додатку в стилі реактивного програмування. Вона побудована на основі патерну проектування Observer і передбачає цілу низку операторів для маніпуляції асинхронними подіями та обробки даних, що передаються ними. RxJS операє об'єктами Observable, які існують у кількох різновидах (Subject, Scheduler) та реалізують принцип push-систем передачі даних від "постачальника" до "споживача". Крім push-систем виділяють ще й pull-системи передачі. Pull-системах "споживач" вирішує, коли отримати дані від "постачальника". Будь-яка JavaScript функція – яскравий приклад такої системи. Код виклику функції – "споживач", сама функція – "постачальник". У випадку RxJS Observable саме відправник вирішує, коли відправити дані одержувачу. До цього типу push-систем належать і об'єкти Promise, які надають дані зареєстрованим callback-функціям та ініціюють їх виклик. Бібліотека NgRx реалізує принцип роботи Redux для додатків Angular. Головна мета NgRx – централізувати та зробити максимально зрозумілим керування всіма станами програми. Мета досягається завдяки закладеним у бібліотеці кільком фундаментальним принципам:

- 1) наявність єдиного джерела даних про стан – сховища;
- 2) доступність стану лише читання;
- 3) зміна стану здійснюється тільки через дії, які обробляються редьюсерами (reducer), що є чистими функціями.

4.2 Серверна частина

Onion архітектура серверної сторони розділяється на:

- 1) data Access layer. Рівень доступа даних, яких містить логіку збереження даних за допомогою Entity Framework Core code first (структурну класів та структуру бази даних наведено на рисунках 4.8, 4.9);
- 2) infrastructure and Services layer (класи бізнес-логіки наведено на рисунку 4.10). Для реалізації бізнес-логіки програми, бібліотека, що відповідає за сервіси, бізнес логіку та DTO (data transfer objects які пов'язують Data Models та View Models);
- 3) presentation layer. Для реалізації веб сервіса Rest Api (рисунки 4.11, 4.12).

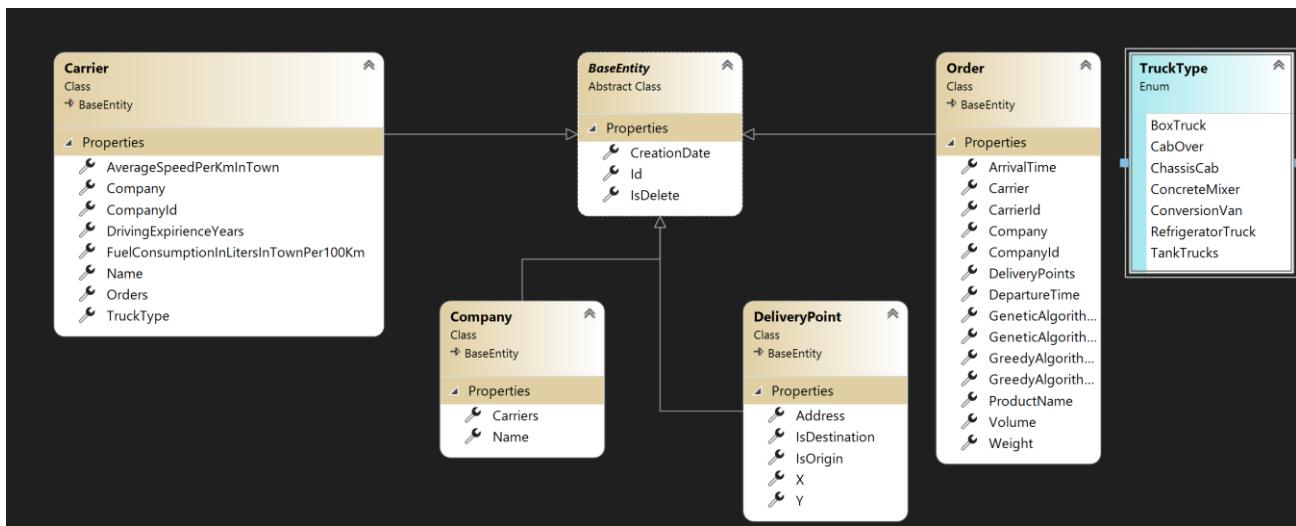


Рисунок 4.8 – Data Access layer діаграма класів

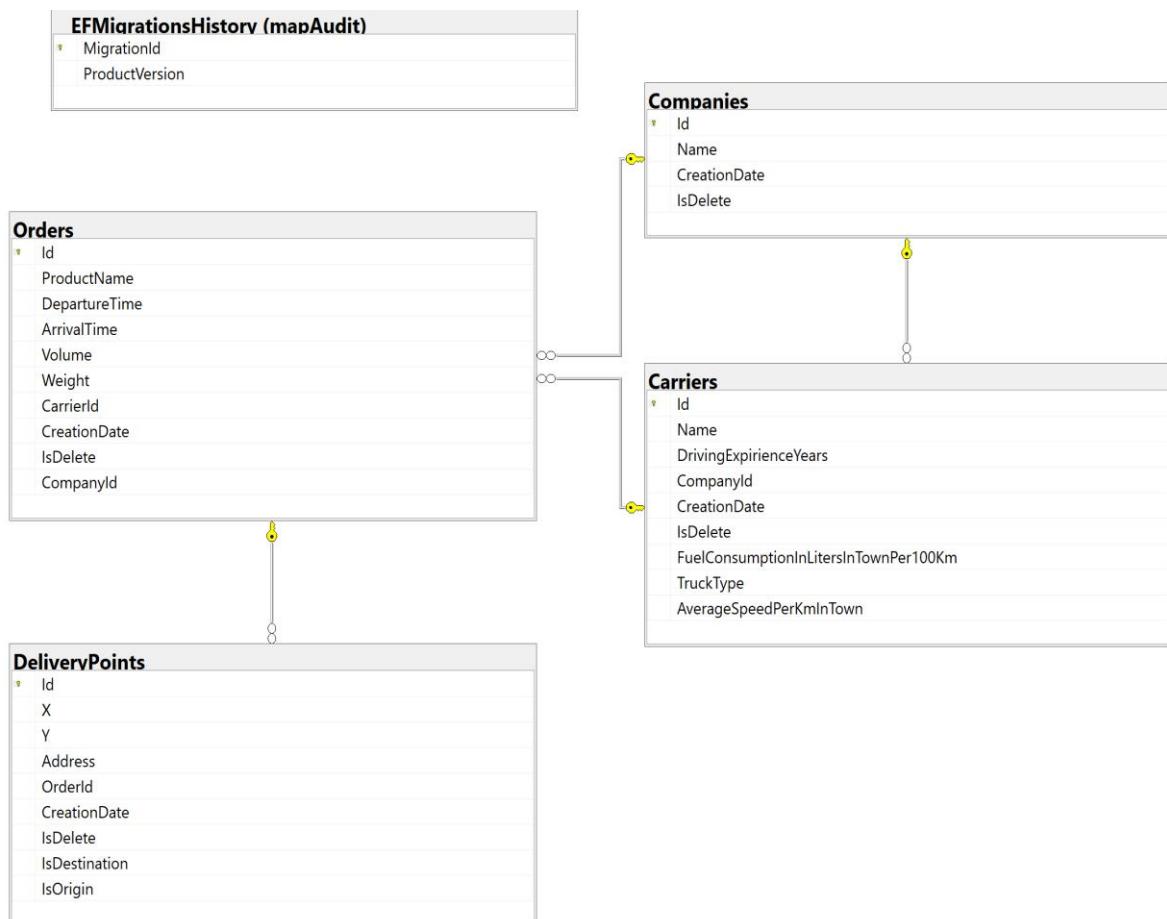


Рисунок 4.9 – Діаграма структури бази даних Sql Server

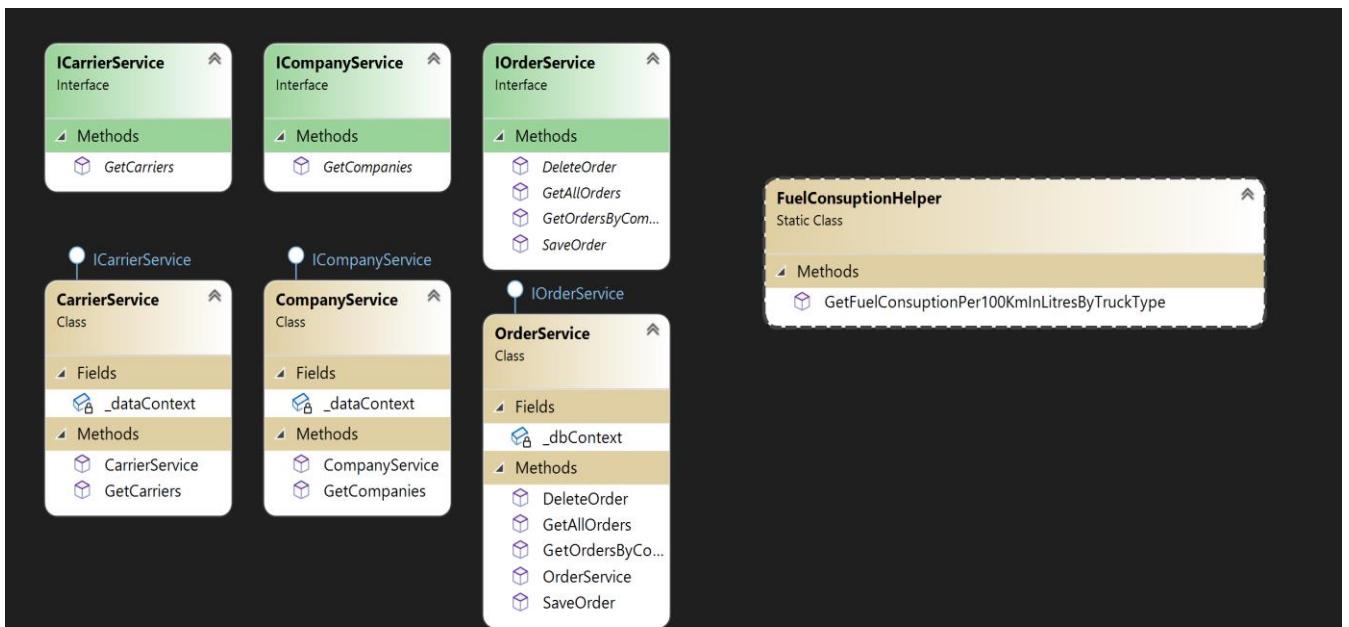


Рисунок 4.10 – Бізнес логіка

The screenshot shows the Swagger UI interface for the `MapAudit.WepApi` application. The top navigation bar includes the Swagger logo, the title `MapAudit.WepApi`, and a dropdown menu for selecting a definition, currently set to `MapAudit.WepApi v1`.

The main content area displays the API endpoints categorized by resource:

- Carrier**: Contains a single endpoint: `GET /Carrier/GetCarriers`.
- Company**: Contains a single endpoint: `GET /Company/GetAll`.
- Order**: Contains four endpoints:
 - `POST /Order/AddOrder`
 - `GET /Order/GetAllOrders`
 - `GET /Order/GetOrdersByCarrierCompany`
 - `DELETE /Order/RemoveOrderById`
- Schemas**: Displays two schema definitions:
 - `DeliveryPoint`
 - `OrderViewModel`

Рисунок 4.11 – Presentation layer

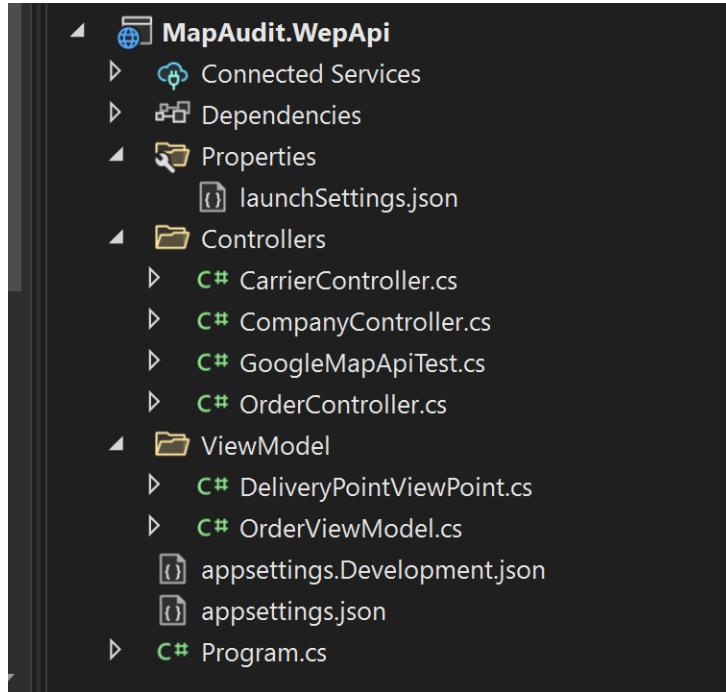


Рисунок 4.12 – Presentation layer

4.3 Висновки за розділом

Спроектували клієнтську та серверну частину, описали задіяні бібліотеки.

5 ОПИС РЕЖИМІВ РОБОТИ ТА ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ

5.1 Тестування генетичного алгоритму

Для створення замовлень клієнти на головній сторінці можуть вибрати спосіб доставки (по повітря або по суші). Приклад наведений на рисунку 5.1.

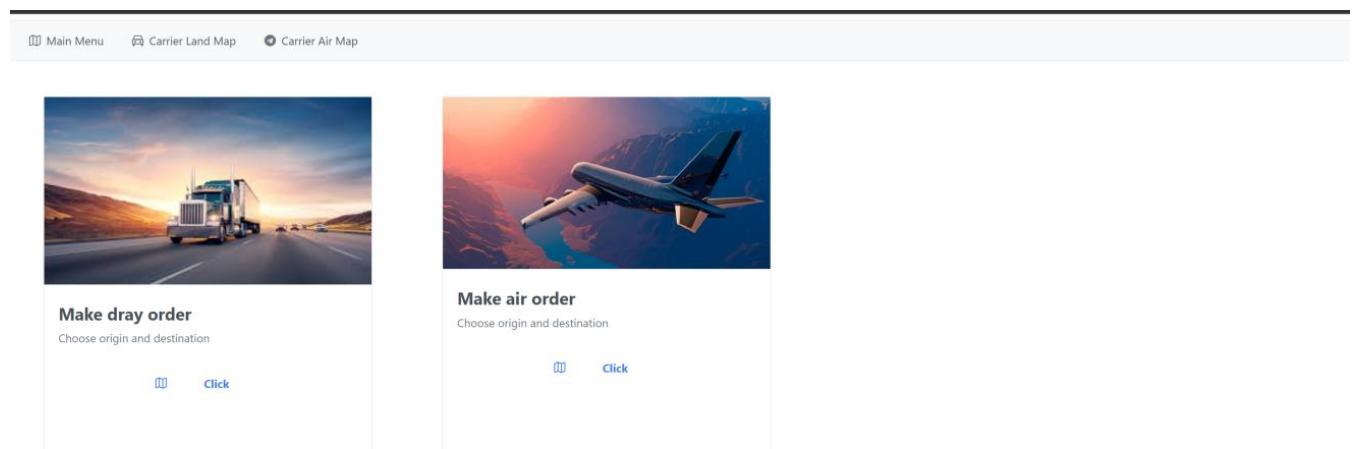


Рисунок 5.1 – Тип доставки

При натисканні кнопки Click, з'явиться карта для вибору пункту відправлення та заповнення інформації (а саме компанія доставки, кур'єр з його інформацією доставки (ім'я, тип транспорту, витрата топливу), дата доставки, вага, об 'єм, ім'я продукту, координати та адреса) по заказу і вибору місця доставки (рисунки 5.2, 5.3).

При натискані другий раз видається інформація об пункті доставки (ім'я продукту, адреса, відстань від відправлення до доставки, час доставки, дата доставки). Форму наведено на рисунку 5.4.

Після додавання пункту доставки, строюється шлях доставки (рисунок 5.5).

Далі є можливість зберегти замовлення. Перейти до мапи з перевізниками, вибрати компанію та перевізника для отримання всіх його замовлень на карті (рисунок 5.6).

Кожний замовлення має окремий колір, як показано на рисунку 5.7.

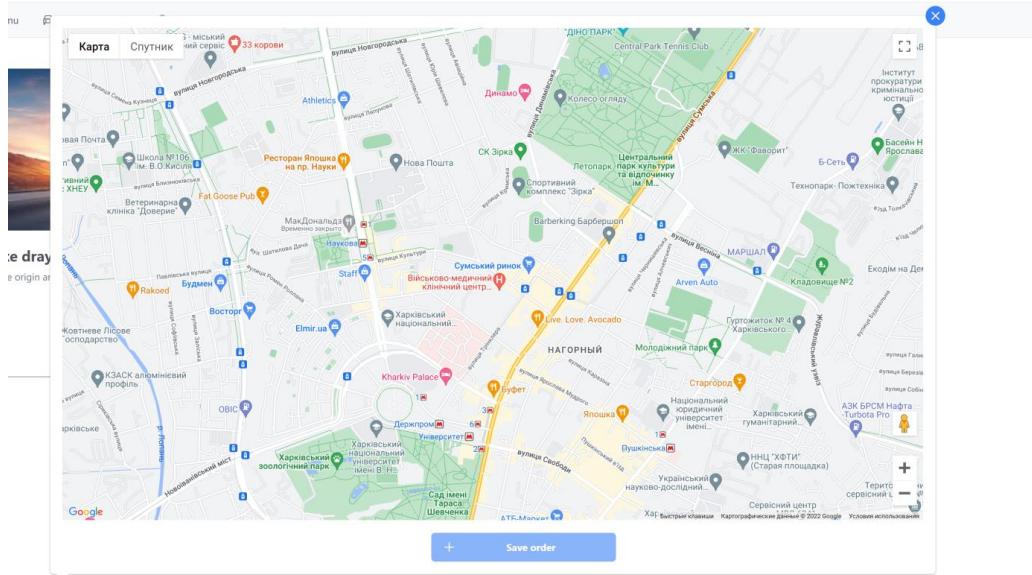


Рисунок 5.2 – Мапа доставки

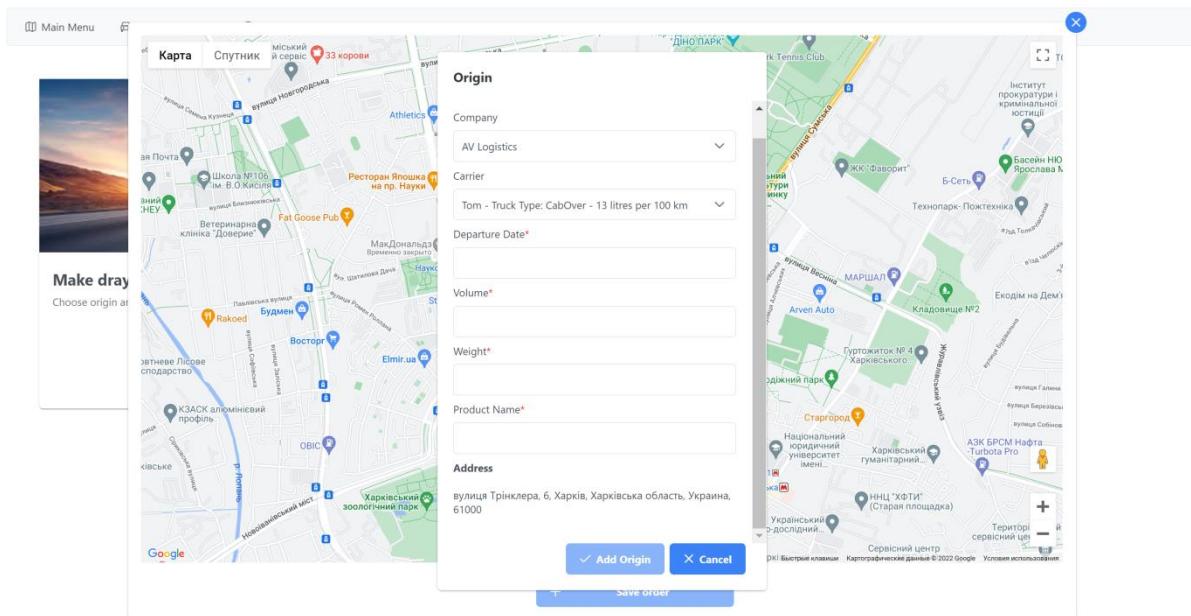


Рисунок 5.3 – Форма відправлення

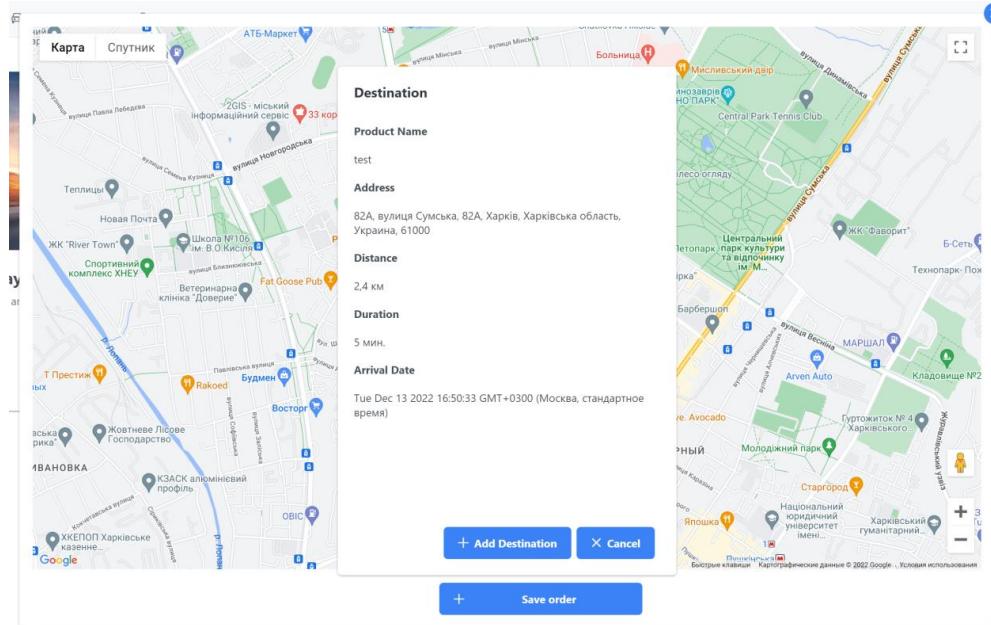


Рисунок 5.4 – Приклад пункту доставки

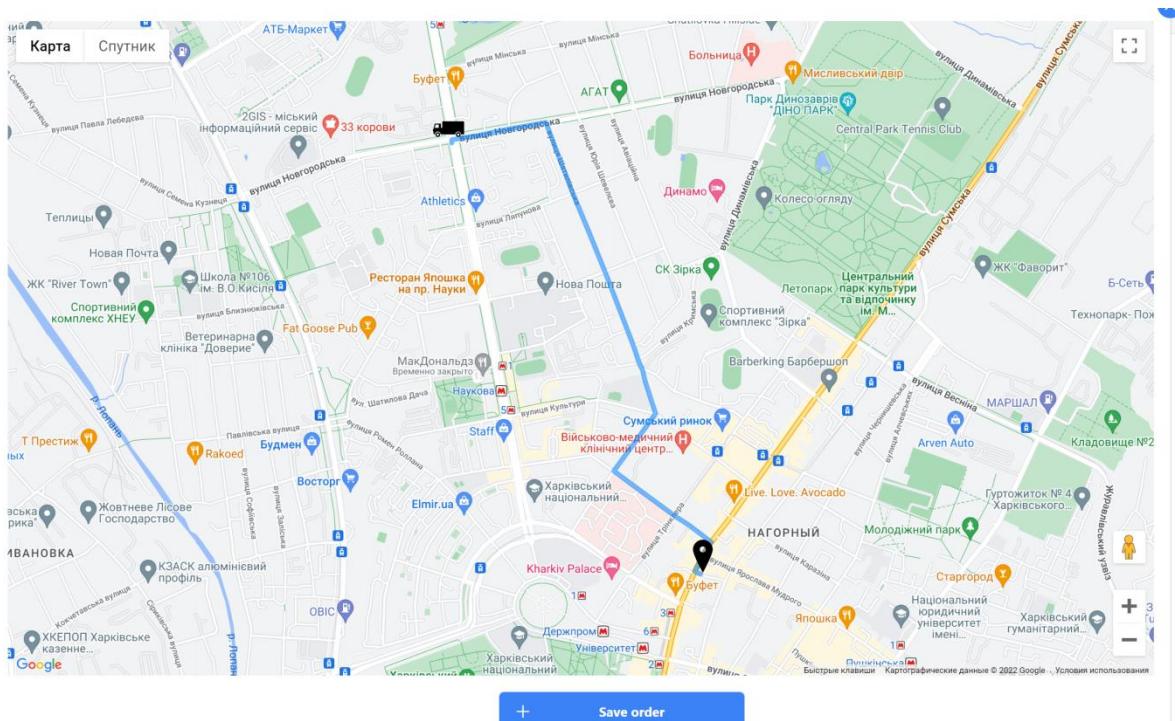


Рисунок 5.5 – Шлях доставки

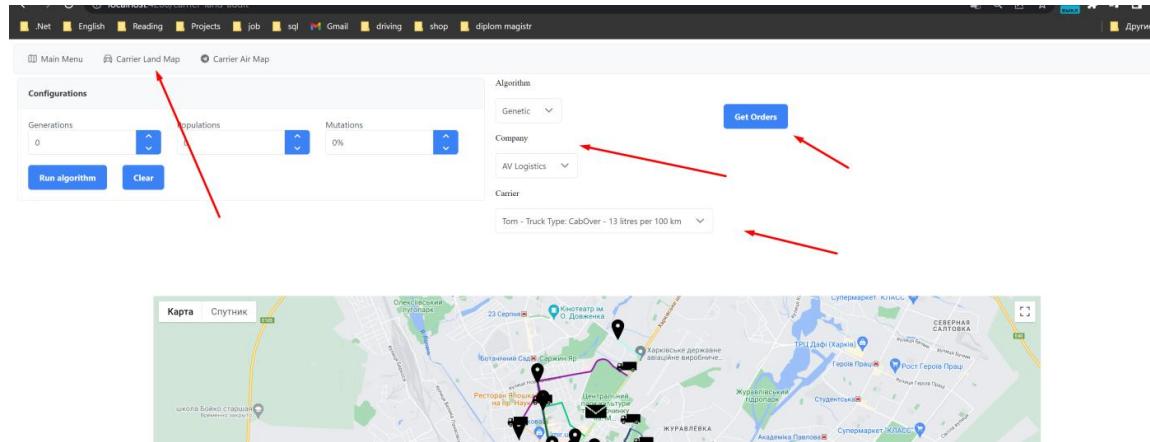


Рисунок 5.6 – Мапа перевізника

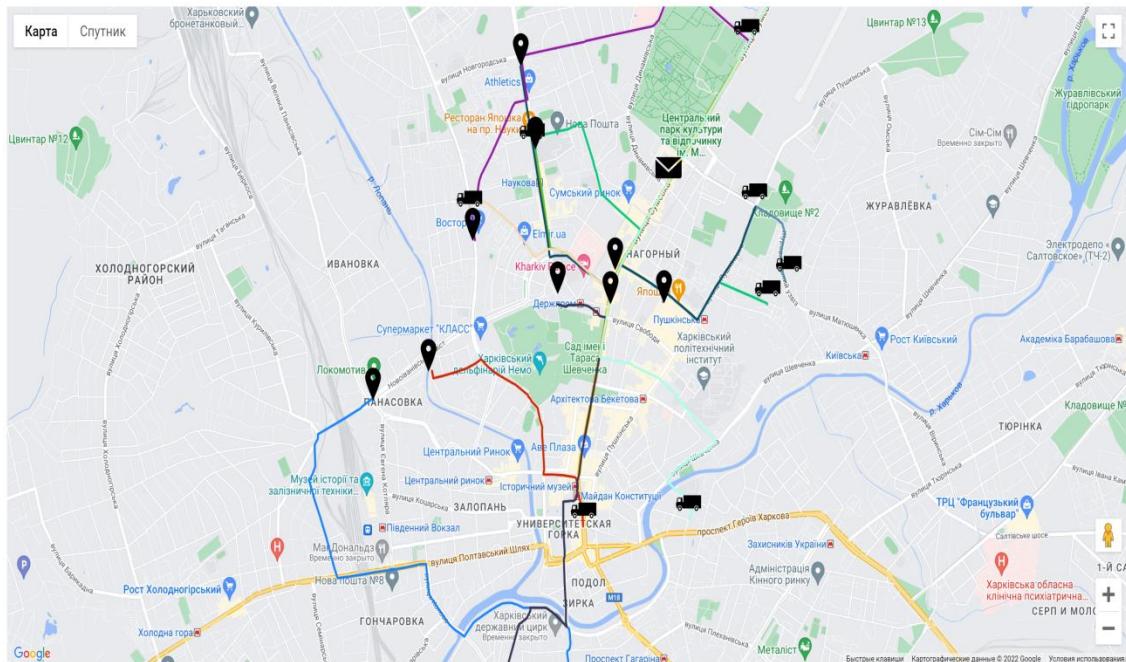


Рисунок 5.7 – Мапа с заказами перевізника

Налаштування генетичного алгоритму, а саме «Кількість поколінь, популяцій, % мутації» наведено на рисунку 5.8.

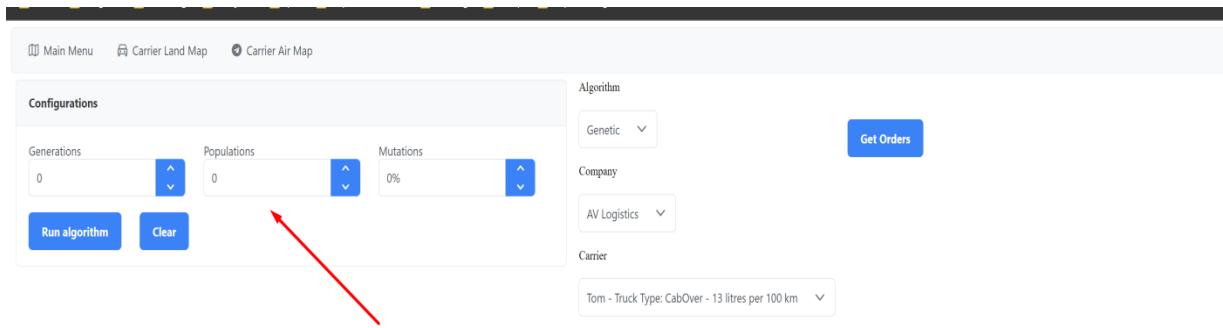


Рисунок 5.8 – Налаштування генетичного алгоритму

5.2 Висновок за розділом

Створили детальний опис роботи програми та її інтерфейсу.

6 ЕКОНОМІЧНА ЧАСТИНА

6.1 Мета розділу

Мета даного розділу – економічне обґрунтування розробки веб-додатку для оптимізації маршрутів вантажоперевезень з застосуванням генетичних алгоритмів. Для досягнення поставленої мети необхідно виконати ряд наступних завдань:

- 1) розрахувати собівартість програмного продукту;
- 2) визначити тривалість виконання роботи;
- 3) знайти оптимальний організаційно-технологічний варіант виконання роботи.

6.2 Визначення тривалості та розрахунок собівартості виконання проекту (перший варіант)

Для розробки веб-додатку для оптимізації маршрутів вантажоперевезень з застосуванням генетичних алгоритмів потрібна команда наступних спеціалістів: full-stack розробник, який відповідає за проектування, розробку та супровід додатку та тестувальник який займеться покриттями тестами.

Тривалість робочого місяця становить 22 дні, а робочого дня – 8 годин. Склад виконавців та розмір заробітної плати кожного наведено в таблиці 6.1.

Фонд основної заробітної плати обчислюється за формулою:

$$Z_{oc} = Z_{cd} \cdot T, \quad (6.1)$$

де Z_{oc} – основна зарплата, грн.;

T – трудовитрати робітника, людино/дні;

Z_{cd} – середньоденна заробітна плата робітника, грн. Вона визначається за формулою 6.2.

$$Z_{cd} = \frac{Z_{mic}}{\Phi}, \quad (6.2)$$

де Z_{mic} – місячна зарплата грн;

Φ – кількість робочих днів в місяць.

У full-stack розробника $Z_{mic} = 50000$ грн., у тестувальника проекту $Z_{mic} = 36000$ грн. $\Phi = 22$.

За наведеною формулою (6.2) середня денна заробітна плата тестувальника $Z_{\text{сд}} = 1636$ грн., у full-stack розробника $Z_{\text{сд}} = 2272$ грн.

Таблиця 6.1 – Склад та зарплата виконавців роботи

Посада	Посадові оклади, грн.	
	місячні	денні
Тестувальник	36 000	1636
Full-stack розробник	50 000	2272
Разом	86 000	3908

Розрахунок трудомісткості робіт представлений в таблиці 6.2.

Таблиця 6.2 – Розрахунок трудомісткості робіт

Вид роботи	Тривалість, дні	Трудомісткість, людина/дні	
		Тестувальник	Full-stack розробник
Складання завдання	3	3	3
Погодження і затвердження завдання	1	1	1
Розробка архітектури проекту	2	0	2
Розробка клієнтської частини	6	0	6
Розробка серверної частини	8	0	8
Тестування додатку розробником	2	0	2
Тестування тестувальником	4	4	0
Огляд вимог та зробленої програми	6	3	3
Розміщення на хостинг	1	0	1
Всього	33	11	26

Згідно (6.1), (6.2) розрахуємо фонд основної заробітної плати:

$$Z_{\text{ос}} = 1636 * 11 + 2272 * 26 = 17996 + 59072 = 77068 \text{ (грн).}$$

Додаткова заробітна плата становить 15% від основної заробітної плати.

$$Z_{\text{дод}} = 0,15 * 77068 = 11560 \text{ (грн).}$$

Фонд заробітної плати складає: $Z_{\text{зар}} = Z_{\text{ос}} + Z_{\text{дод}} = 77068 + 11560 = 88628$ (грн).

Розрахуємо єдиний соціальний внесок. На даний момент він становить 22% від загального фонду заробітної плати. Тоді: $\text{ЕСВ} = Z_{\text{зар}} * 0,22 = 88628 * 0,22 = 19498$ (грн). Необхідно розрахувати вартість матеріалів та послуг, необхідних для розробки веб-додатку. Результати розрахунків наведені в таблиці 6.3.

Таблиця 6.3 – Витрати на матеріали та послуги

Матеріал	Кількість, од.	Ціна за одиницю, грн.	Сума, грн.	Призначення
Jira Confluence	1 міс.	4 064	8128	Комунікація команди та погодження роботи
SSD 250 Гб	1	2200	2200	Збереження копій
A92	200	51,26	10252	Паливо для генератора
Послуги Інтернет	1 міс.	200	200	Пошук інформації, комунікація та інш.
Доменне ім'я	1 рік	350	350	Розміщення додатку
Хостинг	1 рік	1800	1800	Розміщення додатку
Всього			22930	

В результаті вартість матеріалів та послуг, необхідних для розробки веб-додатку, складає 22930 грн. Окрім матеріалів та послуг, для розробки веб-додатку також потрібне обладнання, яке представлене в таблиці 6.4.

Таблиця 6.4 – Витрати на обладнання та устаткування

Обладнання	Кількість, од.	Ціна за одиницю, грн.	Сума, грн.
Ноутбук	2	36300	72600
Стіл	2	5000	10000
Стілець	2	1700	3400
Комп'ютерна миша	2	1200	2400
Принтер	1	2300	2300
Генератор бензиновий	1	30 000	30 000
Роутер WiFi	1	400	400
Всього			121100

Наведемо формули для розрахунку величини амортизації:

1) враховуючи первісну вартість обладнання і річну норму амортизації можливо розрахувати річну суму амортизаційних відрахувань за формулою:

$$A_{j\text{ pič}} = \frac{C_{nj} \times H_{am}}{100\%}, \quad (6.3)$$

де $A_{j\text{ pič}}$ – річні амортизаційні відрахування по j-тому виду обладнання, грн.;

C_{nj} – первинна вартість обладнання даного виду, грн.;

H_{am} – річна норма амортизації, %.

визначається величина амортизаційних відрахувань у розрахунку для однієї години роботи обладнання даного виду:

$$A_{j\text{ год}} = \frac{A_{j\text{ pič}}}{T_j}, \quad (6.4)$$

де $A_{j\text{ год}}$ – величина амортизаційного обладнання j-того виду за одну годину використання, грн.;

T_j – річний фонд роботи даного обладнання, год.

Річний фонд роботи в рамках військового положення в 2022 р. при 40-годинному робочому тижні складає 2054 год.

2) залежно від часу використання обладнання, в процесі виготовлення продукту розраховується величина амортизаційних відрахувань, пов'язаних з виробництвом продукту:

$$A_{j\Pi} = A_{j\text{ год}} \times t_{j\Pi}, \quad (6.5)$$

де $A_{j\Pi}$ – амортизація обладнання j-того виду при виготовленні програмного продукту, грн.;

$t_{j\Pi}$ – час використання обладнання j-того виду при виготовленні даного програмного продукту, год.

Якщо використовується різне обладнання, то величина амортизації повинна бути визначена шляхом підсумовування амортизаційних відрахувань для окремих видів обладнання:

$$A_{\Pi} = \sum_{j=1}^m A_j \Pi , \quad (6.6)$$

де A_{Π} – сума амортизації, грн.;

m – кількість видів устаткування.

Результати розрахунків наведено у таблиці 6.5.

Таблиця 6.5 – Амортизаційні відрахування обладнання

Найменування обладнання	Первісна вартість, грн	Річна норма амортизації, %	Час використання обладнання, год.	Амортизаційні відрахування, грн		
				за рік	за годину використання обладнання	за період виконання роботи
Ноутбук	36300	25	296	9075	4,4	1302
Генератор бензиновий	30000	25	296	7500	3,7	1095
Стіл	5000	15	296	750	0,37	110
Стілець	1700	15	296	255	0,12	36
Роутер Wifi	400	10	296	40	0,02	6
Комп'ютерна миша	1200	20	296	240	0,12	36
Принтер	2300	10	4	230	0,12	0.5
Всього						2550

Згідно з даними таблиці 6.5, амортизація за період виконання проекту становить 2550 гривень.

Також потрібно підрахувати витрати на електроенергію. Розрахунки наведені в таблиці 6.6.

Таблиця 6.6 – Розрахунок витрат на електроенергію

Найменування	Потужність, кВт/год.	Час використання, год.	Вартість кВт/год., грн	Витрати, грн
Ноутбук	0,065	296	1,68	32
Роутер	0,0016	296	1,68	1
Принтер	0,03	4	1,68	0,2
Освітлення	0,035	296	1,68	17,4
Генератор	0,77	296	1,68	383
Всього				434

Після цього необхідно розрахувати собівартість розробленого продукту для первого варіанту розробки, підрахунок собівартості представлено в таблиці 6.7.

Таблиця 6.7 – Собівартість розробки платформи (перший варіант)

№	Статті	Сума, грн.	Примітка
1	Основна заробітна плата (ОЗП)	77068	Формули 6.1 и 6.2
2	Додаткова заробітна плата (ДЗП)	11560	15% від ОЗП
3	Єдиний соціальний внесок	19498	22% (ОЗП +ДЗП)
4	Матеріали і послуги	22930	З табл. 6.3
5	Амортизація	2550	З табл. 6.5
6	Витрати на обладнання та устаткування	121100	З табл. 6.5
7	Витрати на електроенергію	434	З табл. 6.6
8	Собівартість (C)	255140	п.1+п.2+п.3+...+п.6

6.3 Визначення тривалості та розрахунок собівартості виконання проекту (другий варіант)

Для розрахунків другого варіанта використаємо команду наступних спеціалістів: Senior Full-stack розробник. У Senior Full-stack розробника $Z_{\text{mic}} = 80000$. $\Phi = 22$. За формулою (6.2) середня денна заробітна плата у Full-stack $Z_{\text{cd}} = 3636$ грн., Склад виконавців та розмір заробітної плати кожного наведено в таблиці 6.8.

Таблиця 6.8 – Склад та зарплата виконавців роботи

Посада	Посадові оклади, грн.	
	місячні	дennі
Senior Full-Stack розробник	80000	3636
Разом	80000	3636

Розрахунок трудомісткості робіт представлений в таблиці 6.9.

Таблиця 6.9 – Розрахунок трудомісткості робіт для Senior Full-Stack розробника

Вид роботи	Трудомісткість, людина/дні (тривалість, дні)
Складання завдання	3
Погодження і затвердження завдання	1
Розробка архітектури проекту	4
Розробка клієнтської частини	6
Розробка серверної частини	8
Тестування платформи розробником	1
Огляд вимог та зробленої програми	3
Розміщення на хостинг	1
Всього	27

Згідно (6.1), (6.2) розрахуємо фонд основної заробітної плати:

$$Z_{oc} = 3636 * 27 = 98172 \text{ (грн)}.$$

Додаткова заробітна плата становить 15% від основної заробітної плати.

$$Z_{\text{дод}} = 0,15 * 98172 = 14726 \text{ (грн)}.$$

Фонд заробітної плати складає:

$$Z_{\text{заг}} = Z_{oc} + Z_{\text{дод}} = 98172 + 14726 = 112898 \text{ (грн)}.$$

Єдиний соціальний внесок становить:

$$\text{ЕСВ} = З_{\text{зар}} * 0,22 = 112898 * 0,22 = 24838 \text{ (грн).}$$

Також потрібно розрахувати вартість матеріалів та послуг. Результати розрахунків представлена в таблиці 6.10.

Таблиця 6.10 – Витрати на матеріали та послуги

Матеріал	Кількість, од.	Ціна за одиницю, грн.	Сума, грн.	Призначення
Jira Confluence	1 міс.	4 064	8128	Комунікація команди та
SSD 250 Гб	1	2200	2200	Збереження копій
A92	200	51,26	10252	Паливо для генератора
Послуги Інтернет	1 міс.	200	200	Пошук інформації, комунікація та інш.
Доменне ім'я	1 рік	350	350	Розміщення додатку
Хостинг	1 рік	1800	1800	Розміщення додатку
Всього			22930	

В результаті вартість матеріалів та послуг, необхідних для розробки платформи, складає 76900 грн.

Також для розробки веб-додатку потрібне обладнання, яке представлене в таблиці 6.11.

Таблиця 6.11 – Витрати на обладнання та устаткування

Обладнання	Кількість, од.	Ціна за одиницю, грн	Сума, грн
Ноутбук	1	36300	36300
Стіл	1	5000	5000
Стілець	1	1700	1700
Комп'ютерна миша	1	1200	1200
Принтер	1	2300	2300
Генератор бензиновий	1	30000	30000
Роутер Wifi	1	400	400
Всього	7		76900

Використовуючи формули 6.3 – 6.6 та дані, представлені у таблиці 6.11, розрахуємо амортизаційні відрахування обладнання та перенесемо отримані дані до таблиці 6.12.

Згідно з даними, які були розраховані та внесені до таблиці 6.12, амортизація за період виконання проекту складає 1912 гривень.

Далі необхідно підрахувати витрати на електроенергію та собівартість продукту для другого варіанту розробки. Розрахунки наведені в таблицях 6.13 та 6.14.

Таблиця 6.12 – Амортизаційні відрахування обладнання

Найменування обладнання	Первісна вартість, грн	Річна норма амортизації, %	Час використання обладнання, год.	Амортизаційні віdraхування, грн.		
				За рік	За годину використання обладнання	За період виконання роботи
Ноутбук	36300	25	216	9075	4,4	950
Генератор бензиновий	30000	25	216	7500	3,7	800
Стіл	5000	15	216	750	0,37	80
Стілець	1700	15	216	255	0,12	26
Роутер Wifi	400	10	216	40	0,02	4
Комп'ютерна миша	1200	20	216	240	0,12	26
Принтер	2300	10	4	230	0,12	26
Всього						1912

Таблиця 6.13 – Розрахунок витрат на електроенергію

Найменування	Потужність,	Час використання,	Вартість	Витрати,
Ноутбук	0,065	216	1,68	24
Роутер	0,0016	216	1,68	0,6
Принтер	0,03	4	1,68	0,2
Освітлення	0,035	216	1,68	13
Генератор	0,77	216	1,68	279
Всього				317

Таблиця 6.14 – Собівартість розробки веб-додатку (другий варіант)

№	Статті	Сума, грн.	Примітка
1	Основна заробітна плата (ОЗП)	98172	Формули 6.1 и 6.2
2	Додаткова заробітна плата (ДЗП)	14726	15% від ОЗП
3	Єдиний соціальний внесок	24838	22% (ОЗП + ДЗП)
4	Матеріали і послуги	22930	З табл. 6.10
5	Амортизація	1912	З табл. 6.12
6	Витрати на обладнання та устаткування	76900	З табл. 6.11
7	Витрати на електроенергію	317	З табл. 6.13
8	Собівартість (С)	239795	п.1+п.2+п.3+...+п.6

6.4 Висновки за розділом

Було приведено економічне обґрунтування розробки веб-додатку. Для першого варіанту було обрано full-stack розробник та тестувальника. Для другого обрано одного Senior full-stack розробника. Для оцінки вартості проекту враховувались різні витрати. Собівартість розробки платформи у першому варіанті становить – 258266 грн, у другому – 239795 грн, тривалість відповідно 39 та 27 днів.

На основі отриманих результатів можна зробити висновок, що для розробки краще залучати працівників, які спеціалізуються на чомусь одному та які більш кваліфіковані, тому що:

- 1) швидкість розробки значно вища. Тривалість розробки для другого варіанту менша на 12 днів;
- 2) в наслідок швидшої роботи таких спеціалістів загальна ціна розробки платформи значно менша. Різниця становить 18471 грн. Таким чином, можна прийти до висновку, що розробкою проекту повинні займатися розробник з другого варіанту.

ВИСНОВКИ

Результатом кваліфікаційної роботи є математична модель розв'язання задачі комівояжера за допомогою генетичного алгоритму та її програмна реалізація. Були сформовані вхідні дані задачі для отримання результатів. Були розглянуті оператори кросоверу для генетичного алгоритму для роботи з генотипом у вигляді комбінаторної перестановки.

Розроблений програмний продукт у вигляді веб-додатку, який складається клієнтської та серверної частини. Даний програмний продукт здатний оптимізувати задачу маршрути доставки вантажів допомогою генетичного алгоритму. Додаток має зручний інтерфейс, який дозволяє відображати зформований маршрут на мапі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Задача комівояжера [Електронний ресурс] – режим доступу: https://uk.wikipedia.org/wiki/Задача_комівояжера.
2. Генетичний алгоритм [Електронний ресурс] – режим доступу: https://uk.wikipedia.org/wiki/Генетичний_алгоритм.
3. Моторнюк А. І., Лаврів А. М. Задача комівояжера // Математика, що нас оточує: минуле, сучасне, майбутнє. – 2022. – С. 76 – 78.
4. Ємець О., Чілікіна Т. Задача комівояжера: нелінійна модель на представленнях та метод гілок та меж // Cherkasy University Bulletin: Applied Mathematics. Informatics. – 2015. – Т. 351. – №. 18.
5. Базилевич Р., Кутельмах Р. Дослідження ефективності існуючих алгоритмів для розв'язання задачі комівояжера // Вісник НУ «Львівська політехніка». – 2009. – №. 650. – С. 235-245.
6. Ємець О. О., Парфьонова Т. О. Транспортні задачі комбінаторного типу: властивості, розв'язування, узагальнення. – Полтава, ПУЕТ, 2011 – 174 с.
7. Goldberg D. E. Genetic algorithms in search, optimization, and machine learning. Addison Wesley. – 1989. – №. 102. – P. 36.
8. Miller B. L. et al. Genetic algorithms, tournament selection, and the effects of noise // Complex systems. – 1995. – Т. 9. – №. 3. – С. 193-212.
9. Гулаєва Н. М., Шило В. П. Генетичні алгоритми як обчислювальні методи скінченновимірної оптимізації // Кібернетика та комп’ютерні технології. – 2021, № 3. – С. 5 – 14. DOI: <https://doi.org/10.34229/2707-451X.21.3.1>
10. Погорілій С. Д., Білоус Р. В. Генетичний алгоритм розв'язання задачі маршрутизації в мережах. // Проблеми програмування. 2010. № 2–3. Спеціальний випуск. – С. 171 – 177.
11. Мужиков І. О. Реалізація паралельного генетичного алгоритму. – 2017. [Електронний ресурс] – режим доступу: <http://eztuir.ztu.edu.ua/bitstream/handle/123456789/6672/137.pdf>.
12. Рекік А. Програмна система розв'язування транспортної задачі з модифікованими крайовими умовами // Науковий вісник НЛТУ України. – 2012. – Т. 22. – №. 11. – С. 355-362.
13. Литвин, В. В., Угрин, Д. І., Іллюк, О. Д., Білоус, С. В., Рибчақ, З. Л. Система оптимізації маршрутів туризму на основі модифікації генетичного та мурашиного алгоритмів // Вісник Національного університету «Львівська політехніка». Серія: Інформаційні системи та мережі. – 2017. – №. 872. – С. 210-219.
14. Ковальова К. О., Місюра Є. Ю. Програмна реалізація задачі про розподіл транспортних засобів з фіксованими доплатами // Бізнес Информ. – 2018. – №. 5 (484). – С. 167-173.

15. Yaroshko S., Yaroshko S. Побудова багатопотокових програм засобами платформи .NET // Вісник Львівського університету. Серія прикладна математика та інформатика. – № 27(2019): Випуск 27. DOI: <http://dx.doi.org/10.30970/vam.2019.27.10137>.
16. Troelsen A., Japikse P. Introducing C# and. NET 6 //Pro C# 10 with. NET 6. – Apress, Berkeley, CA, 2022. – P. 3 – 26. DOI: 10.1007/978-1-4842-7869-7_1.
17. Build any app with .NET [Електронний ресурс] – режим доступу: <https://dotnet.microsoft.com>

ДОДАТОК А ЛИСТИНГ ПРОГРАМИ

Генетичний алгоритм на с#

```

public class Point
{
    public int Number { get; set; }
    public double X { get; set; }
    public double Y { get; set; }
}

public class DictionaryArrayKeyComparer : IEqualityComparer<int[]>
{
    public bool Equals(int[]? x, int[]? y)
    {
        if (x.Length != y.Length)
        {
            return false;
        }
        for (int i = 0; i < x.Length; i++)
        {
            if (x[i] != y[i])
            {
                return false;
            }
        }
        return true;
    }

    public int GetHashCode([DisallowNull] int[] obj)
    {
        int result = 17;
        for (int i = 0; i < obj.Length; i++)
        {
            unchecked
            {
                result = result * 23 + obj[i];
            }
        }
        return result;
    }
}

using GeneticAlgorithmConsole.Models;
using GeneticAlgorithmConsole.Utils;

var path = Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName;
var text = File.ReadAllText(path + "/Coordinates.txt");
var textCoordinates = text.Split(',');
var points = new List<Point>();
var pointsLines = new Dictionary<(int numberFrom, int numberTo), double>();
var pathPopulations = new Dictionary<int[], double>(new DictionaryArrayKeyComparer());
var r = new Random();

```

```

foreach (var cord in textCoordinates)
{
    points.Add(new Point
    {
        X = double.Parse(cord[1].ToString()),
        Y = double.Parse(cord[3].ToString())
    });
}

for (int i = 1; i < points.Count+1; i++)
{
    for (int j = 1; j < points.Count+1; j++)
    {
        var len = GetLineLength(points[i-1], points[j-1]);

        pointsLines.Add((i, j), len);
    }
}

const bool inversion = true;
const int population = 10;
const int generations = 5;
int mutations = 20;

var populations = new Dictionary<int[], int>(new DictionaryArrayKeyComparer());
for (int i = 0; i < population; i++)
{
    var combinations = new List<int>();
    for (int j = 0; j < points.Count; j++)
    {
        var num = r.Next(1, points.Count + 1);
        if (!combinations.Contains(num) && combinations.Count < points.Count)
        {
            combinations.Add(num);
        }
        else
        {
            while (combinations.Contains(num) && combinations.Count < points.Count)
            {
                num = r.Next(1, points.Count + 1);
            }
            combinations.Add(num);
        }
    }
    if (!populations.TryAdd(combinations.ToArray(), i))
    {
        i--;
    }
}

var arraysPopulations = populations.Keys.ToList();

```

```

for (int i=0; i < arraysPopulations.Count; i++)
{
    var sum = 0.0;

    for (int j =0; j < arraysPopulations[i].Length-1; j++)
    {
        var pathNum = pointsLines[(arraysPopulations[i][j],
arraysPopulations[i][j+1])];
        sum += pathNum;
    }

    pathPopulations.Add(arraysPopulations[i], sum);
}

pathPopulations = pathPopulations.OrderBy(x => x.Value).ToDictionary(x => x.Key, y =>
y.Value);
var copyPopulationsResult = pathPopulations;

foreach (var p in pathPopulations)
{
    Console.Write("[");
    foreach (var e in p.Key)
    {
        Console.Write(${e},");
    }
    Console.Write("]");
    Console.WriteLine("Sum: " + p.Value);
}

for (int g = 0; g < generations; g++)
{
    Console.WriteLine($"Generation ${g}");
    var populatedIndexes = new List<int?>();
    for (int i = 0; populatedIndexes.Count != points.Count; i++)
    {
        var indexFirstParent = r.Next(0, 10);
        var indexSecondParent = r.Next(0, 10);

        while (populatedIndexes.Contains(indexFirstParent))
        {
            indexFirstParent = r.Next(0, 10);
        }
        populatedIndexes.Add(indexFirstParent);

        while (populatedIndexes.Contains(indexSecondParent))
        {
            indexSecondParent = r.Next(0, 10);
        }
        populatedIndexes.Add(indexSecondParent);

        var firstParent = copyPopulationsResult.ElementAt(indexFirstParent).Key;
        var secondParent = copyPopulationsResult.ElementAt(indexSecondParent).Key;
        copyPopulationsResult.Remove(firstParent);
        copyPopulationsResult.Remove(secondParent);
    }
}

```

```

        var crossRes = Crossing(firstParent.ToList(), secondParent.ToList());
        double firstParentSum = 0.0;
        double secondParentSum = 0.0;

        for (int c = 0; c < crossRes.first.Count - 2; c++)
        {
            var len = pointsLines[(crossRes.first[c], crossRes.first[c + 1])];
            firstParentSum += len;
        }

        for (int c = 0; c < crossRes.second.Count - 2; c++)
        {
            var len = pointsLines[(crossRes.second[c], crossRes.second[c + 1])];
            secondParentSum += len;
        }

        copyPopulationsResult.Remove(firstParent);
        copyPopulationsResult.Remove(secondParent);
        copyPopulationsResult.Add(crossRes.first.ToArray(), firstParentSum);
        copyPopulationsResult.Add(crossRes.second.ToArray(), secondParentSum);
        copyPopulationsResult = copyPopulationsResult.OrderBy(x =>
x.Value).ToDictionary(x => x.Key, y => y.Value);
    }

    foreach (var c in copyPopulationsResult)
    {
        Console.Write($"[");
        foreach (var n in c.Key)
        {
            Console.Write($"{n},");
        }
        Console.Write($"]; sum:");
        Console.WriteLine(c.Value);
    }
}

(List<int> first, List<int> second) Crossing(List<int> first, List<int> second)
{
    var r = new Random();
    var resFirst = new List<int>();
    var resSecond = new List<int>();

    var pointBreak = r.Next(1, first.Count+1);

    var beforePointBreakFirstParent = first.Take(pointBreak).ToList();

    var beforePointBreakSecondParent = second.Take(pointBreak).ToList();
    var afterPointBreakSecondParent = second.Skip(pointBreak).ToList();
    var forFirst1 = afterPointBreakSecondParent.Except(beforePointBreakFirstParent);
    var afterPointBreakFirstParent = first.Skip(pointBreak).ToList();
    var forFirst2 = afterPointBreakFirstParent.Except(afterPointBreakSecondParent);
    var forSecond1 = afterPointBreakFirstParent.Except(beforePointBreakSecondParent);
}

```

```

var forSecond2 = afterPointBreakSecondParent.Except(forSecond1);

// take first part before point break from first and after from second
resFirst.AddRange(beforePointBreakFirstParent);
resFirst.AddRange(forFirst1);
if (resFirst.Count != points.Count)
{
    resFirst.AddRange(forFirst2);
}

resSecond.AddRange(beforePointBreakSecondParent);
resSecond.AddRange(forSecond1);
if (resSecond.Count != points.Count)
{
    resSecond.AddRange(forSecond2);
}

var mutation = r.Next(0, 101);
mutations = r.Next(0, 101);

if (mutations < mutation)
{
    Mutate(resFirst);
    Mutate(resSecond);
}

if (inversion)
{
    resFirst = Inversions(resFirst);
    resSecond = Inversions(resSecond);
}

return (resFirst, resSecond);
}

void Mutate(List<int> chromosome)
{
    int rFirst = r.Next(0, chromosome.Count);
    int rSecond = r.Next(0, chromosome.Count);
    while (rFirst == rSecond)
    {
        rSecond = r.Next(1, chromosome.Count);
    }
    var temp = chromosome[rFirst];
    chromosome[rFirst] = chromosome[rSecond];
    chromosome[rSecond] = temp;
}

List<int> Inversions(List<int> parent)
{
    var res = new List<int>();
    int pointBreak = r.Next(1, parent.Count);
    var invFirstPart = parent.Take(pointBreak);
    var invSecondPart = parent.Skip(pointBreak);
}

```

```

        res.AddRange(invSecondPart);
        res.AddRange(invFirstPart);
        return res;
    }

    double GetLineLength(Point start, Point end)
    {
        var res = Math.Sqrt(Math.Pow((end.X - start.X), 2) + Math.Pow((end.Y - start.Y), 2));
        return res;
    }

    void MoveItemsOnStep(ref int[] items, int step)
    {
        var arr = new int[items.Length];
        for (int i = 0; i < step; i++)
        {
            for (int s = 0; s < items.Length; s++)
            {
                if (s == items.Length - 1)
                {
                    arr[0] = items[s];
                }
                else
                {
                    arr[s+1] = items[s];
                }
            }
            items = arr;
        }
        items = arr;
    }

    public static class NarayanaTest
    {
        /// <summary>
        /// Возвращает true, если value_0 меньше value_1, иначе – false
        /// </summary>
        public static bool Less<T>(T value_0, T value_1) where T : System.IComparable
        {
            return value_0.CompareTo(value_1) < 0;
        }

        /// <summary>
        /// Возвращает true, если value_0 больше value_1, иначе – false
        /// </summary>
        public static bool Greater<T>(T value_0, T value_1) where T : System.IComparable
        {
            return value_0.CompareTo(value_1) > 0;
        }

        /// <summary>
        /// Инициализация последовательности
        /// </summary>
        public static void InitSequence(int[] sequence)
    }
}

```

```

{
    // Заполнение последовательности значениями 1, 2, 3...
    for (var i = sequence.Length; i > 0; --i)
        sequence[i - 1] = i;
}

/// <summary>
/// Вывод содержимого последовательности
/// </summary>
public static int[] OutputSequence(int[] sequence)
{
    var array = new int[sequence.Length];
    if (!(sequence == null) && (sequence.Length > 0))
    {
        array[0] = sequence[0];
        for (var i = 1; i < sequence.Length; ++i)
        {
            array[i] = sequence[i];
        }
    }
    return array;
}

public static class Narayana
{
    /// <summary>
    /// Функция, задающая отношение порядка для значений типа T: < либо >
    /// </summary>
    public delegate bool Predicate2<T>(T value_0, T value_1);

    /// <summary>
    /// Поиск очередной перестановки
    /// </summary>
    public static T[] NextPermutation<T>(T[] sequence, Predicate2<T> compare)
    {
        var arr = new T[sequence.Length];
        // Этап № 1
        var i = sequence.Length;
        do
        {
            if (i < 2)
                return arr; // Перебор закончен
            --i;
        } while (!compare(sequence[i - 1], sequence[i]));
        // Этап № 2
        var j = sequence.Length;
        while (i < j && !compare(sequence[i - 1], sequence[--j])) ;
        _SwapItems(sequence, i - 1, j);
        // Этап № 3
        j = sequence.Length;
        while (i < --j)
            _SwapItems(sequence, i++, j);
        arr = sequence;
    }
}

```

```
    return arr;
}

/// <summary>
/// Обмен значениями двух элементов последовательности
/// </summary>
public static void _SwapItems<T>(T[] sequence, int index_0, int index_1)
{
    var item = sequence[index_0];
    sequence[index_0] = sequence[index_1];
    sequence[index_1] = item;
}
}
```

ДОДАТОК Б ГРАФІЧНІ МАТЕРІАЛИ

Мета та об'єкт дослідження

Об'єкт дослідження: оптимізація маршрутів в транспортних задачах.

Предмет дослідження: методи та засоби розв'язання задачі комівояжера з застосуванням генетичних алгоритмів.

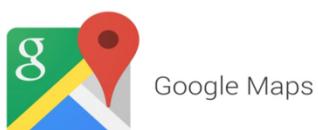
Мета роботи: оптимізація маршруту доставки товарів шляхом удосконалення математичного, алгоритмічного та програмного забезпечення розв'язання задачі комівояжера.



Актуальність

Більшість своїх сил кожен з нас витрачає на пошук найкращого або оптимального вирішення поставленого завдання. Існує багато NP-повних транспортних задач які можливо найти в реальному житті:

- Прискорення кур'єрської служби доставки їди для різних закладів.
- Доставка вантажів на далекі відстані (сушію, океаном, повітрям)
- Робота соціальних, міських та державних служб.
- Пошук знаходження шляху звичайному користувачеві картографічних сервісів.



Огляд існуючих рішень та їх проблеми

Існує немало різних алгоритмів для вирішення транспортної задачі комівояжера:

- Графові рішення алгоритмом Дейкстри
- Повні та випадкові перебори
- Алгоритм Флойда
- Методи потенціалів
- Жадібні алгоритми
- Імітації відпалу

Але існують також і невизначеності такі як:

- Час очікування клієнта або вантажу
- Пробки та аварії, платні дороги, досвід роботи та водіння кур'єру
- Природні та технічні явища, затратити палива



Переваги та недоліки ісуючих рішень

Недоліки звичайних рішень (повні та випадкові перебори):

- обмежені в кількості вхідних даних так як час збільшується експоненційно до них
- відсутність налаштувань

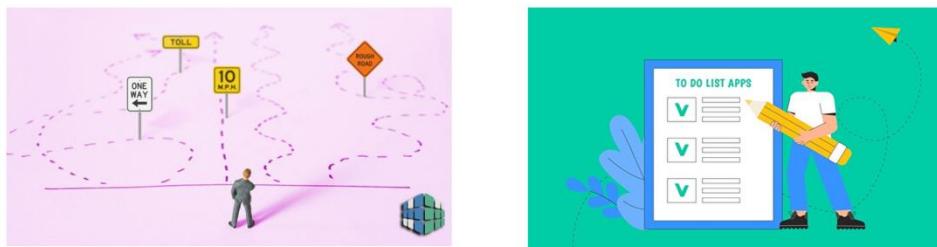
Переваги:

- + простота
- + для маленьких практичних завдань мінімальний час роботи та використання оперативної пам'яті
- + відсутність налаштувань

Для ефективного перебору на практиці я буду застосовувати евристичні алгоритми, які дають не точне, а наближене рішення і добре підходять для перебору великої кількості даних з налаштуванням алгоритму, що поступово поліпшується, наприклад **генетичний алгоритм**.

Постановка задач

1. Аналіз транспортних задач та огляд існуючих рішень.
2. Формалізація та математичне моделювання предметної області.
3. Застосування генетичних алгоритмів до розв'язання задачі комівояжера.
4. Проектування програмного забезпечення оптимізації вантажоперевезень.
5. Програмна реалізація.



Формалізація та математичне моделювання предметної області

Алгоритмічна складність симетричної задачі комівояжера з n пунктів

становить $\frac{(n - 1)!}{2}$ тому вже для 30 пунктів спрямування пошук шляху є складним завданням.

На маршрут кожного транспортного засобу накладається ряд обмежень.
Основні обмеження представлені формулами:

1. $\sum_{k \in V} \sum_{i \in N} X_{ij}^k = 1, \forall i \in C,$
2. $\sum_{i \in C} d_i \sum_{i \in N} X_{ij}^k \leq q, \forall k \in V$ d_i - вантажопідйомність, $i \in C$ - попит відповідного клієнта, C – безліч клієнтів.
3. $a_i \leq S_i^k \leq b_i, \forall i \in N, \forall k \in V$ $[a_i, b_i]$ - час прибуття
 S_i^k - час обслуговування

Застосування генетичних алгоритмів до розв'язання задачі комівояжера

Генотип – унікальні географічні координати (довгота та широта, адрес). В подальшому будуть кодуватимутися в алгоритму як порядок обходу маршруту або графа цифрою та числом.

Застосування генетичних алгоритмів до розв'язання задачі комівояжера

Хромосома - унікальна послідовність генів та сума пройденого шляху в кілометрах, сюди ж додамо витрачений час та паливо цього шляху, тип транспорту, сума доставки від початкового пункту і до нього же. (Рисунок 9.1)

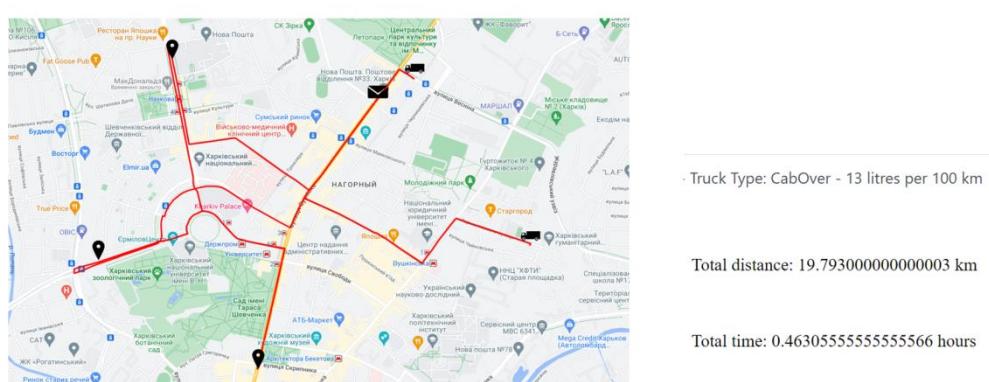
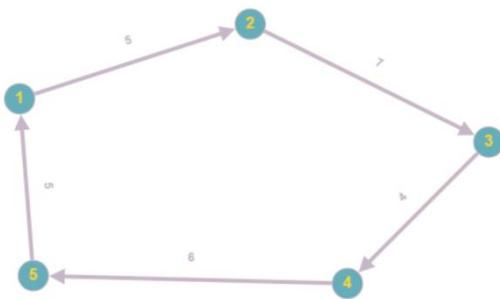


Рисунок 9.1 – Програмна особина

Популяції

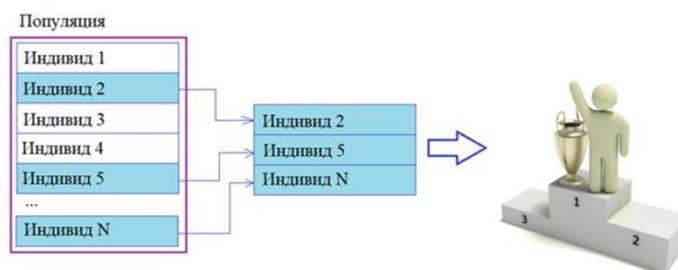
Якщо представити у порядковому вигляді – це набір унікальних цифр або чисел [1,2,3,4,5,1] з сумаю шляху = 27.

Популяцію же можливо назвати унікальний набір хромосом у порядковому вигляді та сумаю повного шляху.



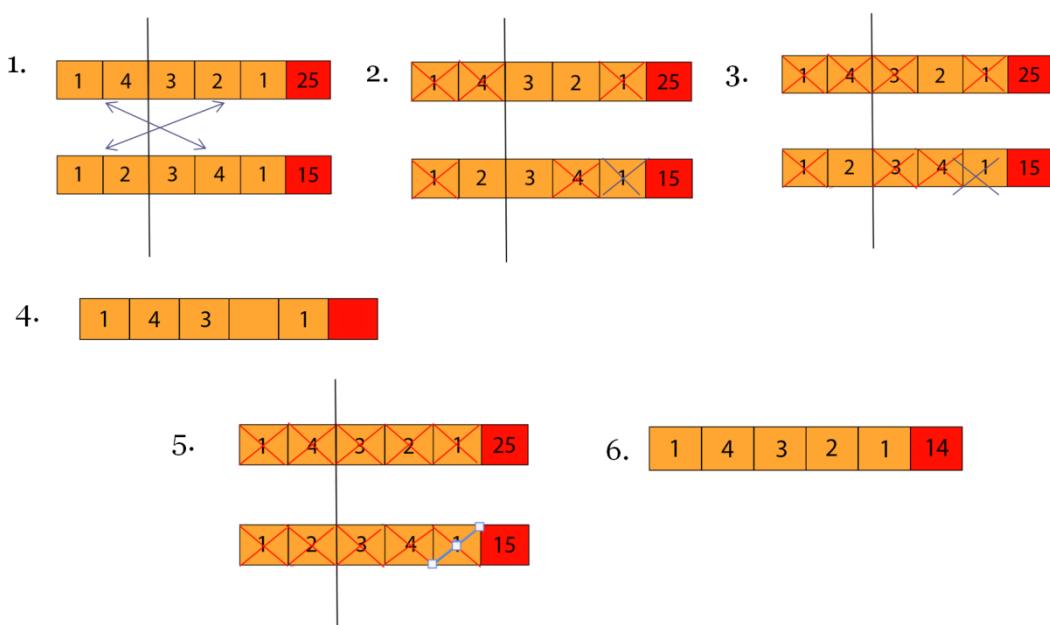
Турнірний відбір

Після формування популяції здійснюється, **турнірний відбір** - як може бути описаний наступним чином: з популяції, що містить N рядків, вибирається випадковим чином t рядків і кращий рядок записується в проміжний масив (між обрамими рядками проводиться турнір). Ця операція повторюється N разів.



Схрещування

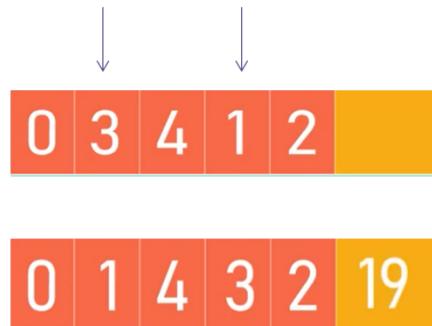
Далі йде схрещування де всі хромосоми мають порядковий вид, етапи схрещування:



Мутації

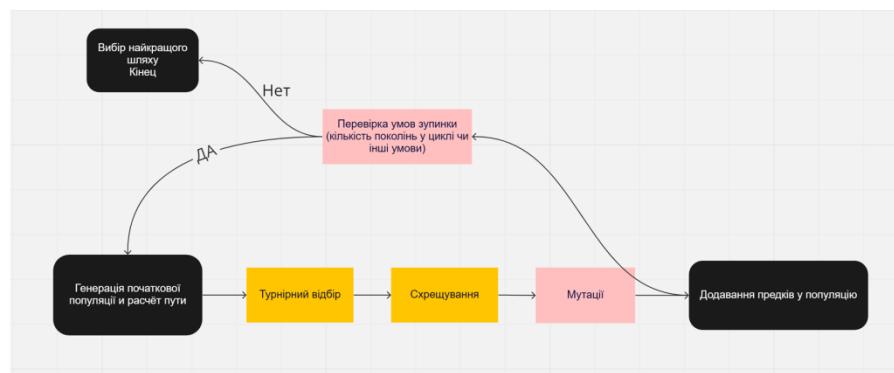
Абстрактний приклад мутації:

- Генеруємо число від 0 до 100.
- Якщо число менше від % мутації в налаштуваннях відбувається мутація.
- Вибирається 2 випадкові гени і вони змінюються місцями.



Застосування генетичних алгоритмів до розв'язання задачі комівояжера

- Додаємо до спільного масиву результатів.
- Повторюємо процес до критерії останови (кількості поколінь).
- Після завершення останнього покоління сортируємо масив за зростанням та знаходимо перший елемент як краще вирішення.



Архітектура додатку клієнтської сторони

Для розробки клієнтської частини веб-застосунку і візуалізації алгоритму застосовувався JS фреймворк Angular останньої версії який зосереджений на компонентному програмуванні та з бібліотекою NgPrime.

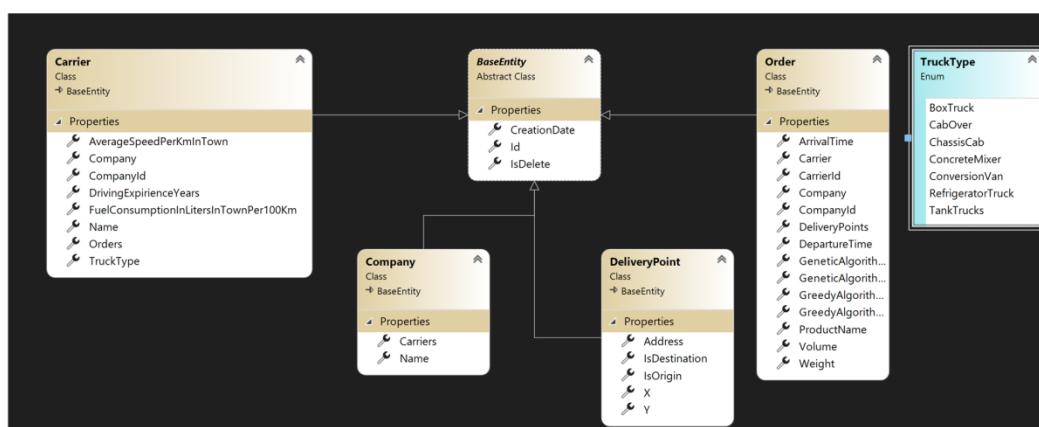
Архітектура поділена на наступні частини:

1. **Declarations/Widget Module.** Модулі із оголошеннями різних сущностей. Як приклад подібних модулів можна навести набори компонентів інтерфейсу користувача, директив, пайпів.
2. **Services Module.** Модулі Сервісів. Наприклад - HttpClientModule.
3. **Routing Module.** Модулі маршрутизації.
4. **Domain Feature Module.** Модулі, що реалізують ключові завдання програми.
5. **Core/Shared Module.** Соге-модуль це модуль для оголошення глобальних сервісів. Shared-модуль — це модуль, у якому оголошують компоненти спільного використання.

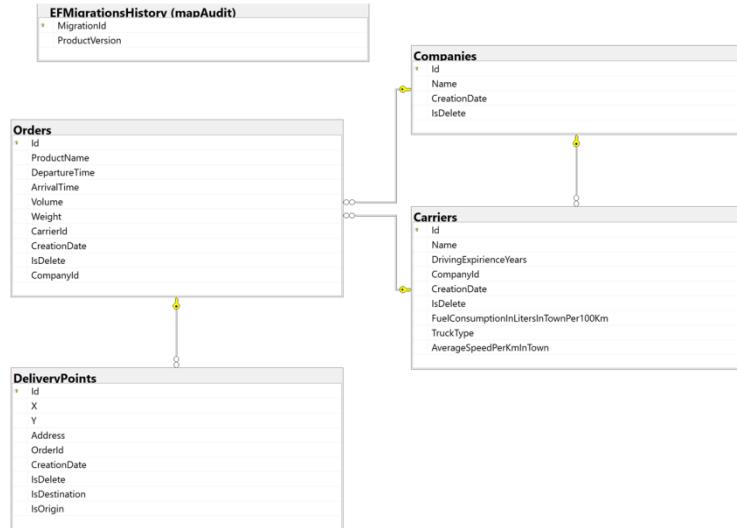
Архітектура додатку серверної сторони в Visual Studio 2022

Архітектура Onion серверної сторони розділена на:

1. **DataAccess layer.** Рівень доступа даних, яких містить логіку збереження даних за допомогою Entity Framework Core code first.

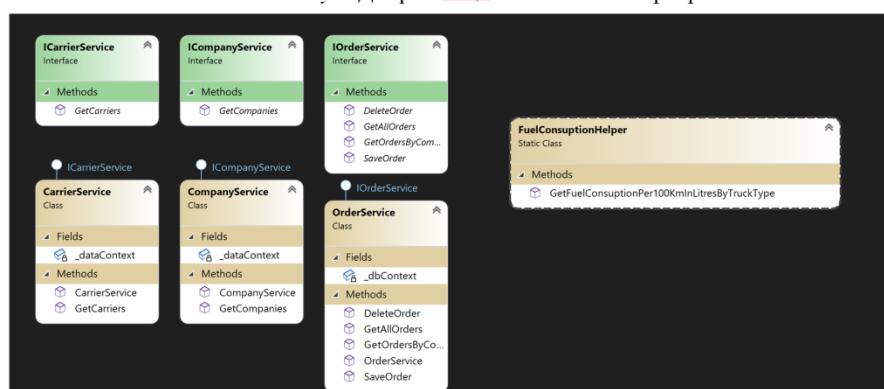


Архітектура додатку серверної сторони в Sql Server



Архітектура додатку серверної сторони в Visual Studio 2022

2. Infrastructure and Services layer. Для реалізації бізнес-логіки програми.



Архітектура додатку серверної сторони

2. WebApi layer. Для реалізації веб сервіса Rest Api.

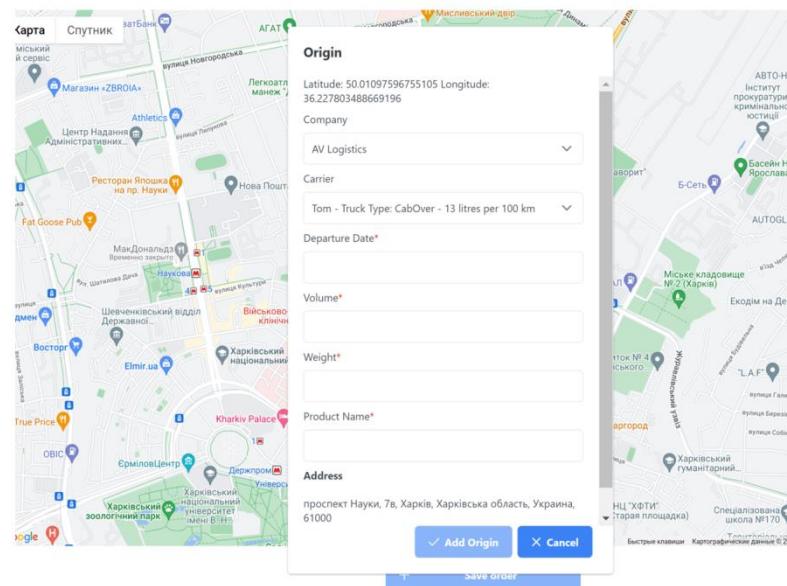
The screenshot shows the Swagger UI for the **MapAudit.WepApi**. The interface includes a navigation bar with the title and a dropdown for selecting a definition. Below this, there are sections for **Carrier**, **Company**, and **Order**. The **Order** section is expanded, showing methods for **POST /Order/AddOrder**, **GET /Order/GetAllOrders**, **GET /Order/GetOrdersByCarrierCompany**, and **DELETE /Order/RemoveOrderById**. At the bottom, there are sections for **Schemas** containing **DeliveryPoint** and **OrderViewModel**.

Програмна реалізація

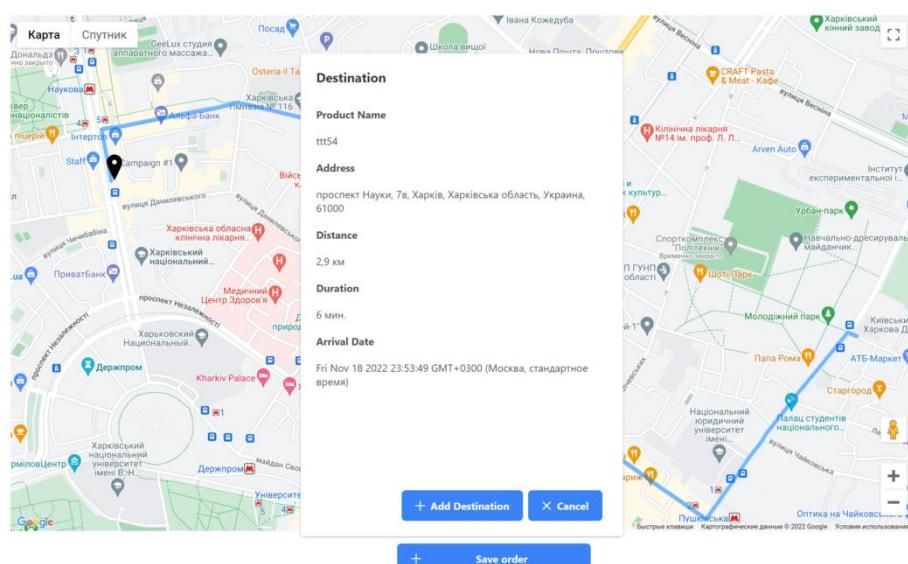
The screenshot displays the application's main interface. At the top, there is a navigation bar with links for .Net, English, Reading, Projects, job, sq, Gmail, driving, shop, and diploma magistr. Below this is a main menu with options for Main Menu, Carrier Land Map, and Carrier Air Map. On the left, there is a sidebar titled "Make order" with the sub-instruction "Choose origin and destination". On the right, there is a large map of Kyiv, Ukraine, showing various delivery points marked with icons. A button labeled "Save order" is located at the bottom right of the map area.



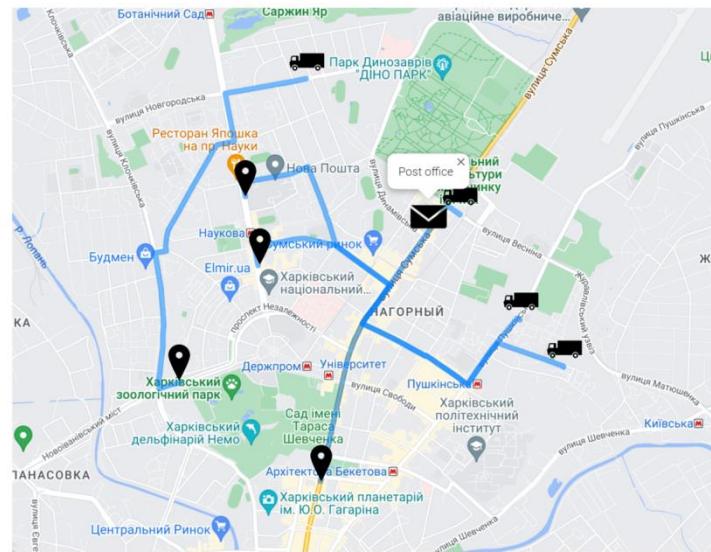
Програмна реалізація



Програмна реалізація



Програмна реалізація



Програмна реалізація

Total distance: 21,588 km

Total time: 0.6688888888888889 hours

