Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп‘ютерних наук
Кафедра Програмної інженерії

АТЕСТАЦІЙНА РОБОТА (ПРОЕКТ)

Пояснювальна записка

другий (магістерський)

«Дослідження генетичних алгоритмів для реалізації підтримки метромарафонів в системі “Transit challenge”»

Виконав: студент 2 курсу, групи ПЗСм-18-1
спеціальності
121- Інженерія програмного забезпечення
освітньо-професійної програми
Програмне забезпечення систем
Гордієнко Т. О.

Керівник: доц. Мазурова О. О.

Допускається до захисту
Зав. кафедри, проф. _____ Дудар З. В.
(підпис)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп‘ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121- Інженерія програмного забезпечення

Тип програми освітньо-професійна програма

Освітня програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 2019 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Гордієнку Тарасу Олександровичу

1. Тема роботи проекту «Дослідження генетичних алгоритмів для реалізації підтримки метромарафонів в системі “Transit challenge”» затверджена наказом по університету від «___»_____2019р. № _____
2. Термін подання студентом роботи (проекту): _____ 2019р.
3. Вихідні дані до роботи: генетичний алгоритми для пошуку оптималного шляху для метромарафону, пояснювальна записка. Використовувати MacOS/Windows, текстовий редактор VS Code.
4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз проблемної галузі і постановка задачі, дослідження генетичних алгоритмів, розробка математичної моделі метромарафону, розробка генетичного алгоритму, експериментальне дослідження розробленого алгоритму.

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Мазурова О.О		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної області та постановка задачі	18.09.19 – 30.09.19	виконано
2	Аналіз існуючих методів	30.09.19 – 10.10.19	виконано
3	Аналіз та моделювання предметної області	10.10.19 – 12.10.19	виконано
4	Розробка математичної моделі предметної галузі	12.10.19 – 14.10.19	виконано
5	Аналіз генетичних алгоритмів та їх основних компонентів	14.10.19 – 16.10.19	виконано
6	Розробка схеми бази даних	16.10.19 – 19.10.19	виконано
7	Розробка генетичного алгоритму	19.10.19 – 25.10.19	виконано
8	Створення коду програми	25.10.19 – 10.11.19	виконано
9	Тестування і налагодження програми	10.11.19 – 20.11.19	виконано
10	Підготовка пояснювальної записки	20.11.19 – 25.11.19	виконано
11	Підготовка презентації та доповіді	25.11.19 – 15.12.19	виконано
12	Попередній захист	16.12.19	виконано
13	Нормоконтроль, рецензування	16.12.19	виконано
14	Занесення диплома в електронний архів	17.12.19	виконано
15	Допуск до захисту у зав. кафедри	19.12.19	

Дата видачі завдання 18 09 2019 р.

Студент _____
(підпис)

Керівник роботи _____ ___доц. Мазурова О.О.___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної магістерської роботи містить 32 рисунка, 8 таблиць.

ГЕНЕТИЧНИЙ АЛГОРИТМ, ГРАФ, ЗАДАЧА КОМІВОЯЖЕРА, ПОШУК ОПТИМАЛЬНОГО ШЛЯХУ, ПРОБЛЕМА СТАНЦІЙ.

Об’єкт розробки – Веб-застосування для підтримки метромарафонів в системі “Transit Challenge”.

Мета розробки – дослідження генетичних алгоритмів та розробка системи, яка б надала змогу знаходити оптимальні маршрути для проведення метро-марафонів.

Методи рішення – методи базуються на теорії графів, а саме на рішенні задачі комівояжера (Transit saleman problem) з деякими відмінностями за допомогою генетичних алгоритмів. А також на використанні сучасних інструментів розробки веб-застосувань – мови програмування JavaScript та фреймворку Svelte.

В результаті роботи розроблено математичну модель метромарафону та проведено дослідження генетичних алгоритмів, що надало змогу знаходити оптимальні маршрути для невеликих метрополітенів, типу українських. Також на основі проведеного дослідження було розроблено програмну систему.

GENETIC ALGORITHM, GRAPH, TRAVELING SALESMAN PROBLEM, FINDING THE OPTIMAL PATH, STATION PROBLEM.

Object of development – Web-applications to support the "Transit Challenge".

The purpose of the development is to develop a system that has made it possible to find optimal routes for a "Transit Challenge".

Decision methods – the methods are based on the graph theory, namely on the Transit saleman problem with some differences using genetic algorithms. And also on

use of modern tools of development of web-appendices – programming language JavaScript and framework SvelteJS.

As a result, a mathematical model of the "Transit Challenge" has been developed and a study of genetic algorithms has been carried out, which made it possible to find the best routes for small subways, such as Ukrainian ones. Also, on the basis of the conducted research, a software system was developed.

ЗМІСТ

ВСТУП

Метро – це досить популярний засіб пересування у великих містах. Практично у кожному густонаселеному місті є метро. В Україні є два великих метрополітени – Харківський та Київський, які мають 30 та 52 станції відповідно. Досить популярним сценарієм використання метро є побудова найкоротшого маршруту між двома чи трьома станціями. Але є більш цікавий сценарій, коли потрібно відвідати абсолютно всі станції метрополітену. І саме такий сценарій представляє спортивний інтерес у багатьох країнах світу серед мандрівників та людей, які зацікавлені у такого роду змаганнях.

Це змагання називають по-різному: «Transit Challenge» чи «Subway Challenge». В країнах СНГ воно більш відомо як метромарафон. Суть цього метромарафону дуже проста: відвідати всі станції метрополітену за найкоротший час. Існує навіть сайт rapidtransitchallenge.com, де зафіксовані правила та рекорди цього змагання.

Перед тим як почати змагання учасники заздалегідь планують маршрут та час, який буде затрачено. Складність цього процесу в тому, що чим більший метрополітен, тим більше різних маршрутів існує. Також в різний час електропоїзди ходять з різними інтервалами, по-різному зупиняються на станціях та мають різну завантаженість. Тому врахування навіть таких незначних деталей може дати виграш у часі.

В Україні такі змагання ніхто не проводив, та навіть не публікував свої результати з побудованим маршрутом. Тому було б цікаво знайти такий маршрут, випробувати його та дізнатись чи може хтось знайти більш оптимальний.

Метою роботи є дослідження генетичних алгоритмів та розробка системи, яка б надала змогу шукати оптимальні шляхи для проведення метромарафону. Галузь застосування обмежена туристами, яка цікавляться урбаністикою та ентузіастами, які знаходять спортивний інтерес в такого роду змаганнях.

Під час виконання магістерської роботи було розроблено математичну модель на основі дослідження проблемної галузі та на її основі сформовано проблему станцій. В слід за цим, було досліджено та застосовано генетичні алгоритми для вирішення цієї проблеми. Також було сформовано основні принципи використання генетичних алгоритмів в рамках даної проблеми. Було обрано спосіб задання геному, сформовано функцію пристосованості, види мутацій та принципи схрещування. Було обрано спосіб, за яким данні графа будуть зберігатися в системі. Після цього було проведено експеримент в результаті якого було сформовано основні рекомендації щодо використання розробленого алгоритму, а саме вибору початкових параметрів та їх зміни в процесі роботи програми. Генетичні алгоритми показали гарний результат на метрополітенах, кількість станцій який не перевищує 100. До таких метрополітенів саме відносяться метрополітени України: Харківський метрополітен та Київський метрополітен.

В ході атестаційної роботи магістра було:

– проведено аналіз та моделювання предметної області;

– розроблено математичну модель метромарафону;

– розроблено схему бази даних та структуру для збереження інформації та роботи з графами;

– на основі математичної моделі було розроблено генетичний алгоритм для пошуку оптимального маршруту для метромарафону;

– реалізовано веб-додаток для пошуку оптимальних шляхів в метрополітені;

– проведено дослідження генетичних алгоритмів для побудови маршрутів метромарафонів на Харківському та Київському метрополітенах. Також алгоритм було протестовано на вигаданому графу, який імітує більш складнішу транспортну схему, чим українські метрополітени;

– сформовано рекомендації щодо початкових параметрів розробленого алгоритму та їх зміні в процесі роботи програми

– за результатами проведеного дослідження сформовано рекомендації щодо

початкових параметрів розробленого алгоритму та рекомендації щодо стратегії їх зміні в процесі роботи програми.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток А).

Було зроблено доповідь на Міжнародному молодіжному форумі. Тези доповіді наведено в додатку Б. Також була написана та подана до опублікування до науково-технічного журналу «Біоніка інтелекту» стаття «Дослідження генетичних алгоритмів для пошуку оптимальних шляхів в системі проведення метромарафонів» (див. додаток В). Лістинг частини коду розроблюваної системи наведено в додатку Г.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області

В багатьох великих містах метро є дуже популярним та зручним засобом для пересування містом. Це пояснюється тим, що електропоїзди рухаються під землею, тому не можуть потрапити в затор на дорозі, а значить прибувають на станцію вчасно. Тим більше, інтервали руху поїздів досить короткі.

В таких містах України, як Харків та Київ досить налагоджена муніципальна транспортна система. Вона складається з трамваїв, тролейбусів, автобусів та метрополітену. Метро займає особливе місце в цій системі. За даними Головного управління статистика у Харківській області в день метро перевозить більше півмільйона пасажирів. За рік пасажири здійснюють близько 206 мільйонів поїздок [13]. Можна зробити висновок, що метро є однією з провідних частин у формуванні маршрутної системи в місті.

При такій кількості пасажирів виникає багато проблем, пов’язаних з орієнтуванням в метро. Ці проблеми повинні вирішуватись навігацією метрополітену. Але в Харкові навігація дуже далека від ідеальної. Не рідко можна зустріти людину, яка питає у перехожих як їй добратися на потрібну станцію, чи як потрапити на перехід.

Перезапуск Харківського метро дуже складна та багатозатратна задача. Змінити таблички, схеми та перезаписати оголошення — питання великої роботи, впливу і обговорень. З тим є проблеми, які можна вирішити за допомогою інформаційних технологій, які пов’язані з метро тільки темою і ніяк не відносять до зміни самого метро. Це, наприклад, веб-застосунки та мобільні додатки. За допомогою них можна значно полегшити задачу орієнтування.

Зараз все більше пасажирів метрополітену використовує інтернет-ресурси для прокладання маршрутів по місту. Але проблема всіх цих ресурсів в тому, що маршрути в метро складаються дуже поверхнево. Зарубіжні системи, якими користуються харків’яни ніяк не враховують особливості метрополітену, не

вказують внутрішнє влаштування маршруту і не допомагають оптимізувати шлях в самому метро. Вони тільки вказують які станції потрібно проїхати та де перейти на іншу лінію.

Основною проблемою, яку можна вирішити за допомогою створення інформаційної системи — це побудова та оптимізація маршруту пасажира всередині метро. Проблема розрахунку оптимального маршруту в метро актуальна для студентів, які приїжджають навчатись до Харкова, для людей, які домовилися про зустріч в місті. Якщо побудувати оптимальний маршрут можна просто дивлячись на схему, навіть враховуючи інтервали руху поїздів, то розрахувати час поїздки власноруч практично неможливо.

При цьому багато туристів, які цікавляться урбаністикою, відвідуючи місто, де є метро, одразу спускаються під землю та їдуть дивитись на найпопулярніші та найгарніші станції. Такий інтерес визваний тим, що багато станцій є витворами архітектурного мистецтва, поєднаного з технологіями. Тому метро представляє великий інтерес для людей, зацікавлених темою урбаністики.

Тема відвідування всіх станцій метро представляє також і спортивний інтерес серед ентузіастів, які цікавляться математикою, урбаністикою та туризмом. В багатьох країнах світу проводяться цілі змагання, де потрібно за найкоротший час відвідати всі станції метрополітену. Такий вид змагань називають «Transit challenge», «Subway challenge», «Underground challenge» або просто метромарафон.

При пошуку оптимальних шляхів в метро, його частіше за все моделюють в виді графа, де вершини графу – станції, а перегони – ребра [20]. Тому методів рішення такого роду задач існує дуже багато. Для пошуку оптимального шляху між двома станціями частіше за все використовують найпопулярніший алгоритм для вирішення такого роду задач – алгоритм Дейкстри [9].

Але в випадку метромарафону, де потрібно знайти маршрут, який включає всі станції, задача є набагато складнішою. Вона має досить високу складність обчислення, якщо знаходити її точний розв’язок, бо варіантів об’їхати весь метрополітен дуже багато. Навіть якщо він має всього три лінії, не говорячи про

метрополітени Нью-Йорку та Лондону. Тому використовують евристичні методи, які не дають гарантію, що знайдений маршрут буде найоптимальнішим, але дають гарний компроміс між оптимальність маршруту та затраченим часом на його пошук.

Деякі з цих евристичних методів намагаються скоротити вже побудований маршрут. Такі методи видаляють з маршруту цілі групи з ребер і замінюють їх іншими ребрами, отримуючи новий маршрут. Інші евристичні методи комбінують пошук глобальних та локальних розв'язків [8].

Такі методи як, мурашиний алгоритм, генетичні алгоритми та нейронні мережі використовують ідеї природних процесів та показують досить високу ефективність порівняно з іншими методами [5].

Для вирішення цих проблем доцільно створити єдину інформаційну систему, яка дасть змогу користувачам власноруч знаходити оптимальні шляхи в метрополітені. Така система дозволила б ентузіастам, які зіткнулися з проблемою пошуку оптимального шляху для задачі на кшталт метромарафону, полегшити його знаходження та заощадити час.

При створенні такої системи, задачі, покладені на неї можна розбити на наступні складові:

– гнучкий вибір потрібного метрополітену серед запропонованих;

– можливість вводу стороннього метрополітену або спорідненої структури в виді посилання на json-файл;

– можливість вибору часу, який необхідно буде провести на кожній станції метрополітену (для того, щоб роздивитися станцію чи сфотографувати її та дочекатись наступного поїзда);

– відображення матриці суміжності та векторної схеми обраного графу;

– побудова оптимального шляху, який би дозволив відвідати всі станції метрополітену за найкоротший час;

– відображення даних алгоритму при пошуку оптимального шляху;

– керування параметрами алгоритму для більш оптимального пошуку;

– можливість зупинити алгоритм в будь-який момент;

– керування швидкістю роботи алгоритму для можливості використання системи на машинах з різною обчислювальною потужністю;

– відображення повного маршруту з відображенням станцій, на яких потрібно вийти з вагону;

– відображення часу, який буде затрачено на кожний етап маршруту;

– відображення схема та анімації маршруту на ній;

– можливість змінити швидкість анімації та продивитися отриманий маршрут на схемі;

Для вирішення перерахованих задач потрібно виповнити такі етапи проектування: розробка дизайну та інтерфейсу; розробка концептуальної моделі; реалізація алгоритму; реалізація клієнтської частини.

Створювана програмна система повинна вирішити проблему пошуку оптимального шляху в метрополітені, який би включав всі станції. При цьому система повинна бути гнучкою для різних метрополітенів та схожих структур та надавати змогу керування параметрами алгоритму безпосередньо при виконанні програми.

1.2 Виявлення проблем та актуалізація рішень

На сьогодні в інтернеті можна знайти багато додатків для побудови транспортних шляхів. Зокрема, для побудови шляху в метро з пересадками. Більшість з них вміє розраховувати приблизний час поїздки та показувати найкоротший шлях від однієї станції до іншої. Кожен з цих додатків має свої переваги, призначення, специфіку та недоліки. Розглянемо декілька найбільш успішних та популярних на прикладі Харківського метрополітену, а саме:

– додаток для обчислення часу на офіційному сайті Харківського метрополітену (metro.kharkov.ua/ru/estim/index);

– додаток від Яндекса – Яндекс-Метро (metro.yandex.ua/kharkov);

– інтерактивна схема Харківського метрополітену від Метробук з розрахунком часу (metrobook.ru/ua_kharkiv).

Треба зауважити, що ні один з додатків не має змоги шукати оптимальний маршрут між всіма станціями метрополітену. Тому буде розглянуто найпростішу задачу – побудова оптимального шляху між двома станціями.

Офіційний додаток на сайті Харківського метрополітену представлено на рисунку 1.1.

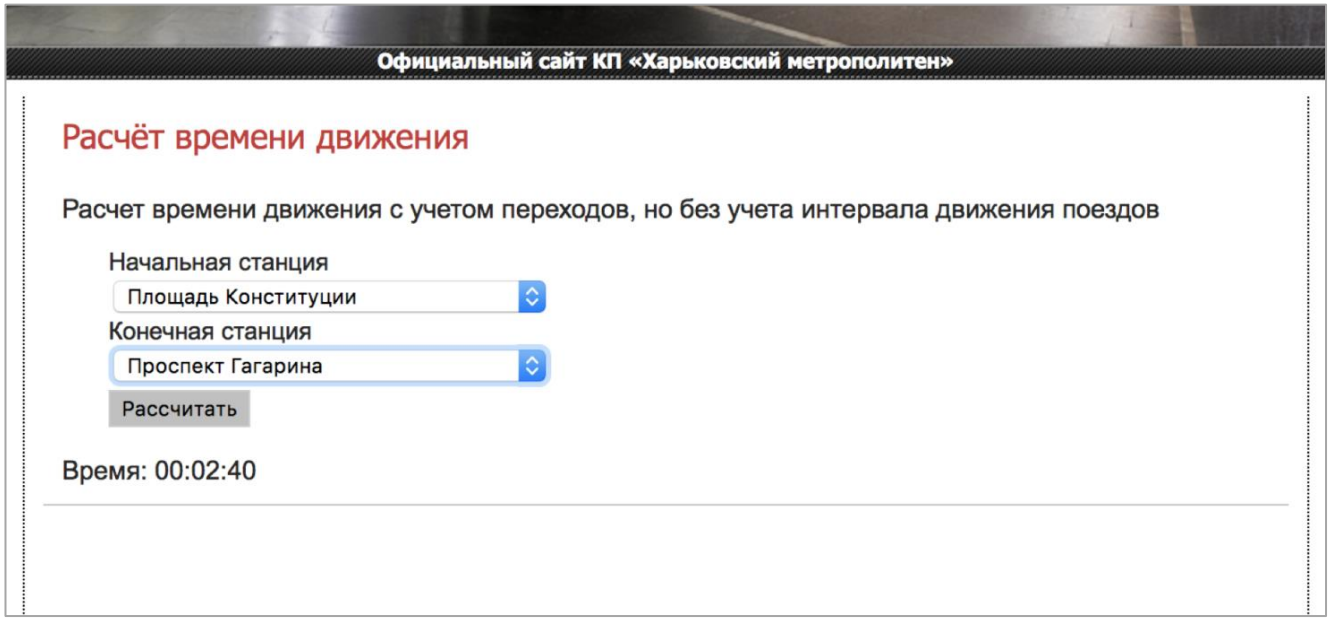


Рисунок 1.1 – Додаток на офіційному сайті метрополітену

Додаток має функцію розрахунку тільки приблизного часу між двома станціями [13]. Користувач отримує доступ до двох стандартних селектів: початкова станція та кінцева станція. Весь додаток не має оформлення.

Додаток виконано на російській мові, українська мова відсутня. Також додаток не враховує інтервали руху поїздів, що досить суттєво впливає на розрахунок часу та навіть маршрут.

Наступний аналог -– додаток від Яндекса Яндекс-метро для Харкова [14]. Так як зараз сервіси Яндекса внесені до списку заблокованих інтернет-ресурсів, практичної користі цей додаток несе дуже мало, але цікаво розглянути його з точки зору функціоналу та інтерфейсних рішень, бо в Яндексі дуже відповідально

підходять до проектування інтерфейсних рішень та дизайну. Один з робочих екранів представлено на рисунку 1.2.

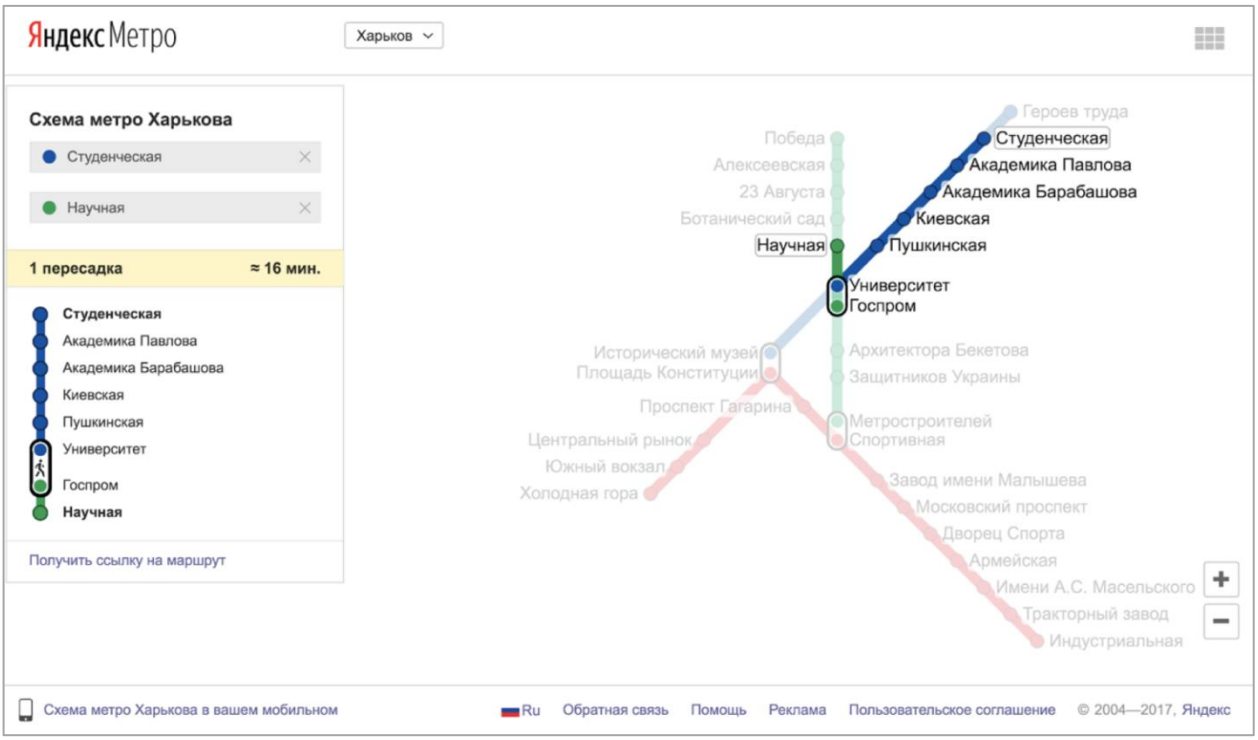


Рисунок 1.2 – Інтерфейс Яндекс-метро для Харкова

На відміну від додатка офіційного сайту Харківського метрополітену, тут представлена інтерактивна схема метро, де можна обрати станцію, клацнувши на неї. Маршрут прокладається візуально, що дуже полегшує сприйняття інформації. Також є мобільний додаток під iOS, Android та Windows Phone. Це значно розширює можливості використання додатку. З найочевидніших плюсів можна виділити розширену локалізацію. У користувача є змога змінити мову на українську, російську чи білоруську.

Приблизно таку ж саму функціональність має і третій аналог: інтерактивна схема Харківського метрополітену від Метробук [15].

Якщо порівняти данні аналогів, то можна помітити, що вони видають абсолютно різні результати. Для порівняння прокладемо маршрут та розрахуємо час від станції «Центральний ринок» до «Наукової». Результати представлені на рисунку 1.3.

Додаток на офіційному сайту показує зовсім інший маршрут, чим два інші аналога. При тому розрахований час набагато більше реального. Найбільш точним виявився Яндекс, якщо враховувати, що маршрут прокладено в години пік.

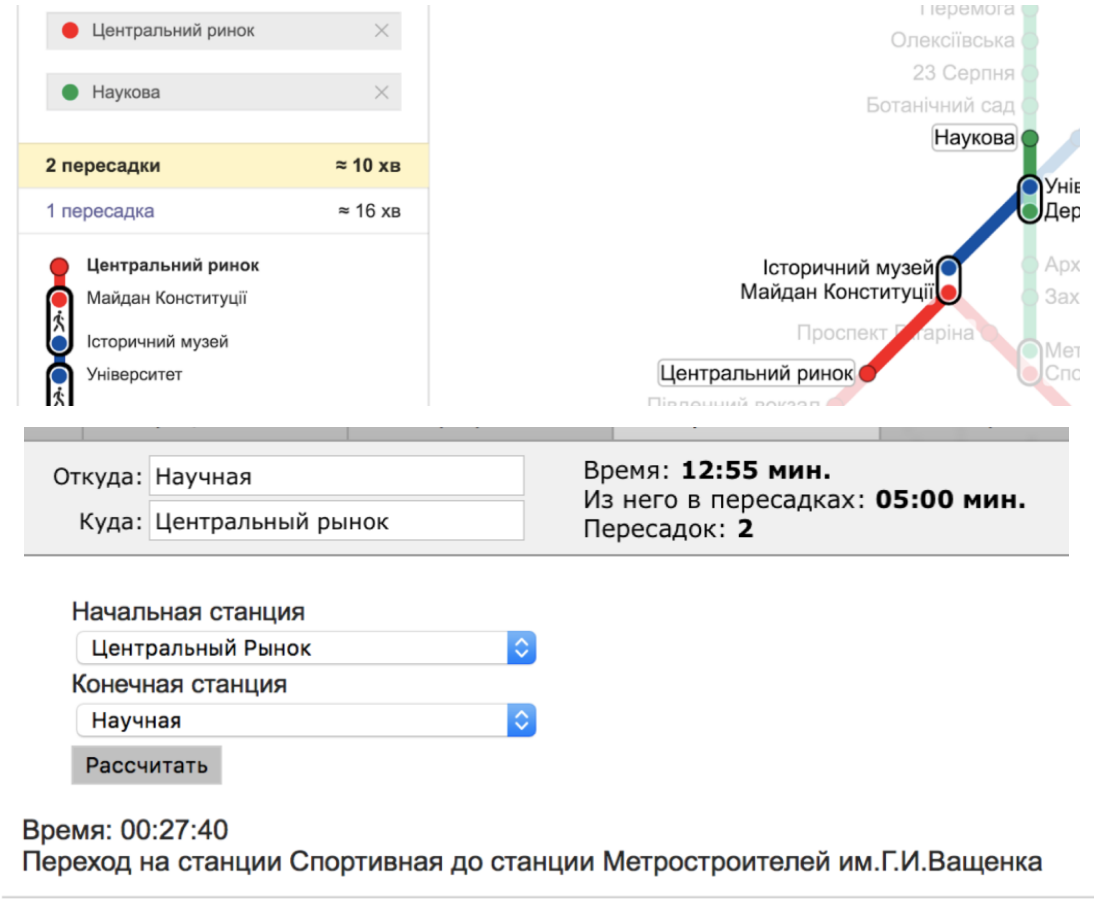


Рисунок 1.3 – Один маршрут на різних аналогах

Отже, при відсутності прямих аналогів та не дуже гарної якості приблизних аналогів, існує необхідність створити додаток, який би дозволив розрахувати не тільки оптимальний маршрут між двома станціями, але й такий, що включав би в себе всі інші. Ні один з розглянутих додатків не враховує інтервали руху поїздів та не показує подробиці маршруту для комфортного та оптимального руху метрополітеном. Ці підстави дають вважати, що розроблюваний додаток буде корисний для туристів та зацікавлених темою метромарафонів.

1.3 Постановка задачі

Метою роботи є дослідження генетичних алгоритмів та розробка системи на основі дослідження, яка б надала змогу знаходити оптимальні шляхи в метрополітенах та схожих структурах. Система повинна бути виконана у вигляді клієнтського веб-застосунку і мати базу даних, в якій будуть зберігатися данні про схему, інтервали, час руху та список станцій.

Було розроблено схему вхідної та вихідної інформації [12]. Цю схему зображено на рисунку 1.4.

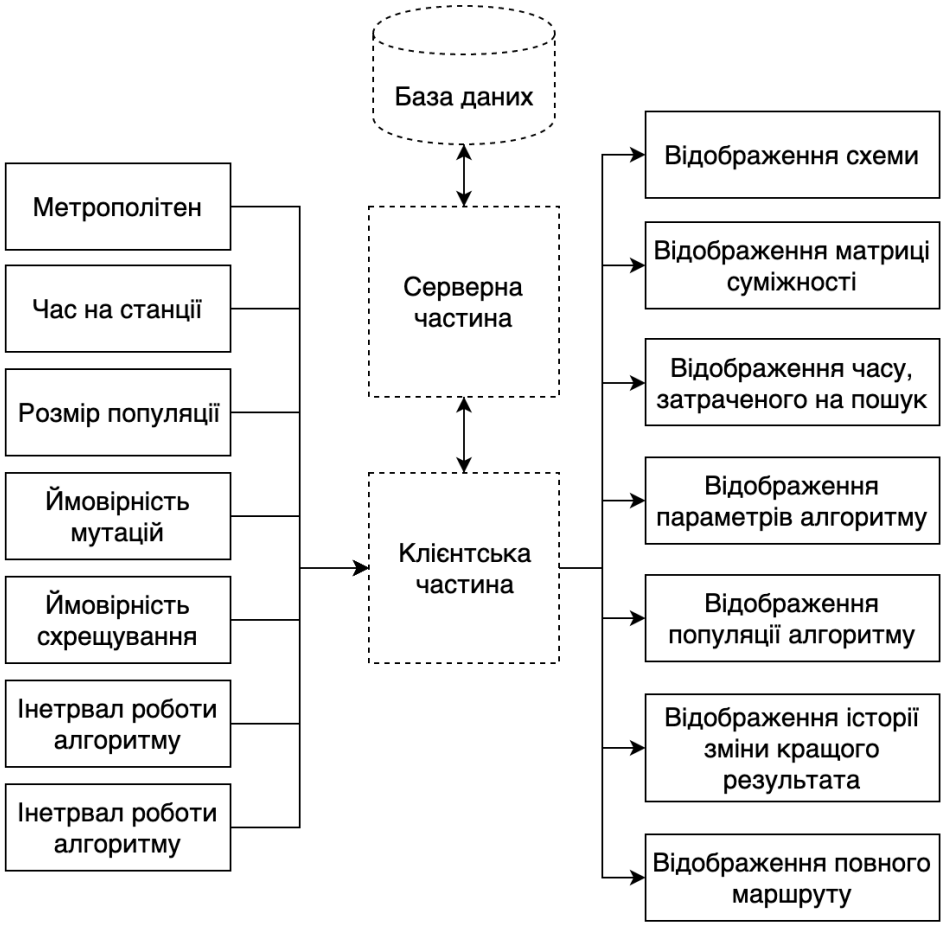


Рисунок 1.4 – Схема вхідної та вихідної інформації

На початковому етапі концептуального моделювання потрібно провести аналіз існуючої інформації про предметну область, визначити об’єкти та їх

атрибути, зв’язки між цими об’єктами, майбутніх користувачів системи та їх функції, описати документообіг та статистику.

Дана система повинна стабільно і правильно працювати у всіх сучасних браузерах з підтримкою стандарту EcmaScript 6. Користувач повинен мати змогу зручно користуватися системою на персональному комп’ютеру, ноутбуку та мобільному телефоні. Всі данні повинні опрацьовуватися на клієнтській стороні. Це дасть змогу відкрити вкладку в браузері і потім, без підключення до мережі, взаємодіяти з нею.

Таким чином потрібно виконати такі задачі:

– провести аналіз та моделювання предметної області проведення метромарафонів;

– розробити математичну модель метромарафону;

– розробити схему бази даних та структуру для збереження інформації з математичної моделі;

– на основі математичної моделі розробити генетичний алгоритм для пошуку оптимального маршруту метромарафону;

– реалізувати веб-додаток для пошуку оптимальних шляхів в метрополітені;

– провести дослідження генетичних алгоритмів для побудови маршрутів метромарафону в Харківському та Київському метрополітенах;

– за результатами проведеного дослідження сформувати рекомендації щодо початкових параметрів розробленого алгоритму та їх зміни в процесі роботи програми.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Функціональні вимоги ПЗ

Веб-додаток повинен бути розроблений за допомогою мови програмування JavaScript по специфікації EcmaScript 6. Також повинен бути підключений транспайлер Babel для підтримки попередніх специфікацій. Повноцінна серверна частина буде відсутня, та буде реалізована на примітивному рівні лише для роботи з базою даних. Клієнтська ж частина працюватиме без окремого фреймворка, та буде побудована на архітектурі Flux. У якості технології для роботи з базою даних буде використовуватись розширення файлів json та спосіб роботи з запитами fetch.

При перегляді сайту не буде розділення на ролі користувачів. Адміністрування додатку не є основною задачею системи. Це пов’язано з тим, що зміна даних досить нечаста. Тому виділяти час на розподіл ролей в системі не є релевантним рішенням і наявність особистого кабінету буде зайвим функціоналом. Зміни в проекті будуть вноситись через програмний код по FTP.

Якщо потрібно буде запам’ятати якусь інформацію, наприклад, введені параметри користувачем, то оптимальним рішенням буде використати технологію LocalStorage. Вона домопогає зберігати дані на стороні клієнта. Так як нема необхідності пов’язувати ці данні з сервером, використовувати для вирішення цієї задачі cookie немає необхідності. Якщо користувач заблокує використання LocalStorage, то це буде не критично для роботи системи. Просто деякі корисні функції не будуть працювати.

Після завантаження сторінки користувач одразу буде мати доступ до наступного функціоналу:

– обрати за допомогою кліку чи тапу потрібний метрополітен;

– обрати варіант користувацького метрополітену та ввести посилання на json-файл з даними по його графу;

– змінити кількість хвилин, які буде затрачено на вихід зі станції та очікування наступного потягу.

– ввести значення кількості особин в популяції;

– ввести значення ймовірності схрещування при роботі алгоритму;

– ввести значення ймовірності мутацій при роботі алгоритму;

– ввести значення інтервалу в мілісекундах для роботи алгоритму;

– по кліку чи тапу запустити роботи алгоритму та зупинити його в будь-який час;

Основою системи повинна стати схема метро. Навколо неї буде побудований весь функціонал. Схема повинна бути зрозумілою і простою. Зараз, серед існуючих схем Харківського метрополітену, таких немає. Тому потрібно намалювати власну. При тому її потрібно розробити у векторному редакторі Adobe Illustator чи якомусь подібному. Це пов’язано з тим, що потім цю схему можна буде імпортувати в SVG формат для вставки безпосередньо на сторінку.

Для всього тексту на назв станцій потрібно буде використовувати шрифти, які просто розрізнити. Та обов’язково з великим набором знаків для коректного відображення текстів на інших мовах.

Так як система розрахована на широку аудиторію, яка включає в себе не тільки людей, у який дуже швидке з’єднання з інтернетом та найсучасніший комп’ютер, то при створенні системи потрібно взяти до уваги наступні аспекти:

– швидкість завантаження сторінки, її елементів та затримка до можливості взаємодії з елементами інтерфейсу. Схема та інші елементи повинні чітко відображатися на екранах з гарною роздільною здатністю та на застарілих моніторах;

– доступність елементів, можливість користуватися системою без мишки, а тільки за допомогою клавіатури;

– зручність роботи з сайтом. Інтерфейс повинен бути зручний та інтуїтивно зрозумілий. Він не повинен вимагати спеціальних знань та умінь роботи з комп’ютером чи телефоном.

– довідкова інформація повинна бути розташована на першому екрані та бути добре видимою для користувача;

– у випадку, якщо з’єднання з інтернетом буде перервано, але сторінка при цьому була завантажена та зберігається у пам’яті, то система має правильно працювати і виконувати поставлені задачі. Тобто, основні задачі системи повинні виконуватись в браузері на стороні клієнта.

2.2 Вимоги до клієнтської частини

Система розробляється у вигляді веб-додатку. Тому користувач не повинен мати якихось спеціальних умов для того, щоб нею користуватися. Серед базових умов до клієнта можна виділити такі:

– персональний комп’ютер чи смартфон з підключенням до інтернету та бажано з екраном більше 4 дюймів. З екраном меншого розміру може бути не так комфортно працювати;

– наявність будь-якого сучасного браузера. В застарілих браузерах типу «Internet Explorer 6» користувач отримає сповіщення, що його браузер застарів і що веб-додаток може працювати некоректно.

Система повинна бути побудована таким чином, щоб люди з вадами в сприйнятті кольорів також могли користуватися нею. Тому серед вимог до клієнта потрібно враховувати, що у нього може бути певна форма дальтонізму чи інші вад зору. Також користувач не повинен мати швидкого з’єднання з інтернетом та найсучаснішого комп’ютера.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Аналіз та моделювання предметної області проведення метромарфонів

Проблема метромарафону в загальному вигляді формулюється досить просто – необхідно за якомога коротший час проїхати весь метрополітен. При таких простих умовах виникає декілька різновидів метромарафону:

а) необхідно відвідати всі станції метрополітену, не виходячи з поїзду;

б) необхідно відвідати всі станції метрополітену, виходячи з поїзду та чекаючи наступного;

в) необхідно відвідати всі лінії метрополітену, тобто проїхати по кожному з перегонів.

В контексті створення даної системи для двох українських метрополітенів (Харківський та Київський) останній вид метромарафону, при якому потрібно відвідати всі перегони, не є актуальним. Це пояснюється тим, що українські метрополітени, зокрема Харківський та Київський, не мають станцій, які б належали одразу двум лініям.

Серед перших двох умов більш актуальною є друга умова. Саме вона представляє спортивний інтерес та найбільш популярна серед учасників метромарафонів.

Для предметної області основними поняттями є станція метро та лінія метро. Так як лінії метро зв’язані між собою переходом на деяких станціях, то необхідно ввести ще одну допоміжну сутність – перехід між лініями. Це дасть змогу отримати станції, які пов’язують лінії метро між собою.

Також важливо враховувати інтервали руху поїздів. Отже, потрібно додати ще одну сутність для представлення інтервалів. Ця сутність обов’язково повинна враховувати лінію метро, бо інтервали руху на кожній лінії відрізняються.

За допомогою побудови діаграми Use Case або діаграми прецедентів можна прослідкувати за сценаріями використання програмного продукту. Хоча діаграма

і має одну роль, її побудова є досить важливою. Побудовану діаграму представлено на рисунку 3.1.

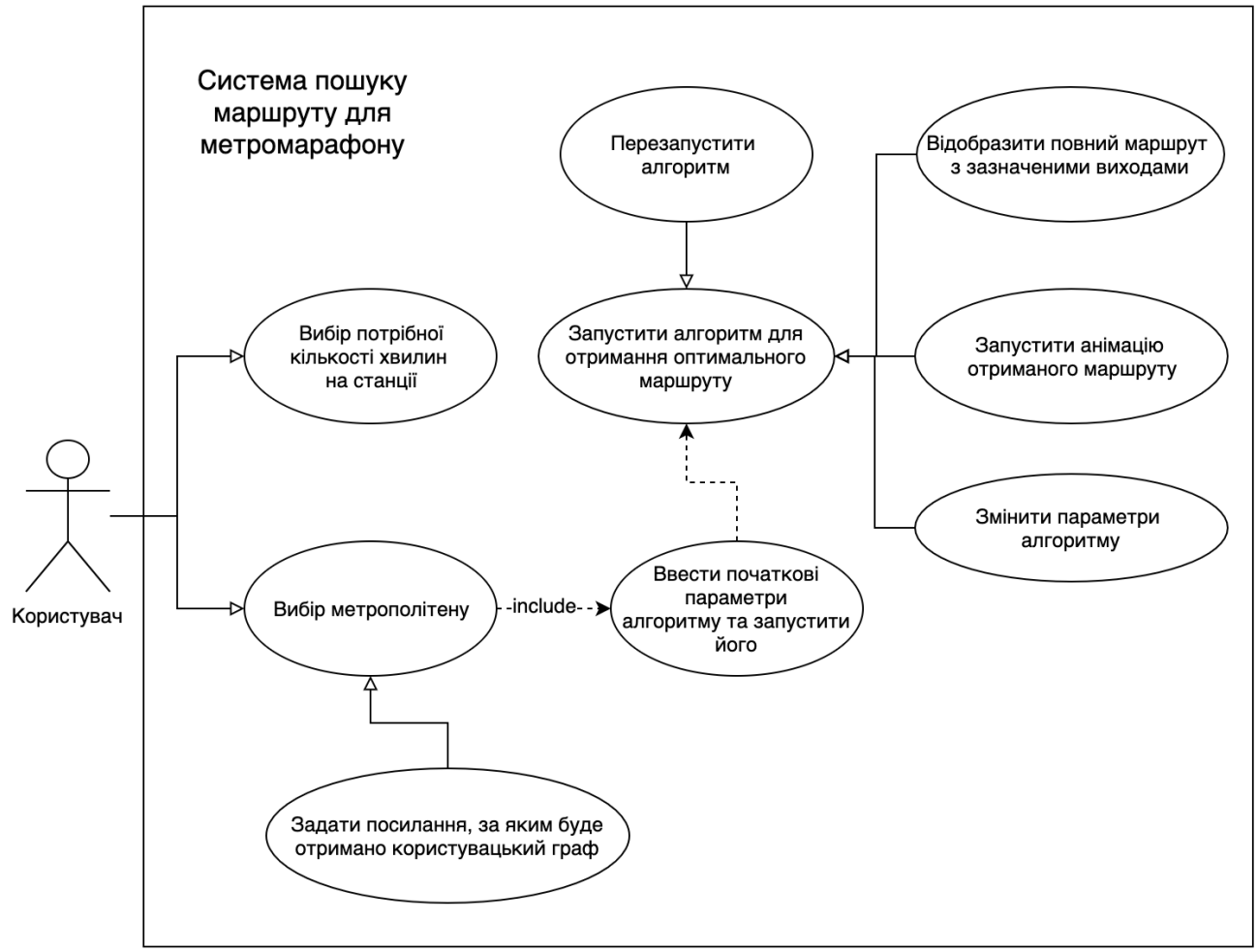


Рисунок 3.1 – Діаграма Use-Case

Таким чином, вибір об’єктів та сутностей предметно, встановлення взаємозв’язків та сценаріїв показує цілісну картину розроблюваної програмної системи. За результатами аналізу та моделювання предметної області можна спроектувати базу даних з обраними сутностями та взаємозв’язками. Також, в результаті моделювання стало відомо, які функції та сценарії потрібно реалізувати для повноцінної роботи програмної системи згідно поставлених вимог. Ці сценарії було відображено на діаграмі Use-Case.

3.2 Розробка математичної моделі системи метромарафону

При пошуку оптимальних шляхів метрополітени зазвичай представляють у вигляді графів, де вершина – це станція, а дуга – це перехід чи перегон між станціям. Час, необхідний для переходу з однієї вершини до іншої, моделюється як вага дуги. Тому в кожному випадку враховується середній час на здійснення кожного переходу, а не фактичний час метрополітену. На щастя, результати показали, що середній час наближається до фактичного часу. Це пов’язано з тим, що будь-який метрополітен – це дуже налагоджена система, яка працює по чіткому графіку.

Слід зазначити, що існують і інші проблеми маршрутизації, схожі на проблему пошуку оптимального маршруту в метро: По-перше, існує цілий клас, проблем під назвою Arc Routing Problems (Arc Routing Problems, ARP). Такі проблеми більше зосереджуються на властивостях дуг і маршруту в цілому [4].

До таких проблем можна віднести:

– проблема китайської листоноші (Chinese Postman Problem, CPP), де потрібно знайти оптимальний маршрут по всім дугам графа;

– проблема сільського листоноші (Rural Postman Problem, RPP), де потрібно знайти тільки оптимальний маршрут на певній підмножина графа.

В тому чи іншому сенсі варіантами ARP можна назвати різні ситуації, які виникають в житті. Наприклад доставка пошти, збір сміття, доставка суші чи підтримка великої мережі. Однак, не всі ці приклади можуть бути змодельовані, як CPP або RPP проблеми [11]. Оскільки в реальних умовах потрібно враховувати багато інших чинників, а знехтувати цими чинниками може бути критично для фінального результату.

Також, тісно пов’язана з пошуком оптимального шляху по всім станціям метрополітену проблема комівояжера, так звана (Traveling Salesman Problem, TSP). А також узагальнена версія цієї проблеми під назвою Generalized Traveling Salesman Problem (GTSP). Проблема полягає в тому, щоб знайти

найкоротший маршрут, який проходить через всі вершини. За допомогою такого моделювання можна вирішити багато практичних проблем, пов’язаних з логістикою та доставкою товарів [11].

При побудові моделі враховується, що потрібно відвідати всі вершини графа. Іншою, хоча і схожою на задачу комівояжера, вважається задача, в якій потрібно пройтися по всім дугам графу, тобто проїхатись по всім перегонам метрополітену. В даному ж випадку не має ніякого значення, яка лінія та напрямок використовуються для того, щоб відвідати станцію.

Така проблема дуже тісно пов'язана з асиметричним видом узагальненої задачі комівояжера (TSP) і є NP-повною задачею [7]. На основі цієї задачі ми і будемо будувати математичну модель.

Якщо представити транспортну мережу у вигляді певного графу, то отримаємо наступну структуру:

– скінченний набір станцій, який позначимо за S.

– скінченний набір ліній метрополітену L, де кожна лінія $L \in \mathrm{L}$ – це впорядкований набір станцій.

Припустимо, що кожна лінія працює в обох напрямках і що кожна лінія складається щонайменше з двох станцій.

Отже, тепер проблему можна змоделювати як орієнтований граф G = (V, A), де V – множина вершин, які зв’язані між собою множиною ребер A. Цей граф має такі властивості:

– для кожної лінії $L \in \mathrm{L}$ і всіх станцій, які $s \in \mathrm{L}$ завжди є початкова вершина $\vec{v}(s, \mathrm{L}) \in \mathrm{V}$;

– для кожної лінії $L \in \mathrm{L}$ і всіх станцій, які $s \in \mathrm{L}$ завжди є кінцева вершина $\overleftarrow{v}(s, \mathrm{L}) \in \mathrm{V}$.

Далі визначаємо множину всіх вершин, що належать станції $s \in \mathrm{S}$:

$$\mathrm{V}(s) = \mathrm{US}(\mathrm{L} \in \mathrm{L},\ s \in \mathrm{L})\{\ \vec{v}(s, \mathrm{L}),\ \overleftarrow{v}(s, \mathrm{L})\ \}$$

Також позначимо дуги, що відповідають сегментам, які утворюють перегін між двома станціями. Для всіх s $\in$ L маємо дугу ($\vec{v}(s, L)$, $\overleftarrow{v}(s`, L)$ , де s` наслідує s. Аналогічно позначимо дугу ($\vec{v}(s, L)$, $\overleftarrow{v}(s``, L)$ ), де навпаки s наслідує s``. Таким чином, набір дуг, що з'єднує тільки вершини однієї лінії та напрямку позначимо як R.

Крім цього, потрібно не забувати про переходи. Тож додамо дугу (u, v) для кожної можливої пари вершин у кожному V(s). Такі дуги дозволять змінювати напрямок.

Кожна дуга $a \in A$ має певну вагу t(a) > 0. Якщо a відноситься до R, то будемо вважати, що це час в дорозі між станціями, а якщо a $\notin$ R, то це відповідає часу, необхідному на перехід до іншої станції. Отриманий граф назвемо метрографом (графом метрополітену) від L та S.

На рисунку 3.2 зліва зображено схему метро, яка складається з двох ліній – червоної та синьої. Цей метрополітен маленький і має всього один перехід, але добре ілюструє принцип побудови графа.

На рисунку 3.3 зображено приклад побудови графу, який відповідає цьому метрополітену.

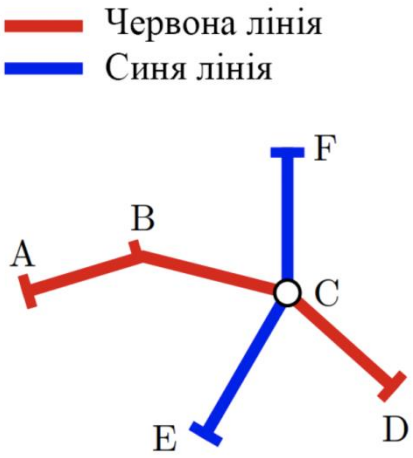


Рисунок 3.2 – Схема метрополітену з двох ліній

Користуючись даною математичною моделлю, розглянемо дві проблеми:

– проблема сегментів;

– проблема станцій.

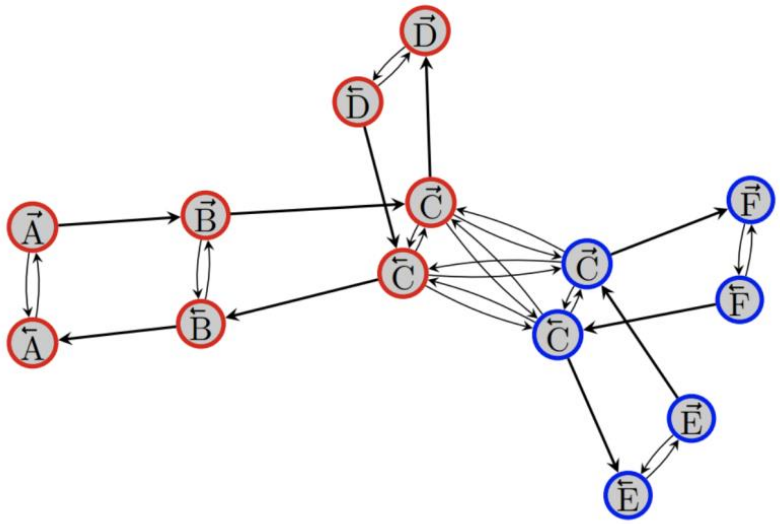


Рисунок 3.3 – Представлення ліній цього метрополітену в вигляді графа

Сформулюємо проблему сегментів наступним чином: нехай G = (V, A) є графом метрополітену з вагами дуги t(a), де t(a) > 0 та t(a) ∈ Q. Проблема сегментів полягає в тому, щоб знайти найдешевший маршрут T, що містить кожну дугу a ∈ R.

Сформулюємо проблему станцій наступним чином: нехай G = (V, A) є графом метрополітену з вагами дуги t(a), де t(a) > 0 та t(a) ∈ Q. Проблема станцій полягає в тому, щоб знайти найдешевший маршрут T, де кожній станції s ∈ S принаймні один вузол в V_S належить T.

В рамках даного дослідження нас цікавить лише друга проблема – проблема станцій, бо тільки вона задовольняє нашій умові, за якою має значення відвідування кожної станції, а не проїзд по всім перегонам.

Отже, проблема станції відповідає наступній ситуації: з урахуванням мережі метрополітену: потрібно відвідати кожну станцію хоча б один раз. При цьому не

має значення, яка лінія та напрямок використовуються. Перехід між двома лініями і проїзд в різні напрямки між станціями займає певну кількість часу. При тому цей час не завжди однаковий для кожного напрямку. Ціль задачі полягає в мінімізації загального часу, необхідного для такого маршруту.

Ця проблема дуже тісно пов'язана з асиметричним випадком задачі комівояжера і є NP-повною.

Математична модель такої задачі буде виглядати таким чином:

$$\min \sum_{a \in A} t_a x_a ,$$

де *a* – певна дуга, що *a* ∈ A;

t(a) – функція розрахунку ваги ребра a;

x(a) – значення з множини {1, 0}, при якому враховується чи належить дане ребро до оптимального маршруту чи ні.

Також за умовами проблеми станцій ми повинні додати до математичної моделі певні обмеження.

$$\sum_{a \in \delta^+(v)} x_a - \sum_{a \in \delta^-(v)} x_a = 0 \qquad \forall v \in V .$$

де δ +(*v*) – набір дуг без вершини v;

δ - (*v*) – набір дуг з вершиною v.

Перше обмеження гарантує, що рішення складається тільки з набору підмаршрутів.

$$\sum_{a \in \delta^+(V_s)} x_a \geq 1 \qquad \forall s \in S$$

Завдяки другому обмеженню кожна станція відвідується принаймні один раз в деякому підмаршруті, де S – це множина станцій.

$$\sum_{a \in \delta^+(C)} x_a + \sum_{a \in \delta^-(C)} x_a \geq 2 \qquad \forall C \subset V, C$$

$$x_a \in \mathbb{N}_{\geq 0} \qquad \forall a \in A$$

Третє обмеження також є необхідним, оскільки воно гарантує, що кожна станція з’єднана з усіма іншими станціями. Тобто кожен елемент множини станцій повинен мати хоча б по одному зв’язку в кожному з напрямків.

Також потрібно розглянути функцію t, яка відповідає за розрахунок ваги кожної з дуг. Таких функцій буде дві:

– для розрахунку часу на перегоні між двома станціями;

– для розрахунку на перехід між двома лініями.

У першому випадку:

$$t = K + F, \quad K \in N \geq 0, F \in N \geq 0,$$

де за K ми позначимо час, який витрачає потяг на подолання перегону між станціями у хвилинах, а за F – інтервал руху поїздів в певний час доби, зазначений у хвилинах.

У другому випадку:

$$t = J + F, \quad J \in N \geq 0,$$

де J – це час на перехід між станціями. При тому слід зазначити, що він може відрізнятися в залежності від того, в якому напрямку необхідно перейти зі станції на станцію.

3.3 Дослідження генетичних алгоримів для вирішення проблеми станцій

Ідея генетичних алгоритмів вперше була представлена Джоном Холланом у 1960 році як техніка оптимізації, яка наближає результат до оптимального рішення. І так як дана проблема відноситься до класу NP-повних задач, тобто ми не маємо поки що іншого способу знайти найоптимальніше рішення, чим перебрати всі можливі варіанти, варіант спробувати використати генетичні алгоритми для рішення цієї задачі є цілком перспективним [3].

Також в теорії ця задача є трансобчислювальна. для кількості вершин більше за 66 [19]. Але й в меншому графі кількість можливих рішень може бути дуже великою для того, щоб розраховувати оптимальні маршрути в реальні строки. Так для прикладу на таблиці 3.1 зображено кількість можливих рішень для кожного з метрополітенів України.

З таблиці видно, що кількість можливих рішень навіть для Харківському метрополітену дуже велика. Хоча він майже в два рази менший за Київський. Якщо взяти комп'ютер, який здатний аналізувати по 1×10^{23} рішення в секунду, то на розрахунок оптимального шляху знадобиться щонайменше 84 роки.

Таблиця 3.1 – Кількість станцій та можливих рішень проблеми станцій для Харківських метрополітенів

Назва метрополітену	Кількість станцій	Кількість можливих рішень проблеми станцій
Харківський	30	2.6525286×10^{32}
Київський	52	8.0658175×10^{67}

Отже, варіант застосувати алгоритм, який би наблизив рішення до оптимального досить доречно в даній ситуації. Тобто необхідно використати алгоритм, який би на кожній наступній ітерації знаходив все більш і більш оптимальні рішення, при цьому була б можливість зупинити його тоді, коли рішення буде здаватися достатньо оптимальним.

На рисунку 3.4 зображено загальну блок схему генетичного алгоритму.

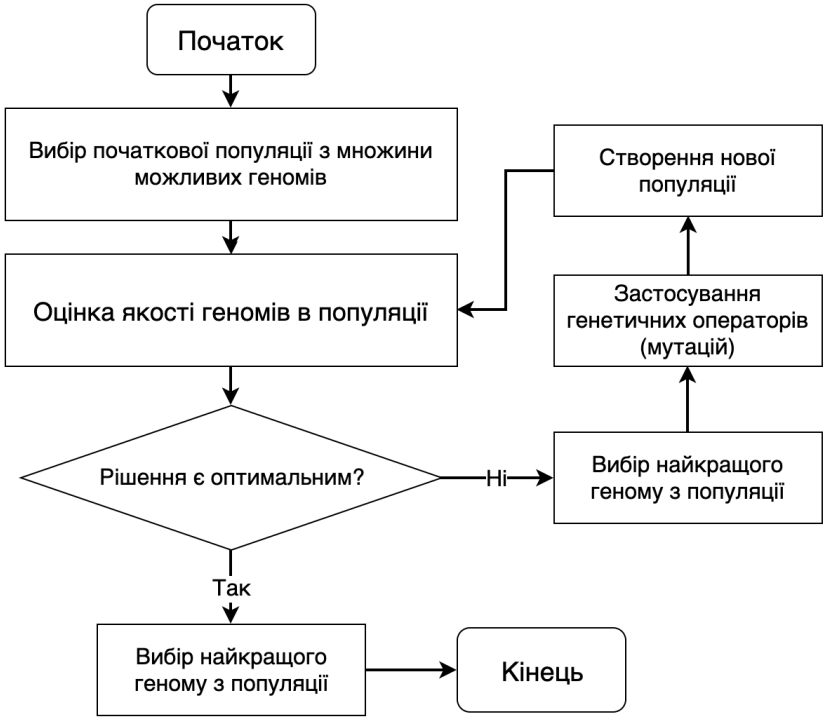


Рисунок 3.4 – Блок-схема генетичного алгоритму

Ми можемо позначити задачу оптимальності, як задачу пошуку функції $f(x_1,x_2,x_3,\ldots,x_n)$, яка в теорії генетичних алгоритмів має назву функції пристосованості. Ця функція дозволяє виявити найбільш пристосованих особин популяції для розмноження та найменш пристосованих, які будуть в майбутньому видалені в процесі роботи алгоритму для збільшення пристосованості нового покоління.

Потрібно, щоб на області визначення функції виконувалась нерівність $f(x1,x2,x3,\ldots,xn) \geq 0$. Така область є обмеженою. Параметри функції записуються у вигляді рядків, що складаються з бітів. Рядок, який був отриманий шляхом конкатенації рядків називається особиною [10]:

1010 10110 101 . . . 10101

$| x_1 | x_2 | x_3 | \ldots | x_n |$

Генетичний алгоритм є досить універсальним, так як тільки функції пристосованості та кодування рішень залежать від умов поставленого завдання. При виконанні алгоритму враховуються наступні правила: початкова популяція вибирається випадково, кількість її особин залишається незмінною, кожна з

особин записується як рядок з певною довжиною кодування [10]. Кожен крок алгоритму можна розбити на три етапи:

а) пропорційний залежно від пристосованості відбір особин поточного покоління, які мають право дати потомство, і формування з них проміжної популяції;

б) проміжну популяцію ділять на пару і з деякою імовірністю схрещують, в результаті в нове покоління потрапляє сама пара або її нащадки. Нащадки формуються з відсічених частин батьківських рядків, розділених певною точкою;

в) відбувається мутація отриманого покоління, що не допускає передчасної збіжності. Кожен біт особини з імовірністю не більше 1% записується як протилежний початковому [10]. В якості заздалегідь заданих критеріїв отримання оптимального рішення може бути певне число зміни поколінь або сходження популяції. Іншими словами умова виконується, якщо всі рядки є майже ідентичними і знаходяться в області екстремуму. Рішенням алгоритму буде особина з найбільшим значенням функції пристосованості [10].

Існують кілька різних реалізацій класичного генетичного алгоритму в залежності від підходу до етапів схрещування і мутації. І деякі з них дали хороші показники при тестуванні.

Дослідження, представлене в статті «Comparison of Algorithms for Solving Traveling Salesman Problem» [23] показує результати порівняння трьох методів: алгоритму найближчого сусіда, генетичного та жадібного алгоритмів. Як приклад було використано карту США, яка нараховує 20 точок (представлено на рисунку 3.5).

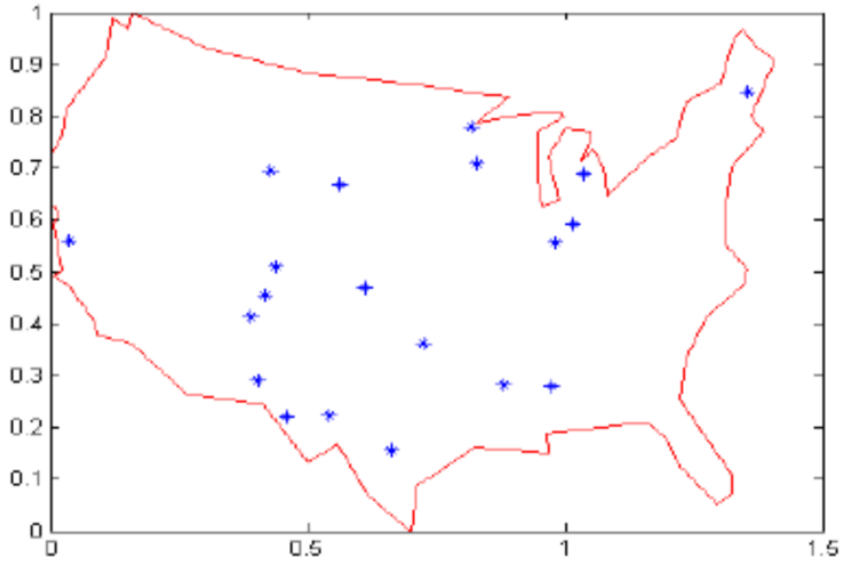


Рисунок 3.5 – Карта США з позначеними містами

Оптимальне рішення має загальну довжину маршруту 4616 км. Це рішення має високу складність і складається з найбільшого числа ітерацій. Вирішуючи ту ж задачу методом найближчого сусіда, був отриманий маршрут, показаний на рисунку 3.6. Його довжина становить 15800 км.

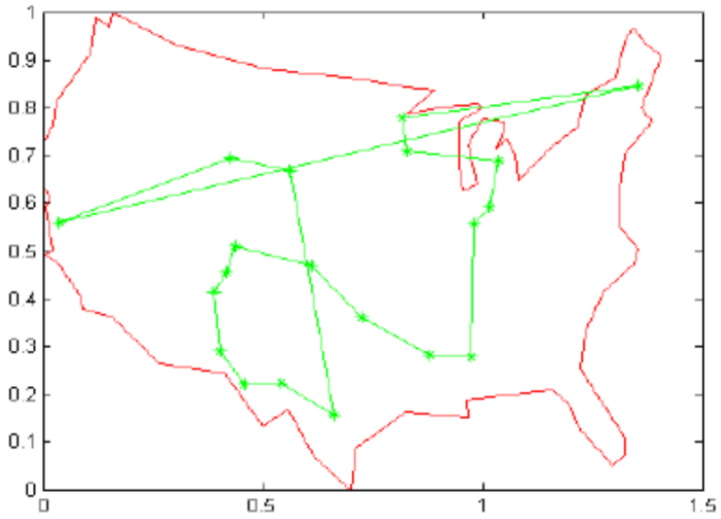


Рисунок 3.6 – Результат, отриманий за допомогою методу найближчого сусіда

Рішення TSP для 20 міст з використанням генетичного алгоритму показано на рисунку 3.7. Хоча було проведено більше ітерацій, ніж в попередньому прикладі, даний алгоритм знайшов більш оптимальний маршрут рівний 11900км.

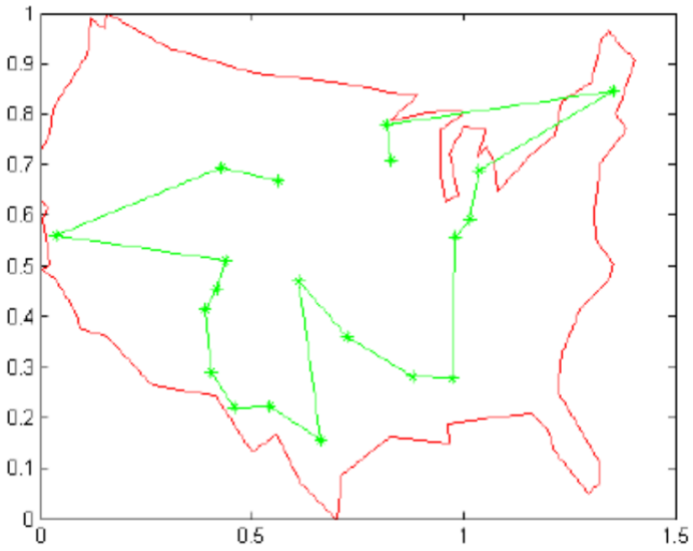


Рисунок 3.7 – Результат, отриманий за допомогою генетичного алгоритму

Той же самий приклад TSP було вирішено з використанням жадібного алгоритму. Результат показаний має довжину 12900км. Хоча це рішення більш оптимальне в порівнянні з алгоритмом найближчого сусіда, у нього більш висока складність і час виконання. Генетичний алгоритм показав себе як найефективніший метод вирішення TSP з невеликим обсягом даних.

Подібні порівняння було наведено для задач великої розмірності. Результати дослідження, проведені на 2,2 GHz 16GB of 1600MHz DDR3L SDRAM Intel Core i5 MacBook і включають в себе порівняння таких параметрів як довжина оптимально шляху в кілометрах, витрачений час в секундах і кількість ітерацій, представлені в таблицях 4.2 та 4.3.

Дослідження вище наочно показує, що при кількості міст в задачі комівояжера близькому до 100, генетичний алгоритм сильно програє жадібному за всіма параметрами. При аналізі TSP с 1000 вихідними містами генетичний алгоритм дуже неефективний.

Таблиця 3.2 - Порівняння алгоритмів для вирішення TSP на 100 вершинах

Алгоритм	Довжина оптимального	Затрачений час (сек)	Кількість ітерацій

	маршруту (км)		
Найближчого сусіда	26664	2.5	100
Генетичний	225479	45	10000
Жадібний	23311	0,07	18

Таблиця 3.3 - Порівняння алгоритмів для вирішення TSP на 1000 вершинах

Алгоритм	Довжина оптимального маршруту (км)	Затрачений час (сек)	Кількість ітерацій
Найближчого сусіда	83938	95.2	1000
Генетичний	282866	468	10000
Жадібний	72801	127	151

Отже, головним недоліком генетичного алгоритму є відсутність гарантії, що отримане рішення є оптимальний, як і його знаходження за більш-менш оптимальний час. У той же час генетичні алгоритми показує високу ефективність в задачах з невеликим кількістю вершин. У ситуаціях, коли в задачах великої розмірності відсутня упорядкованість вступних даних, генетичний алгоритм є єдиною альтернативою алгоритму повного перебору [22]. Головна його перевага в тому, що його можна використовувати для вирішення складних неформалізованих проблем для яких не існує окремих методів вирішення, що дозволяє йому ефективно вирішувати нестандартні завдання [2].

Таким чином, враховуючи велику варіативність генетичних алгоритмів, необхідно визначитися з основними складовими цього алгоритму, які б відповідали поставленій задачі. Потрібно визначитися зі способом запису геному, мутаціями та способами схрещування особин. Також необхідно експериментальним шляхом знайти найвдалішу комбінацію мутацій, коефіцієнти випадковості та інші модифікації алгоритму, які б дозволили отримати найоптимальніший маршрут.

3.4 Розробка генетичного алгоритму для вирішення проблеми станцій

Для застосування генетичного алгоритму потрібно описати основні параметри та функції, які буде застосовано в процесі виконання програми. До основних параметрів належать: популяція, особина та хромосома. Кожен з цих параметрів повинен бути представлений чимось в предметній галузі.

Після визначення представлення особини, геному та хромосоми потрібно сформулювати основні функції генетичного алгоритму: функцію відбору, пристосованості, схрещування та функцію мутації. Ці функцію дадуть змогу взаємодіяти з параметрами алгоритму, змінювати їх, порівнювати та відбирати найкращих. Маючи ці складові, ми маємо змогу запустити роботу алгоритму та спостерігати за еволюцією.

Як вже було відмічено, потрібно визначитися з представленням особин, популяції та генів.. Їх взаємозв’язок представлено на рисунку 3.8.

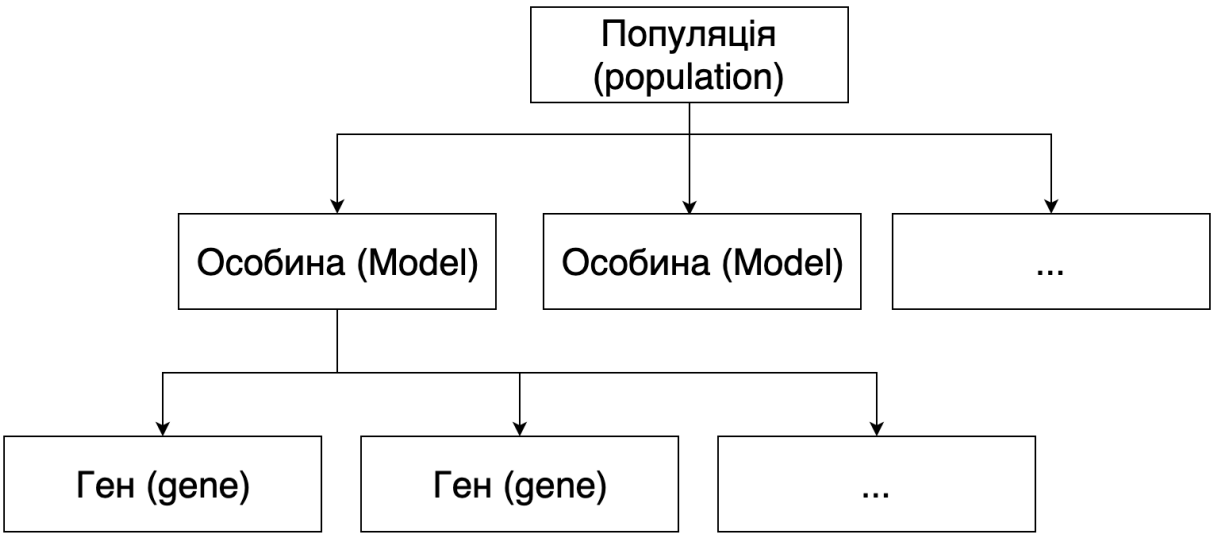


Рисунок 3.8 – Взаємозв’язок основних параметрів генетичного алгоритму

Кожне рішення проблеми буде представлено деяким вектором, який в цьому алгоритмі буде грати роль геному або, інакше кажучи, набором хромосом. Таке рішення буде називатися особиною. Набір особин буде називатися популяцією.

Першим кроком, як було визначено, буде опис рішення у вигляді вектору. Цей крок є дуже важливим, тому потрібно обґрунтовано підійти до вибору способу представлення геному. На перший погляд можна описати рішення як набір всіх станцій в порядку їх слідування в маршруті. Але таке рішення приховує декілька неприємних моментів. По-перше, при такому підході ми отримаємо вектори різної довжини, які буде дуже складно схрещувати між собою, а по-друге, багато векторів буде невалідними, тобто такими, які не дозволять відвідати всі станції. В результаті такого вибору в множині геномів кількість можливих кандидатів на звання оптимального геному буде дуже маленькою. В результаті кожен з геномів потрібно буде окремо перевіряти на валідність, що займе додатковий час та збільшить складність алгоритму. Схематичне представлення геному в такому вигляді зображено на рисунку 3.9.

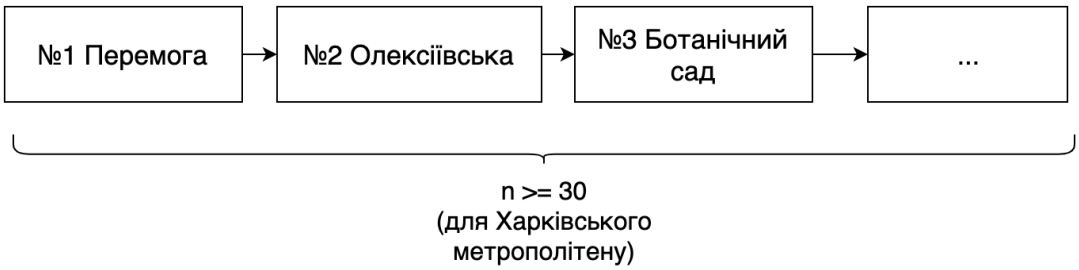


Рисунок 3.9 – Представлення геному у вигляді послідовності станцій в рішенні

Іншим рішенням буде взяти на роль геному перестановку множини станцій метрополітену. Для Харківського метрополітену це 30. Тобто такий вектор довжини 30, в якому кожна станція зустрічається лише один раз. Але при такому підході поряд може опинитися пара станцій між якими немає прямого маршруту. Для вирішення цієї проблеми ми можемо перейти до повного графу. Для цього потрібно буде розрахувати найкоротші маршрути між кожною парою станцій. Складність такого розрахунку досить висока O(n^3), але у випадку з метрополітенами, де кількість станцій не дуже висока, зробити такі розрахунки

можна дуже швидко. Для Харківсього метрополітену це всього лише 27000 варіантів. Схематичне зображення такого геному показано на рисунку 3.10.

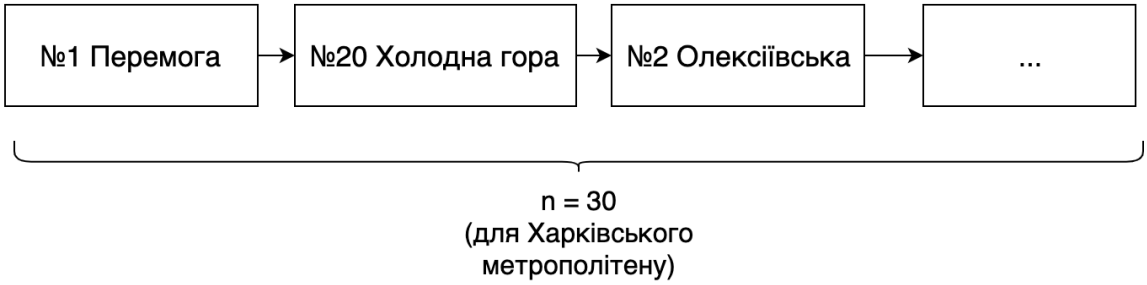


Рисунок 3.10 – Представлення геному у вигляді перестановки всій станцій

Також важливо відмітити, що розрахунок оптимального шляху між двома станціями може залежати від часу доби. Так, оптимальний маршрут в години пік може виявитись неоптимальним ближче до закриття метрополітену. Тому потрібно розрахувати такі графи для кожного періоду, коли інтервали руху незмінні.

Також, потрібно сформулювати функцію пристосованості цього геному, тобто критерій, по якому буде порівняно оптимальність геномів. Функцією пристосованості має бути цільова функція, яку можна застосувати в ролі порівняльного показника якості. Показником якості повинна бути певна цілочисельна характеристика, яку можна отримати в ході обчислення пристосованості хромосом.

В даній роботі таким критерієм є час, який буде затрачено на подолання даного маршруту. Якщо для одного вектора час менший, чим для іншого, то за функцією якості він буде вважатися більш оптимальним.

Таким чином, нехай $G_i = \{g_{ik}\}$, де $k = 1 \dots n$ — буде хромосомою і-й особини, де g_{ik} значення k-го гена на і-й особині, а n — кількість генів в хромосомі (у випадку з Харківським метрополітеном це 30). Тоді при вирішенні задачі мінімізації, тобто пошуку шляху, час на подолання якого буде мінімальним, ми можемо сформулювати умову:

$$f_i = f(G_i) < f_j = f(G_j)$$

Якщо значення функції пристосованості для хромосоми G_i буде меншим за значення функції для хромосоми G_j, то таку хромосому можна вважати більш пристосованою. Таким чином, чим менше значення має цільова функція, тим краще дане рішення.

Наступним кроком буде встановлення правил мутацій та схрещування геномів. Тобто потрібно визначити дані функції. Ці функції повинні вирішувати деякі проблеми. В першу чергу геноми повинні правильно сходитись між собою, а також потрібна можливість вибратись з локального оптимуму [6]. Було розглянуто типові мутації, які використовують в генетичних алгоритмах. Вони в більшості базуються на випадковостях, тому при їх виборі потрібно буде підбирати оптимальні коефіцієнти. Також, при виборі мутацій необхідно звертати увагу на те, коли саме можна її застосовувати. Деякі мутації краще проявляють себе спочатку роботи алгоритму, а деякі більш ефективні під кінець, коли отриманий результат дуже близький до кінцевого. Великою проблемою в пошуку оптимальних шляхів є потрапляння в локальний оптимум [21]. Тому потрібні мутації, які зможуть дати шанс вибратись з нього та продовжити пошук. Таким чином до списку можна додати наступні правила:

– випадкове перетасування випадкового інтервалу генома. Така мутація є швидкою, але не дуже акуратною. Є сенс використовувати її на ранніх етапах еволюції, та на фіналі вона з дуже низькою ймовірністю може поліпшити отримане рішення;

– дзеркальне відображення випадкового інтервалу генома. Ця мутація може бути корисною для виходу з локального оптимуму. При цьому є високий шанс зберегти якість маршруту в контексті задачі;

– перестановка K пар випадкових значень в геномі місцями. Це досить непогана мутація, але повільна. Може бути ефективною ближче до завершення еволюції, але з самого початку має дуже невелику ефективність. З локального оптимуму з такою мутацією вибратися дуже складно;

– схрещування або «crossover» двох окремих геномів. При схрещуванні будемо обирати точку на геномі, розрізатимемо по ній геноми та мінятимемо їх

місцями. Після такого є ймовірність отримати невалідний маршрут, тому всі втрачені вузли потрібно буде додати в випадкові місця геному.

Схема мутацій та схрещування представлена на рисунку 3.11.

Рисунок 3.11 – Схема застосування мутацій та схрещування

В результаті на кожній ітерації ми застосовуємо ті чи інші мутації на деякій множині найкращих особин і формуємо з їх нащадків нове покоління.

Функція формування нового покоління буде базуватися на відборі деякої частини покоління, яка пристосована краще всього. Це стандартний прийом для функції відбору в генетичних алгоритмах [3]. Відношення тих, хто виживе до тих, хто загине, буде задано заздалегідь.

Спосіб відбору буде також задано заздалегідь, так як різні способи можуть показати різні результати. Для рішення даної задачі підходить турнірна селекція та метод ранжування. При таких способах ми зберігаємо кращих особин та перестраховуємо себе від попадання в локальний оптимум, що є однією з найбільших небезпек в ході пошуку рішення. Також, гарною практикою є заміна втраченої частини популяції новими особинами, повністю випадковими маршрутами. Таке рішення зумовлюється проблемою, яка в еволюції називається інбридинг, тобто схрещування близькоспоріднених організмів, що призводить до виродження покоління. Це досить розповсюджена практика в формуванні функції

відбору [6]. Таким чином ми ще раз перестрахуємо себе від попадання в локальний оптимум.

Отже, враховуючи математичну модель, обрані параметри та мутації ми можемо сформулювати блок-схему використання генетичного алгоритму, яка буде включати сам генетичний алгоритм та його модифікації та надбудови. Блок-схема повного алгоритму програми представлена на рисунку 3.12.

Таким чином, отриманий алгоритм повністю готовий до реалізації. При реалізації потрібно обов’язково врахувати те, що ймовірності p’, p’’, p’’’, p’’’’, p’’’’’ повинні мати змогу змінюватися безпосередньо в процесі роботи самого алгоритму. Тобто, користувач повинен сам мати змогу вирішувати, коли зупинити роботу алгоритму, та розрахувати час, на зміну значення ймовірностей мутацій та схрещування.

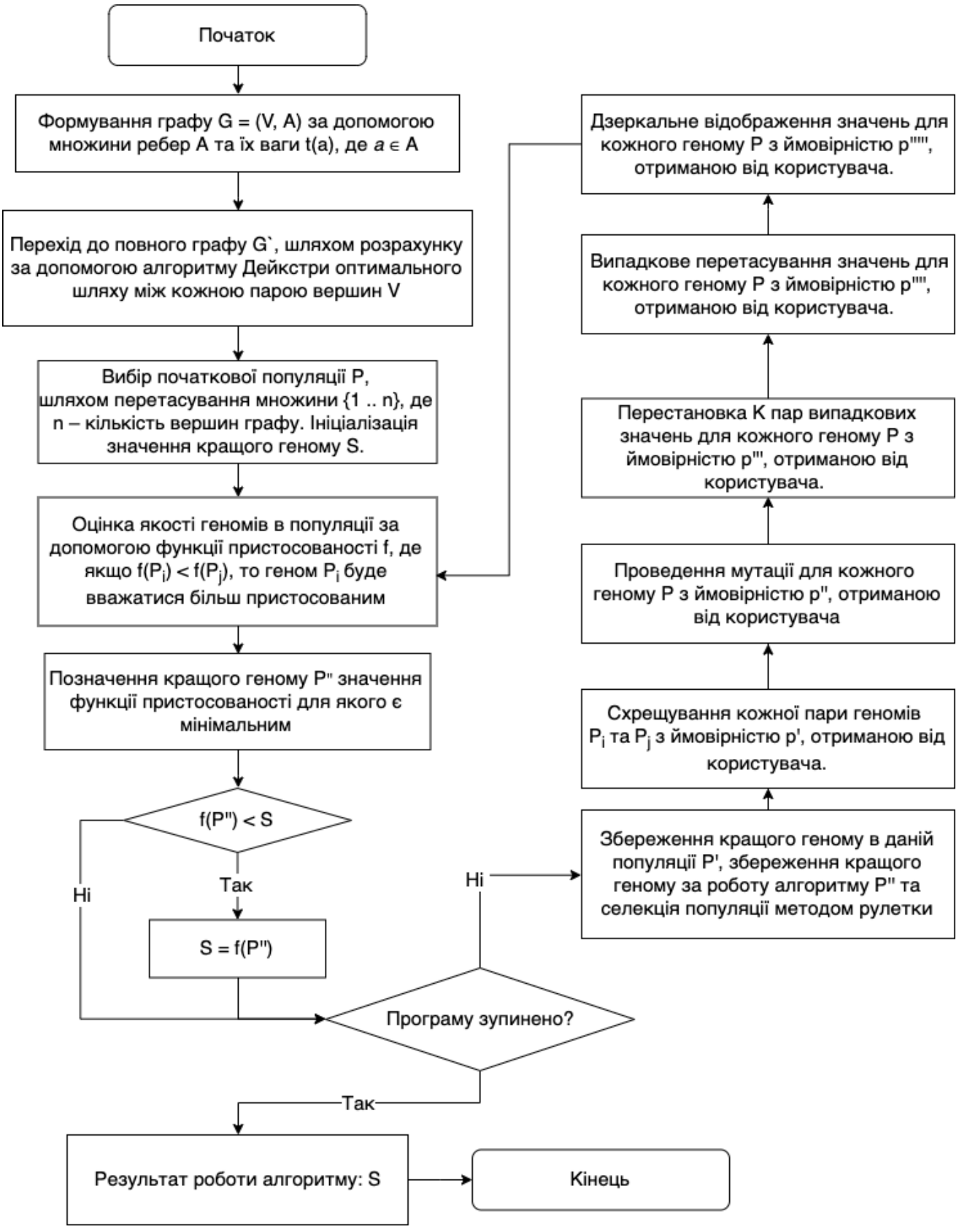


Рисунок 3.12 – Загальна схема алгоритму

3.5 Проектування бази даних

Для зберігання даних про лінії, станції, інтервали, виходи та переходи було вирішено розробити реляційну базу даних.

Реляційна база даних - це сукупність взаємопов'язаних таблиць, кожна з яких містить інформацію про об'єкти певного типу. Рядок таблиці містить дані про один об'єкт, а стовпці таблиці описують різні характеристики об'єктів - атрибути. Рядки таблиці мають однакову структуру, вони складаються з полів, що зберігають атрибути об'єкта. Кожне поле стовпця описує тільки одну характеристику об'єкта і має строго певний тип даних [16].

Отже необхідно виявити основні об’єкти системи та їх атрибути. Основними сутностями предметної галузі є «Станція» та «Лінія». Кожна лінія має багато станцій. Крім того, потрібно ввести допоміжні сутності «Перехід», «Інтервал» та «Вихід».

Схему взаємозв’язку об’єктів наведено на рисунку 3.13.

На основі аналізу та моделювання предметної області було розроблено схему базу даних і приведено її до третьої нормальної форми.

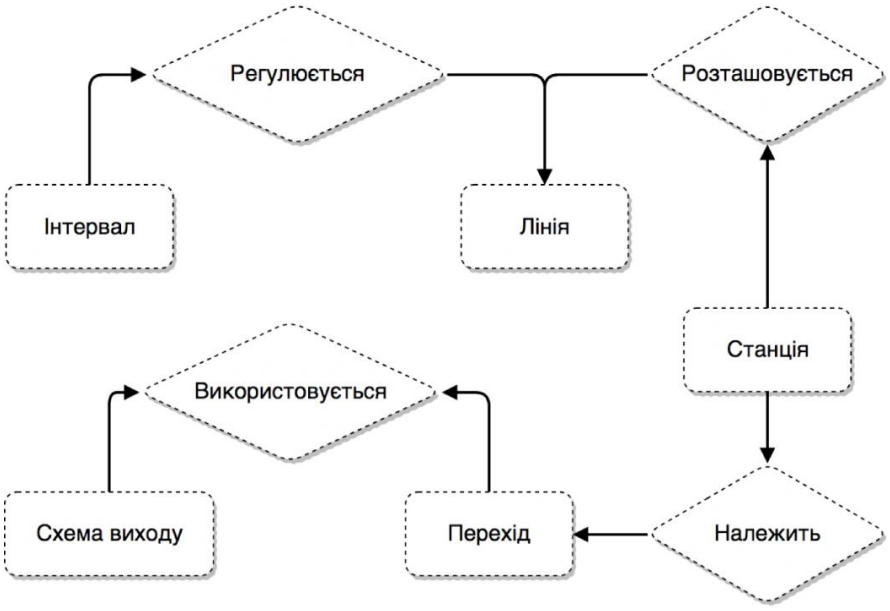


Рисунок 3.13 – Схема взаємозв’язку об’єктів

Схему БД наведено на рисунку 3.14.

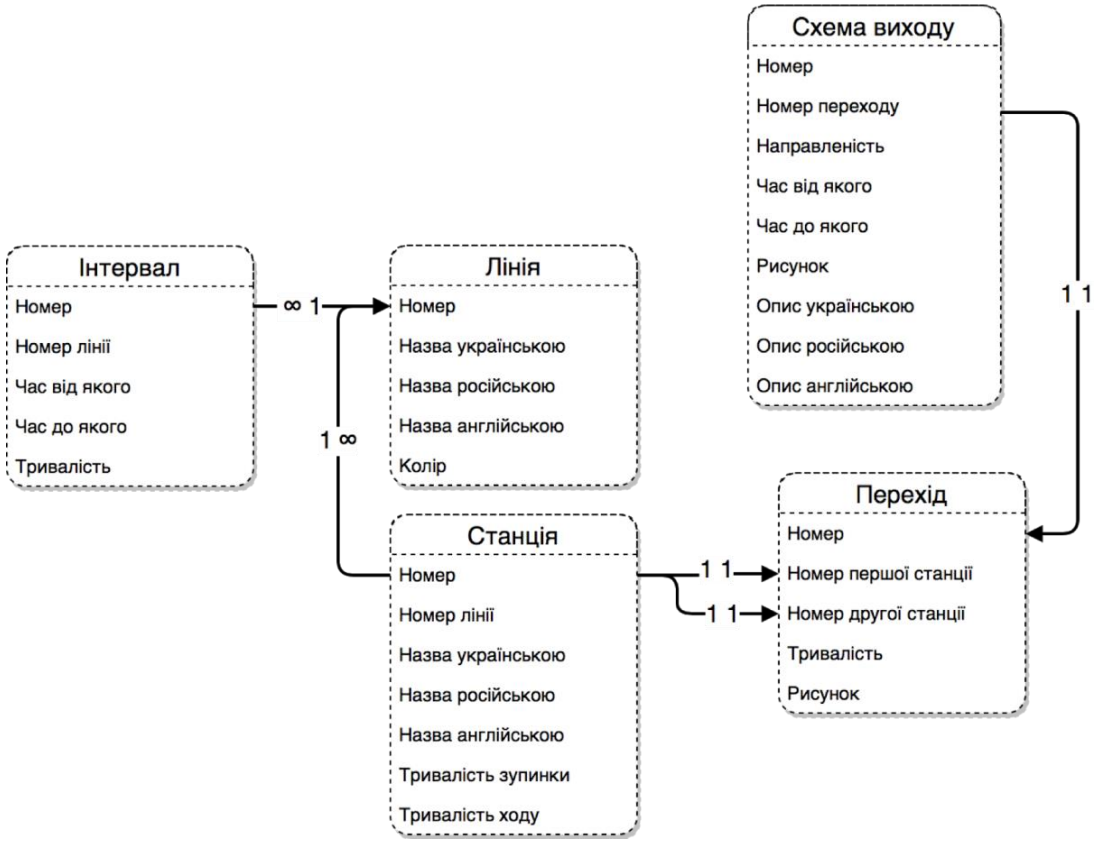


Рисунок 3.14 – Схема бази даних

База даних містить всю інформацію, яку було розглянуто при виборі об'єктів предметної області та їх параметрів. Дані в таблицях залежать винятково від основного ключа, а поля, що залежать від будь-якого іншого поля були винесені в окрему таблицю. З цього можна зробити висновок, що схему приведено до третьої нормальної форми [17].

3.6 Вибір методу оцифрування графу метрополітена

Для розробки даної програмної системи необхідні заповнити розроблену у підрозділі 4.5 базу вхідними даними для певного метрополітена. Розділимо задачу збору даних на три етапи:

– оцифрування графу метро;

– доповнення графу даними;

– збір даних про оптимальні виходи та переходи.

Якщо з останнім пунктом все зрозуміло – зібрати дані про оптимальні вагони для виходу з електропоїзду можна просто покатавшись у метрополітені. При тому відсутність цих даних не сильно вплине на фінальний результат розрахунку повного маршруту з усіма станціями. Таким чином цей пункт будемо вважати не обов’язковим та не будемо розглядати детально.

Щодо оцифрування графу та доповнення його різною корисною інформацією, то тут дещо складніше. Досить очевидно, що вершинами у графа будуть станції, а ребрами – переходи та перегони. Граф буде неповний та орієнтований, бо електропоїзди не завжди рухаються за однаковий час в різні сторони.

Інформація про час, який витрачають електропоїзди на подолання шляху між станціями, та скільки вони проводять часу на станції не є таємною. Її можна зібрати з аналогів розроблюваної програмної системи. Для того щоб розробити прототип системи цього досить, але як було з’ясовано в підрозділі 1.2, ці системи використовують дуже усереднені дані та нехтують багатьма чинниками. Тому для серйозних розрахунків ці дані не підходять. Потрібні більш точні та реальні дані, які відповідають дійсності.

Про збір даних вручну можна говорити тільки в межах Харківського метрополітену, але для тестування рішення потрібно буде розрахувати оптимальний маршрут для інших метрополітенів. Ручний збір даних в таких метрополітенах буде дуже важким та неоптимальним рішенням.

Існує кілька популярних програм, які не було розглянуто серед аналогів, бо їх функціональність не зовсім відповідає даній програмній системі, але ці програми можуть знадобитися при зборі даних. Серед таких програмних систем система pMetro та mMetro. Це досить старі настільні програми з народними замірами часу в багатьох популярних метрополітенах. Перша з них перестала підтримуватися розробниками ще в 2011 році, але багато метрополітенів вісім

років тому назад не сильно відрізнялися від тих, що зараз. Наприклад, у Харкові з 2011 року було відкрито лише одну станцію метро.

Дані в програмі pMetro подані не дуже в зручному форматі, але написавши невеликий парсер ми зможемо конвертувати його у json. Зміст файлу з координатами та даними про станції зображено на рисунку 3.15.

Маючи потрібні дані, ми можемо приступити до оцифровування графу метрополітена. Для зручної роботи з даними було вирішено скористатися уже написаною javascript бібліотекою. Було знайдено декілька бібліотек на одному з найбільших веб-сервісів для спільної розробки GitHub. Серед знайдених бібліотек є проекти з MIT ліцензією Algorithms.js та Ngrapg.graph, що дає нам змогу використовувати ці бібліотеки в розроблюваній системі.

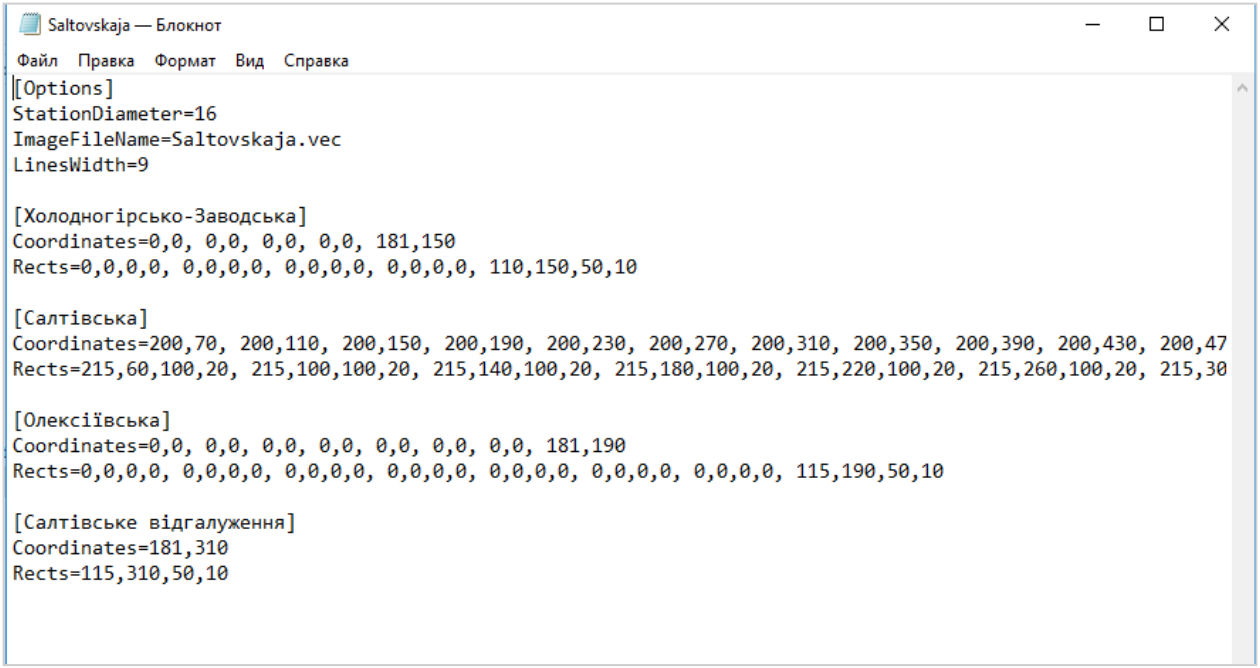


Рисунок 3.15 – Файл з даними для Салтівської лінії

Отже, за допомогою знайдених даних та бібліотеки ми маємо змогу оцифрувати граф для подальшої роботи з ним.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Вибір засобів розробки

При виборі засобів для написання розроблюваної програмної системи потрібно враховувати їх актуальність, легкість підтримки та доречність використання в межах даної розробки. Також, не можна забувати про те, що програмна система працює з SVG. А у цієї технології є свої особливості. Перед початком розробки потрібно було визначитися з мовою програмування на клієнтській та серверній частині системи, бібліотеками, системою збірки та запуску задач, системою управління базами даних (СУБД) та системою для транспіляції коду. Транспіляція в даному випадку потрібна для підтримки застарілих браузерів.

Головною частиною розроблюваної системи є клієнтська частина. Тому для її написання було обрано мову програмування JavaScript. Це досить стандартне рішення, тим більше додаток, написаний цією мовою, можна легко відобразити в будь-якому браузері. Ця мова має дуже велику спільноту в інтернеті та докладну документацію.

Для серверної частини, яка складає невелику частину проекту, було обрано мову PHP. Такий вибір пояснюється тим, що за досить невеликі кошти можна придбати хостинг з підтримкою найновішої версіїї PHP. Ця мова зараз втрачає популярність, але все одно зарекомендувала себе, як надійна для написання серверу. Інші технології для написання серверної частини не так добре підтримуються хостинговими компаніями. Тому було обрано мову програмування PHP.

Також було розглянуто можливість встановлення деяких популярних фреймворків, наприклад, Zend Framework чи Yii Framework. Але їх функціонал дуже перевантажений, як для розроблюваної системи. Тому було прийнято рішення відмовитись від використання будь-якого фреймворку на серверній стороні.

Разом з розглядом можливості використання фреймворку на серверній стороні, було розглянуто можливість використання бібліотек для зручної розробки клієнтської частини. Зокрема, було розглянуто бібліотеку React, як найпопулярніший сучасний фреймворк для розробки фронтенду. У нього є цілий ряд плюсів:

– простота використання та простота підтримки;

– невеликий розмір самої бібліотеки;

– велика спільнота по підтримці React-застосунків, в якою розглянуто дуже багато типових проблем при розробці;

– швидкість роботи в браузері завдяки технології Shadow-DOM;

– Redux-архітектура, побудована на принципах Flux.

Разом з технологією React останні роки набирає популярність VueJS. В нього, на відміну від React є свої переваги:

– простота роботи зі структурою веб-сторінки;

– нативний HTML, що дуже важливо при роботі з технологією SVG;

– спеціальні бібліотеки для роботи з SVG;

– Vuex-архітектура, побудована на принципах Flux.

Більш специфічний підхід до розробки користувацьких систем на клієнтській частині демонструє фреймворк Svelte. Він компілює написаний код в нативний javascript код. Тому в результаті розмір javascript бандлу мінімальний. Також Svelte забезпечує інструменти для зручної роботи з SVG зображеннями.

Таким чином було прийнято рішення відмовитися від популярних фреймворків типу React чи VueJS та взяти за основний інструмент Svelte та нативний javascript. Це пояснюється тим, що незважаючи на популярність та велику кількість готових рішень, ці бібліотеки мають багато можливостей, які не потрібні в даній системі, але не завантажувати їх реалізацію на клієнтській стороні не має можливості. При тому Svelte надає можливості використання принципів реактивності без їх самостійної реалізації. Таким чином, можна запрограмувати динамічне змінення параметрів алгоритму. При цьому було вирішено використати всі переваги нового стандарту EcmaScript ES8 за

допомогою траспайлера коду BabelJS, який транспілює код до старого стандарту, що підтримує більшість браузерів.

Для роботи з базою даних було обрано СУБД MySQL. Це стандартна система управління базами даних, яка підтримується на хостингу разом з мовою програмування PHP.

Систем для збірки проекту та запуску задач на сьогодні представлено дуже багато. Кожна з них має свої переваги та недоліки. Серед них:

– Gulp (простий в налаштуванні, велика спільнота);
– Grant (швидко працює);
– Webpack (дуже функціональний, має багато модулів).
– RollUp (аналізує використання коду)

З цих чотирьох було обрано систему для збірки RollUp, яка має змогу забезпечити мінімальну кількість коду, що не буде використовуватися в результуючому файлу.

Середовище написання коду — VSCode. Це застосунок від компанії Microsoft, який має дуже багато зручних плагінів, що полегшують та пришвидшують роботу з кодом. Серед аналогів також було розглянуто такі редактори, як Atom та Sublime Text. Але найзручнішим виявився VSCode.

Для збірки пакетів та контролю їх версій було вирішено використати звичну технологію npm. Аналогів у цієї технології небагато, серед них є Yarn, але ця система ще досить нова та незвична.

Отже, для розробки було вирішено використовувати такі технології:

– JavaScript по стандарту ES8;
– Фреймворк Svelte для роботи з реактивністю;
– танспілятор коду BabelJS;
– PHP 7 версіїї для розробки сервера та СУБД MySQL;
– RollUp для збірки та запуску задач;
– npm для контролю пакетів;
– VSCode для написання коду.

4.2 Опис програмної реалізації алгоритму

Реалізацію алгоритму розділено на декілька файлів. Основний файл під назвою PathCalculate.svelte – це Svelte-компонент в якому реалізовано роботу з параметрами алгоритму через двухсторонню прив’язку значень таким чином:

```
<LabelItem
      step="0.01"
      min="0.01"
      max="1"
      bind:value={mutationProps.mutationProbability}>
            Mutation probability:
</LabelItem>
```

Таким чином реалізовано прив’язку ймовірності мутації в алгоритмі та задано мінімальне, максимальне значення та значення кроку для поля вводу.

За запуск алгоритму відповідає окрема функція:

```
function GAStart() {
      initData();
      GAInitialize();
      mainInterval = setInterval(render, intervalDuration);
}
```

В першому рядку всім змінним присвоюються початкові значення. Другий рядок обирає початкову популяцію та обирає краще значення в популяції. Третій рядок відповідає за ініціалізацію головного інтервалу алгоритму та встановлює йому обране користувачем значення його тривалості.

Функція render, яку було передано, як callback-function в інтервал відповідає за створення нового покоління. Це створення відбувається таким чином:

```
function GANextGeneration() {
      currentGeneration++;
      population = selection(
            population,
            currentBest,
            best,
            values,
            populationSize
      );
      population = crossover(
            population,
            dis,
```

```
            populationSize,
            crossoverProbability
        );
        population = mutation(
        population,
        populationSize,
        mutationProps
);
        setBestValue();
}
```

В процесі виконання функції збільшується лічильник кількості поколінь та по черзі визиваються функції селекції, схрещування та мутації популяції. Після чого серед всієї популяції обираються краще значення за допомогою функції setBestValue. Треба зауважити, що реалізація функції selection, crossover та mutation дещо відрізняється від інших функцій. Вони реалізовані як чисті функції. Вони приймають всі необхідні для роботи параметри та повертають в якості значення нову популяцію, ніяк не змінюючи початкову популяцію та інші значення. Тобто ці функції працюють лише з локальними змінними.

Мутації та більшість функції алгоритму реалізовано теж як чисті функції. Наприклад мутація з перестановкою пар:

```
export function doMutate(seq) {
  let m, n;
    do {
      m = randomNumber(seq.length - 2);
      n = randomNumber(seq.length);
    } while (m >= n);

    for (var i = 0, j = (n - m + 1) >> 1; i < j; i++) {
      seq.swap(m + i, n - i);
    }
    return seq;
}
```

Та допоміжні функції, які розширюють можливості масиву та допомагають генерувати випадкові числа:

```
Array.prototype.swap = function(x, y) {
  if (x > this.length || y > this.length || x === y) {
    return;
  }
  var tem = this[x];
  this[x] = this[y];
  this[y] = tem;
};
```

```
export function randomNumber(boundary) {
  return parseInt(Math.random() * boundary);
}
```

Одна з найважливіших функцій алгоритму, через яку обчислюється значення для порівняння ефективності маршруту реалізовано як чисту функцію через метод reduce, що дозволяє перебрати всі елементи масиву зі збереженням проміжного значення [18]. Дану функцію реалізовано таким чином:

```
export function evaluate(indivial, dis) {
  return indivial.reduce(
    (sum, element, index, array) =>
( sum += dis[element][array[index - 1]] )
  );
}
```

Отримання даних реалізовано через стандартний метод fetch та найпростішу реалізацію кеша для обмеження повторних запитів до серверу:

```
let cache = {};

export async function getGraph(url, timeOnStation) {
  const res = await fetch(url);
  const data = await res.json();
  return createGraph(data, timeOnStation);
}

export async function getStations(url) {
  const res = await fetch(url);
  const data = await res.json();
  return data;
};

export async function getScheme(url) {
  const res = await fetch(url);
  const data = await res.json();
  return data;
};

export async function getData(variable, func) {
  if (cache[variable]) {
    return cache[variable];
  }
  const result = await func();
  cache[variable] = result;
  return result;
}
```

4.3 Опис інтерфейсу користувача

Весь інтерфейс користувача побудована на одному екрані. В процесі роботи з графом необхідні елементи з’являються внизу сторінки. Інтерфейс має мінімалістичний вигляд. Він позбавлений всіх зайвих елементів, які можуть якось відволікати користувача. Стартовий екран показано на рисунку 4.1.

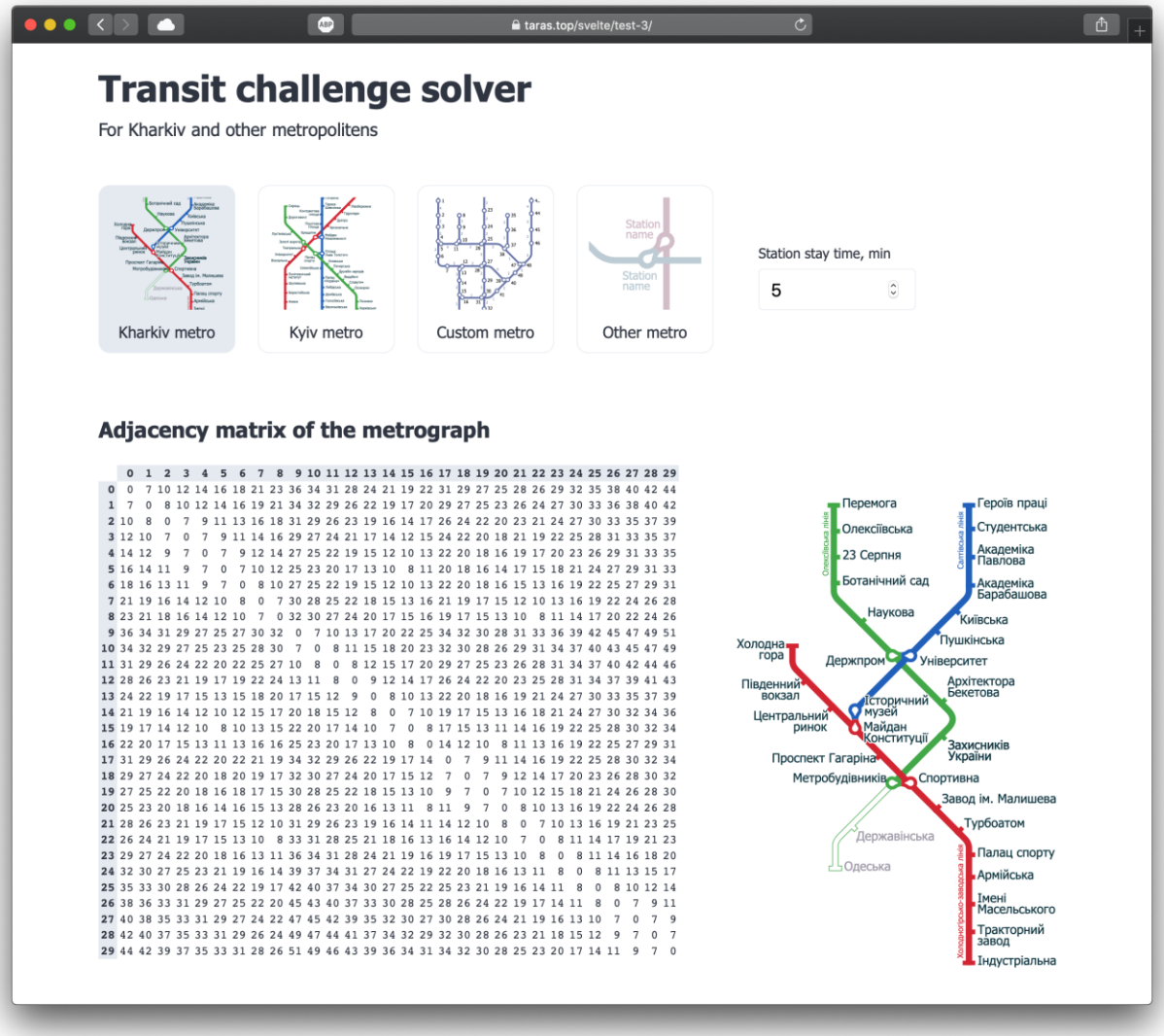


Рисунок 4.1 – Початковий екран програмної системи

На стартовому екрані окрім заголовку розташований вибір доступних метро та графів, а також можливість для завантаження свого графу. Зліва від вибору графа є поле для вводу часу, який необхідно буде провести на кожній станції для зарахування відвідування. Нижче розташоване графічне зображення схеми метро та матриця суміжності для повного графу, побудованого з розрахунком обраного метрополітену та введеного значення кількості необхідних хвилин на станції.

Під матрицею суміжності знаходиться основний блок для розрахунку оптимального маршруту. Цей блок зображено на рисунку 4.2.

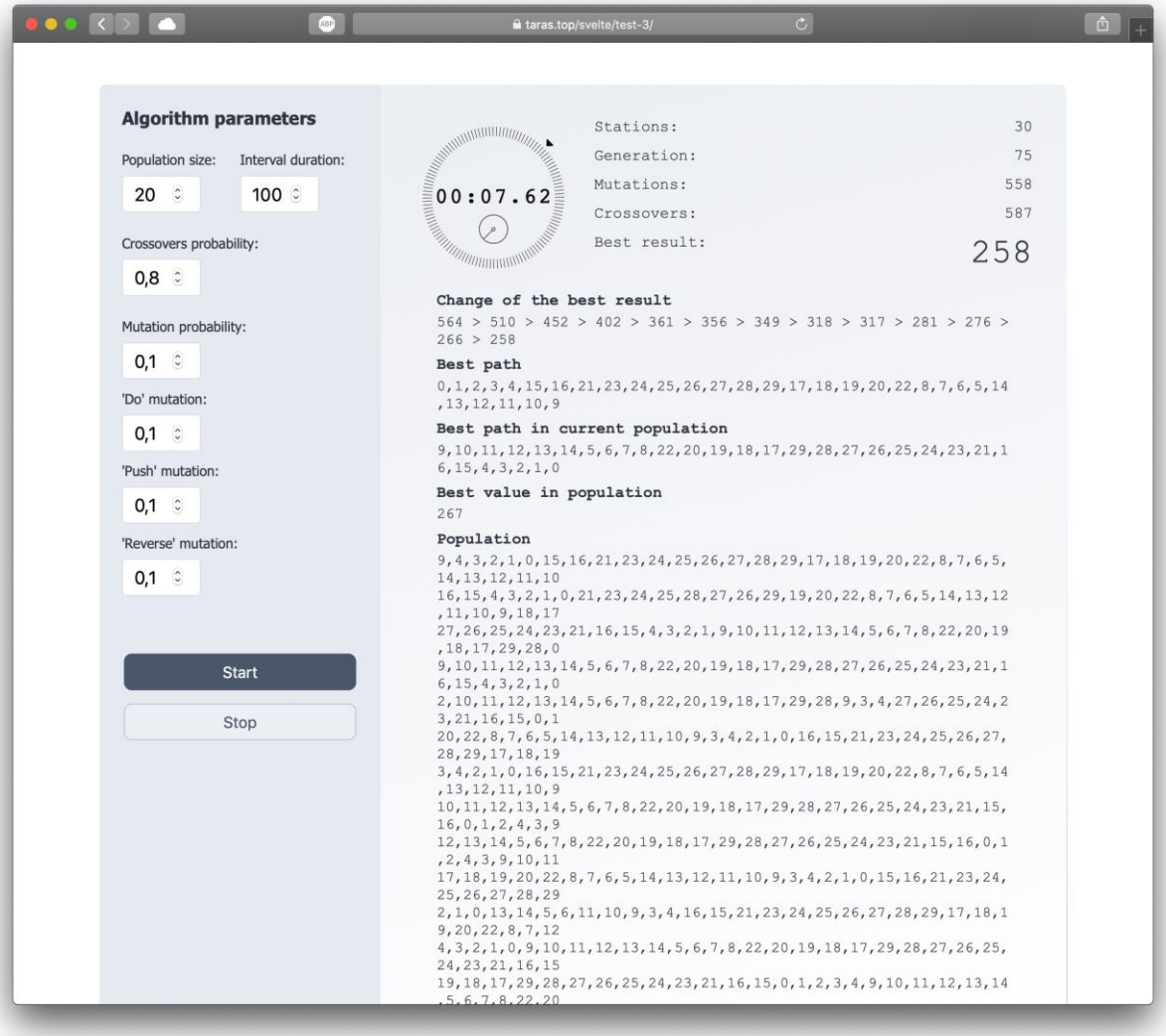


Рисунок 4.2 – Блок для запуску генетичного алгоритму

В лівій частині цього блоку знаходиться колонка з управляючими елементами. Тут є можливість задати основні параметри для роботи з алгоритмом та обрати тривалість інтервалу формування одного покоління. Під параметрами розташовані основні кнопки для початку та зупинки алгоритму. Зліва розташований секундомір для відображення затраченого на обчислення часу. Поряд з ним розташована вся інформація по розрахункам. Інформація динамічно змінюється в процесі роботи алгоритму. Після зупинки роботи алгоритму вкінці блоку з’являються кнопки для збереження отриманого результату (рисунок 4.3). Також передбачена кнопка для очистки збережених даних.

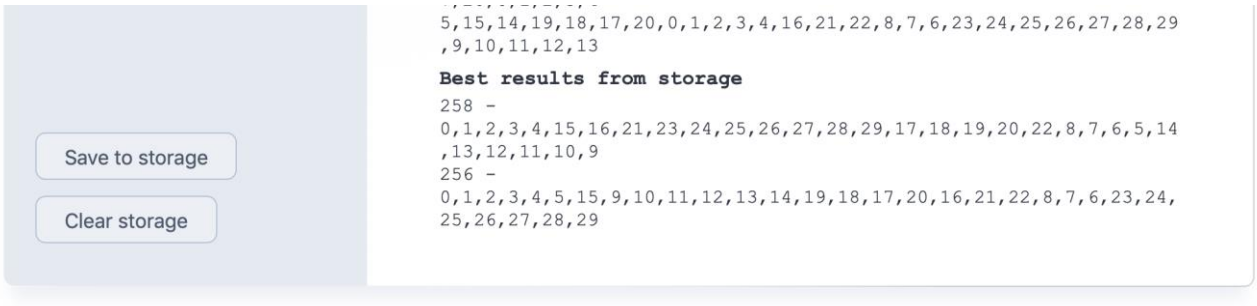


Рисунок 4.3 – Функціонал для збереження результатів

Після розрахунку оптимального маршруту, цей маршрут наглядно відображається на схеми та в списку з зазначенням виходів. Приклад відображення цього списку зображено на рисунку 4.4.

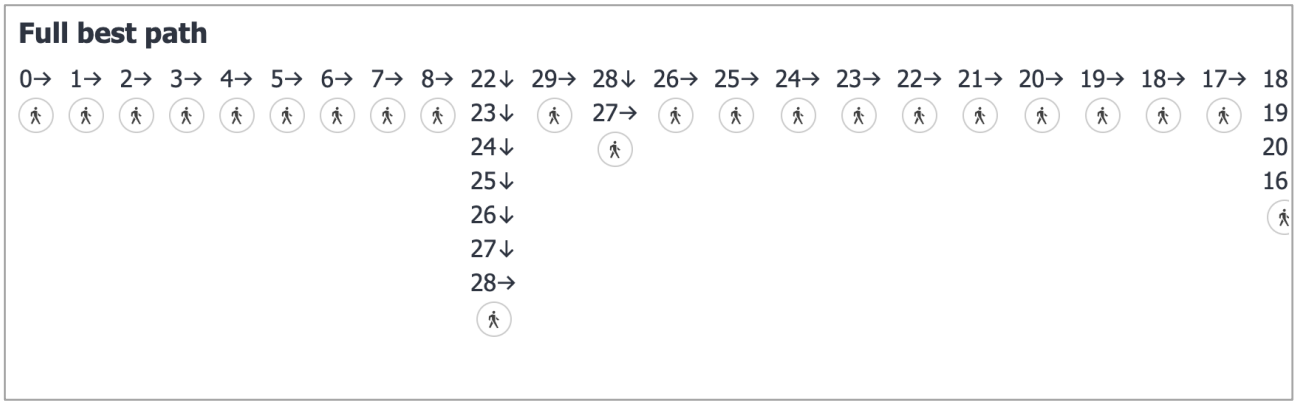


Рисунок 4.4 – Відображення номерів станцій в маршруті з зазначенням виходів

Для більшої наочності цей маршрут відображено у вигляді списку назв станцій в метрополітені. На кожну станцію можна навести курсор та подивитися її розташування на схемі. Також є можливість запустити анімацію маршруту при якій кожну станцію буде по черзі показано на схемі та підсвічено в списку. Екран з прикладом відображення станції на схемі наведено на рисунку 4.5.

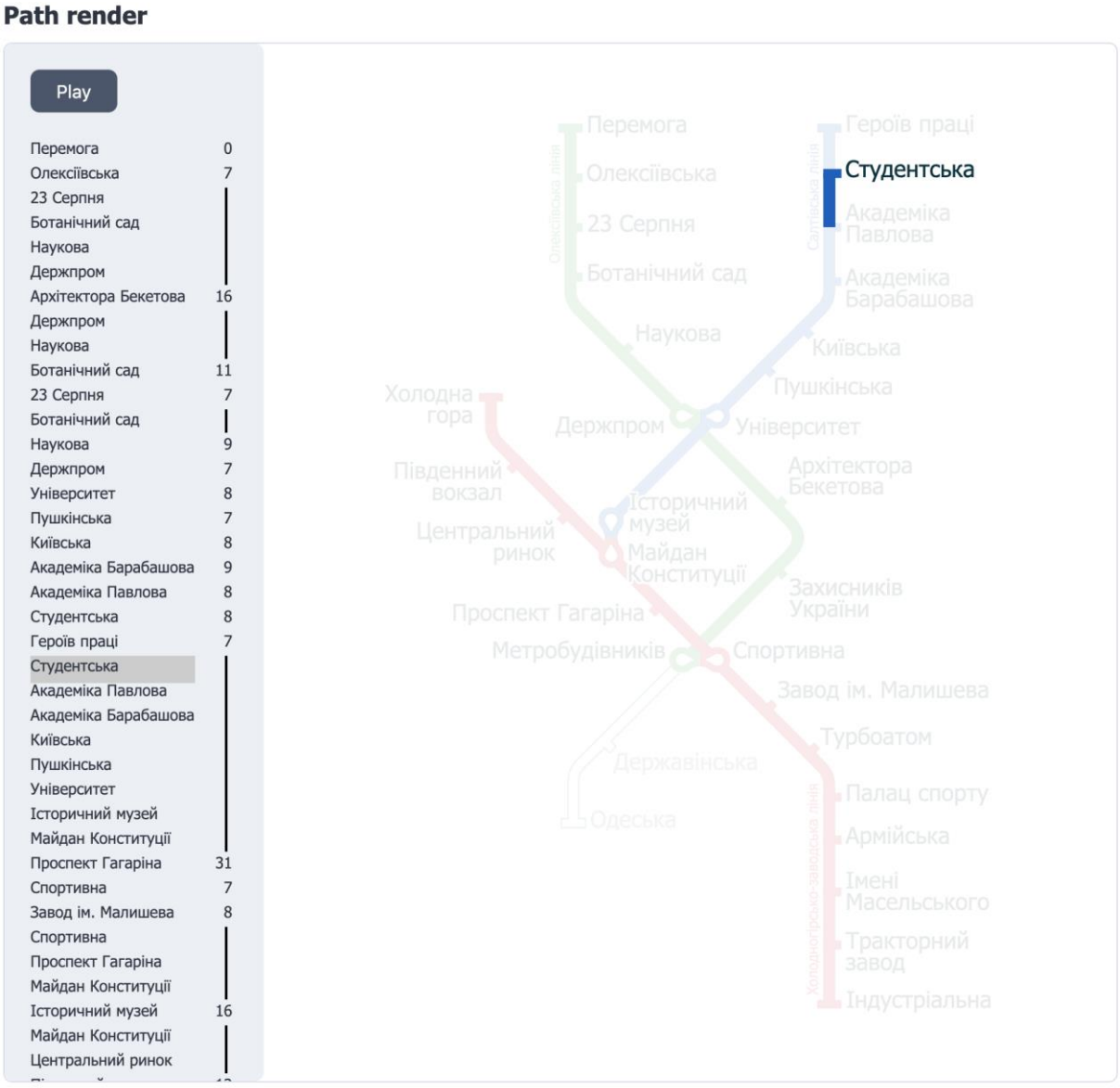


Рисунок 4.5 – Відображення обраної станції на схемі

Відображення потрібних станцій на схемі реалізовано за допомогою накладання двох SVG-зображень один на одного:

```
<div class="svg-render">
```

```
    <img
      class="svg-render__back {isMapActive ? 'active' : ''}"
      alt="metro-image"
      src={metroImage} />

    {#if schemeSVGData}
      <svg
        font-family={schemeSVGData.font}
        viewBox={schemeSVGData.viewBox}
        xmlns="http://www.w3.org/2000/svg"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        class="map {isMapActive ? 'map-active' : ''}">

        {#if schemeSVGData.defs}
          <defs>
            {@html schemeSVGData.defs}
          </defs>
        {/if}

        <g fill={colors.text}>
          {#each stationsPath as station, index (station.id)}
            {#if station}
                <g class="station {showingStation == station.id ?
             'fadein' : ''}">

                <g stroke={colors[station.color]}>
                  {@html station.path}
                </g>

                <g fill={colors[station.color]}>
                  {@html station.stop}
                </g>

                <text style={station.style}>
                  {@html station.text}
                </text>
              </g>
            {/if}
          {/each}
        </g>

      </svg>
    {/if}

  </div>
```

Кожен елемент зроблено адаптивним. Тобто таким, що добре відтворюється на екранах настільних комп’ютерів, ноутбуків та мобільних телефонах.

5 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ

5.1 Планування експерименту

При дослідженні розробленого генетичного алгоритму для вирішення проблеми станцій цілком доречно використовувати графи, побудовані на основі Харківського та Київського метрополітенах. При тому є можливість застосувати даний алгоритм для будь-якого метрополітену, схожого на українські (однакова логіка маршрутів та кількість станцій до 100). Це наприклад метро Мінська, Санкт-Петербургу тощо. Більш того, є можливість проекспериментувати з правилами мутацій та схрещування особин для того, щоб досягти більш оптимального результату за коротший час.

При проведенні експерименту та дослідження генетичного алгоритму ми маємо можливість задати параметри, які будуть впливати на ймовірність схрещування та застосування окремих мутацій. Кожен з цих параметрів може в той чи іншій мірі впливати на результат. До таких параметрів належать:

- кількість особин в популяції – g;
- ймовірність схрещування геномів – p’;
- ймовірність проведення мутації на геномі – p’’ ;
- ймовірність застосування мутації перестановки декількох пар – p’’’;
- ймовірність застосування мутації з випадковим перетасуванням – p’’’’;
- ймовірність застосування мутації дзеркального відображення – p’’’’’;

При дослідженні буде використано два графи, побудовані на основі Українських метрополітенів:

- граф, побудований на основі Харківського метрополітену (30 вершин);
- граф, побудований на основі Київського метрополітену (52 вершини);

Схеми цих метрополітенів представлені на рисунках 5.1 та 5.2 відповідно.

Рисунок 5.1 – Схема Харківського метрополітену

Рисунок 5.2 – Схема Київського метрополітену

Також окрім цих графів буде досліджено вигаданий граф, який імітує транспортну схему з 10 переходами та 50 вершинами. Даний граф дозволить протестувати алгоритм на більш складніших даних, чим українські метрополітени. Такий граф, з зазначенням номерів вершин та ваг ребер представлено на рисунку 5.3.

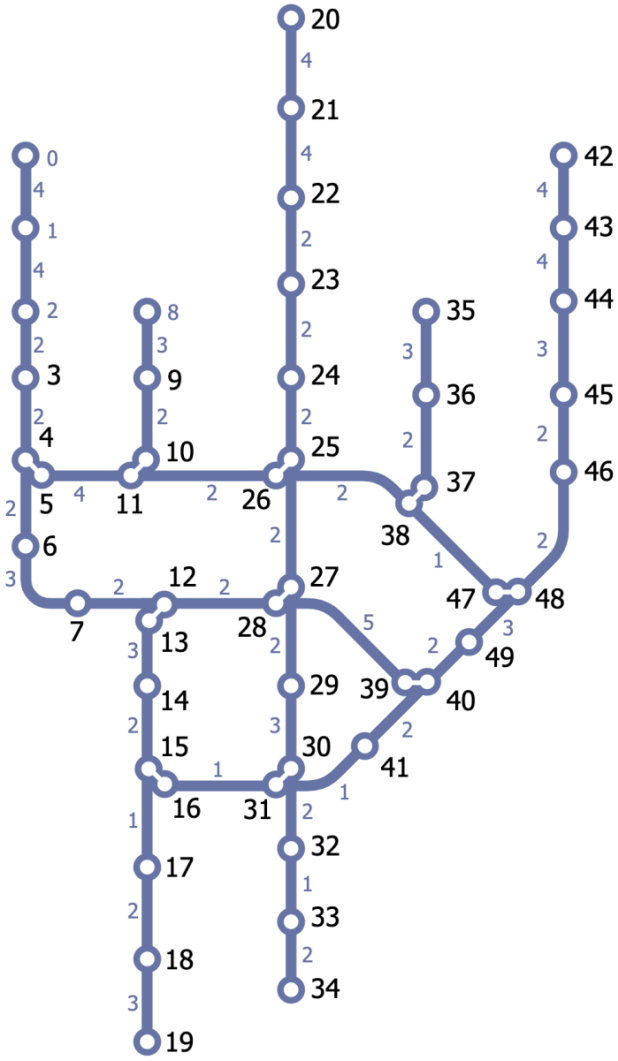


Рисунок 5.3 – Вигаданий граф з великою кількістю пересадок

Для дослідження розробленого генетичного алгоритму та знаходження найбільш ефективного варіанту його використання необхідно провести серію експериментів.

Під час проведення серії експериментів необхідно дослідити вплив початкових параметрів алгоритму та їх зміни у процесі роботи програми. Таким чином буде виявлено найоптимальніші параметри та стратегію пошуку потрібного маршруту.

Для коректного порівняння ефективності потрібно обрати час, на протязі якого буде працювати алгоритм. Теоретично він може працювати нескінченно довго, тому потрібно обрати значення, при якому результат почне змінюватися дуже повільно. Для цього було проведено кілька пробних запусків алгоритму при різних параметрах на графі Харківського метрополітену. На графіку, наведеному на рисунку 5.4, точками зображено значення функції пристосованості та час, за який вони були знайдені. З графіку видно, що після 60 секунд алгоритм втрачає темп покращення результату та демонструє стагнацію.

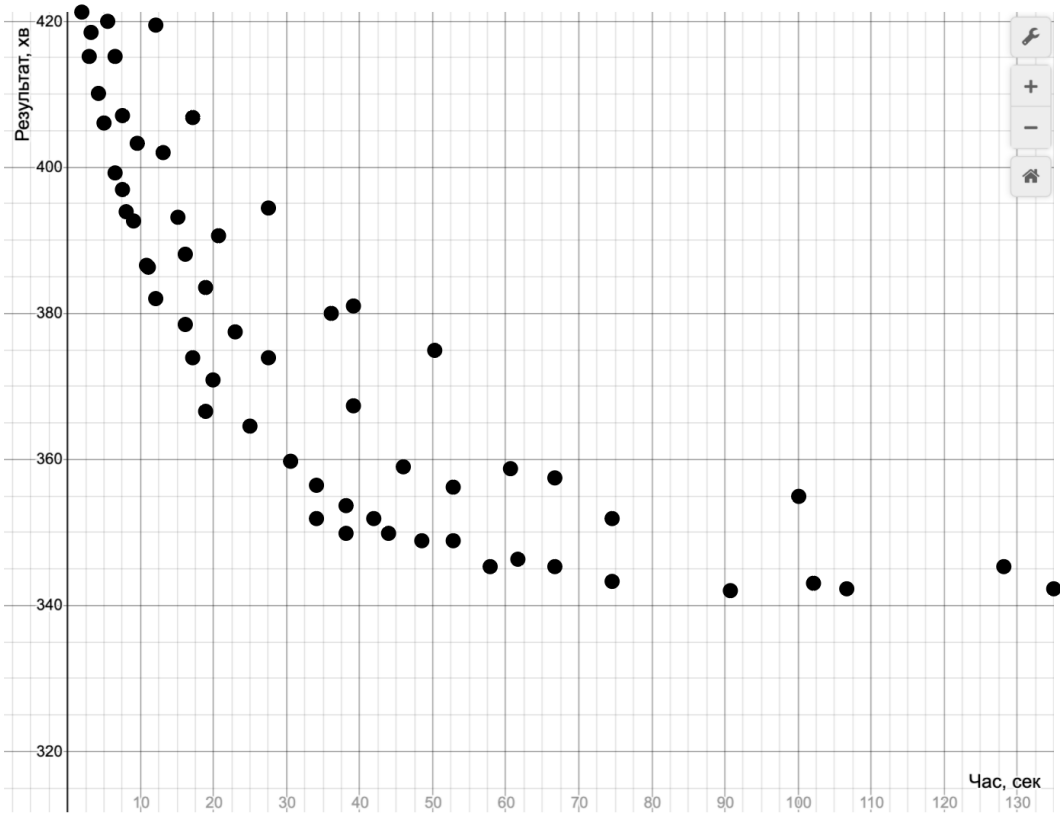


Рисунок 5.4 – Динаміка знаходження оптимальних маршрутів для Харківського метрополітену

Таким чином, кожен запуск алгоритму буде тривати приблизно одну хвилину. Після закінченні хвилини алгоритм буде зупинено, та обрано за фінальний результат краще значення за всю історію популяції. Саме це значення буде порівнюватися з іншими. Якщо будь-який параметр має бути змінено в процесі виконання програми, то це буде зроблено в останні 30 секунд роботи. Кожен цикл алгоритму буде тривати 50 мілісекунд. Саме за такий період браузер встигне виконати всі функції та перейти до іншої ітерації. Слід зауважити, що на більш продуктивних машинах цей показник можна зменшити. На кожній станції буде проведено по 5 хвилин мінімум, включаючи інтервали поїздів. Саме за цей час можна встигнути подивитися всю станцію її архітектуру, сфотографуватись на ній, тобто відмітити її як відвідану.

Набори параметрів, які буде досліджено при тестуванні алгоритму вказано в таблиці 5.1.

Таблиця 5.1 – Набір параметрів та їх зміни для тестування

№	Ймовірності або їх зміна					К-сть особин в популяції, шт
	Мутації взагалі	Мутації «Перестановка пар»	Мутації випадкового перетасування	Мутації дзеркального відображення	Схрещування	
1	95%	10%	10%	5%	1%	20
2	5%	5%	5%	5%	98%	20
3	10%	10%	10%	5%	80%	50
4	10%	5% → 30%	30% → 5%	5%	10%→80%	20
5	10%	30% → 5%	5% → 30%	5%	80%→10%	20

При встановленні ймовірностей для мутацій було прийнято таку відповідність значень:

– 95% – дуже висока ймовірність. При значеннях від 95% до 100% кількість мутацій стає настільки високою, що комп’ютер не справляється з такими обчисленнями і браузер перестає відповідати;

– 10% – звичайна ймовірність. Стандартне значення ймовірності мутації для генетичного алгоритму, рекомендоване в роботі Shortest path problems with resource constraints [1].

– 30% – висока ймовірність, втричі більша за стандартну;

– 5% – невелика ймовірність, вдвічі менша за стандартну.

При встановленні ймовірностей для схрещування було прийнято таку відповідність значень:

– 98% – дуже висока ймовірність. При значеннях від 98% до 100% кількість мутацій стає настільки високою, що комп’ютер не справляється з такими обчисленнями і браузер перестає відповідати;

– 80% – звичайна ймовірність. Стандартне значення ймовірності схрещування для генетичного алгоритму, рекомендоване в роботі Shortest path problems with resource constraints [1];

– 90% – висока ймовірність, на 10% більша за звичайну;

– 10% – маленька ймовірність, є стандартною ймовірністю мутації;

– 1% – дуже маленька ймовірність при якій схрещування відбувається в 80 разів рідше, чим при звичайній ймовірності.

Величина популяції була встановлена за стандартною рекомендацією [1] як середнє арифметичне кількості вершин всіх графів поділене на два.

Перший набір даних має на меті перевірити залежність результату від кількості мутацій, тому ймовірність мутації було збільшено до 95%. Другий набір даних більш стандартний для такого роду задач та має показати непогані результати. В цьому наборі велика ймовірність схрещування та невелика ймовірність мутації. Третій набір має на меті перевірити роботу з популяцією в 50 особин, тобто в 2,5 рази більшою, чим в інших наборах. Четвертий набір

базується на зміні ймовірності мутацій і схрещування в процесі роботи програми та є прогнозовано найоптимальнішим, як було досліджено в підрозділі 3.4. У п’ятому наборі стратегія зміни параметрів повністю протилежна попередньому пункту.

Кожен тест буде запущено на кожному графі по 3 рази. Це пояснюється тим, що результат може збігатися в локальний оптимум з якого буде дуже складно вибратись за хвилину. Тому серед 3 спроб на кожному наборі даних буде обрано один кращий результат.

При проведенні експерименту програма дозволяє фіксуватися наступні показники в реальному часі:

– історія зміни кращого результату;
– кількість мутацій та схрещувань;
– номер покоління;
– кращий результат в виді геному та значення функції пристосованості;
– вся популяція в виді набору геномів та її кращий геном.

Серед цих показників нас цікавить кращий результат в виді значення функції пристосованості. За ним ми будемо порівнювати ефективність того чи іншого набору даних. Такі показники як кількість мутацій, схрещувань та номер покоління дадуть уявлення про перебіг роботи алгоритму та дадуть можливість підтвердити відповідність введених параметрів до очікуваного результату. Кількість поколінь у кожному експерименті повинна бути приблизно однаковою. Якщо це значення буде сильно відрізнятися, то це буде значити, що експеримент був невдалим. За відповідністю значення функції пристосованості до кількості мутацій та схрещувань буде зроблено висновки щодо впливу їх кількості на ефективність алгоритму.

Усі дослідження проводяться на одному фізичному комп’ютері, який має наступні характеристики: операційна система MacOS Mojave 10.14.2, процесор 2 GHz Intel Core i5, оперативна пам’ять 8 ГБ 1867 MHz LPDDR3, відеокарта Intel Iris Graphics 540 1536 МБ, браузер Chrome 78.

5.2 Дослідження розробленого генетичного алгоритму

Результати дослідження серії експериментів для Харківського метрополітену наведено на рисунку 5.5 та в таблиці 5.2.

Результати дослідження серії експериментів для Київського метрополітену наведено на рисунку 5.6 та в таблиці 5.3.

Результати дослідження серії експериментів для вигаданого графу, що імітує транспортну схему наведено на рисунку 5.7 та в таблиці 5.4.

На наведених рисунках представлено по 5 скриншотів результатів запуску алгоритму відповідно до кожного з набору параметрів. Кращий результат наведено в хвилинах.

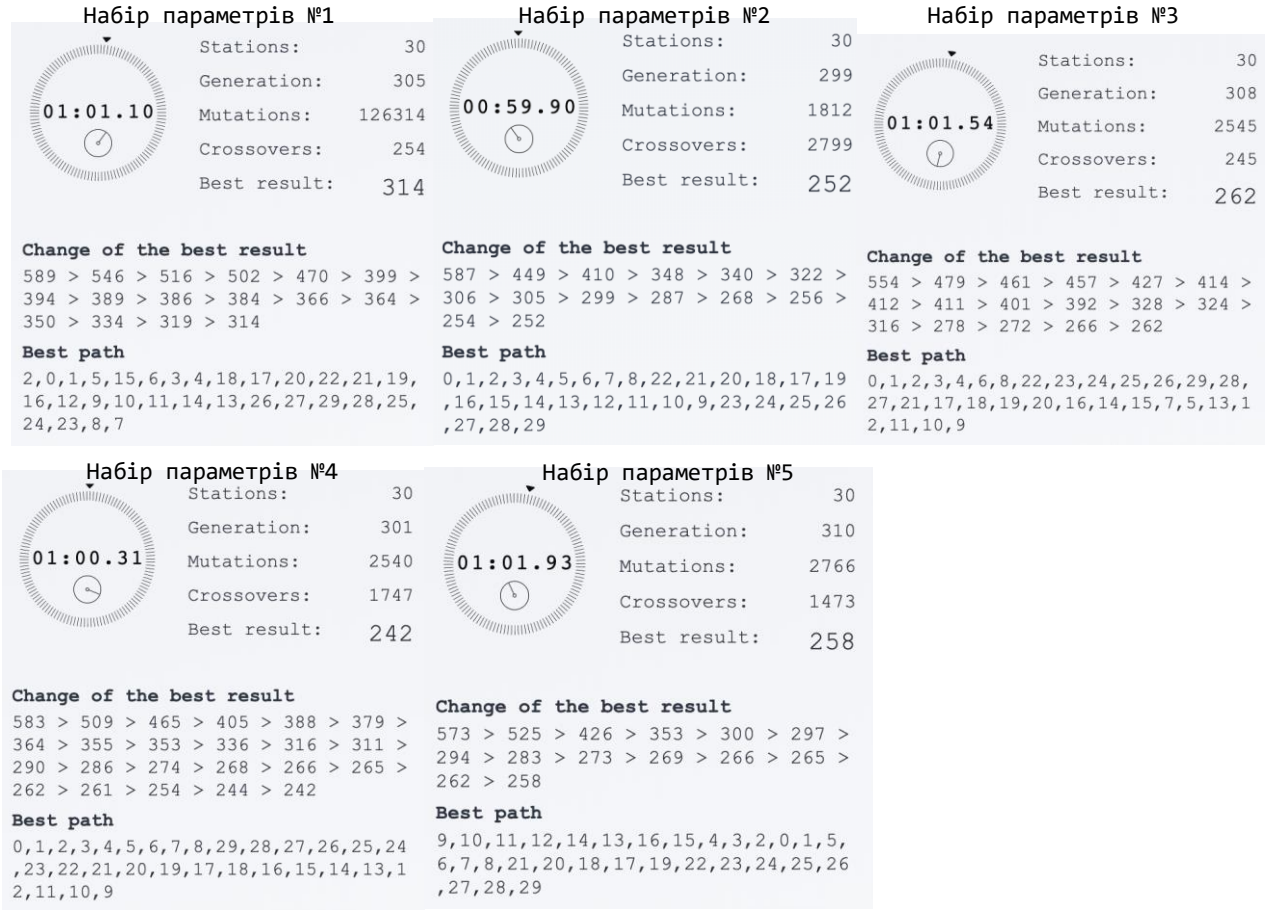


Рисунок 5.5 – Скриншоти результатів серії експериментів для Харківського метрополітену з наборами параметрів №1-5 відповідно

Таблиця 5.2 – Результати серії експериментів для Харківського метрополітену (30 станцій, 3 переходи)

Показник	Експеримент номер				
	1	2	3	4	5
Кращий результат, хв	314	252	262	242	258
К-сть мутацій	126314	1812	2545	2540	2766
К-сть схрещувань	254	2799	245	1747	1473
К-сть поколінь	305	299	308	301	310

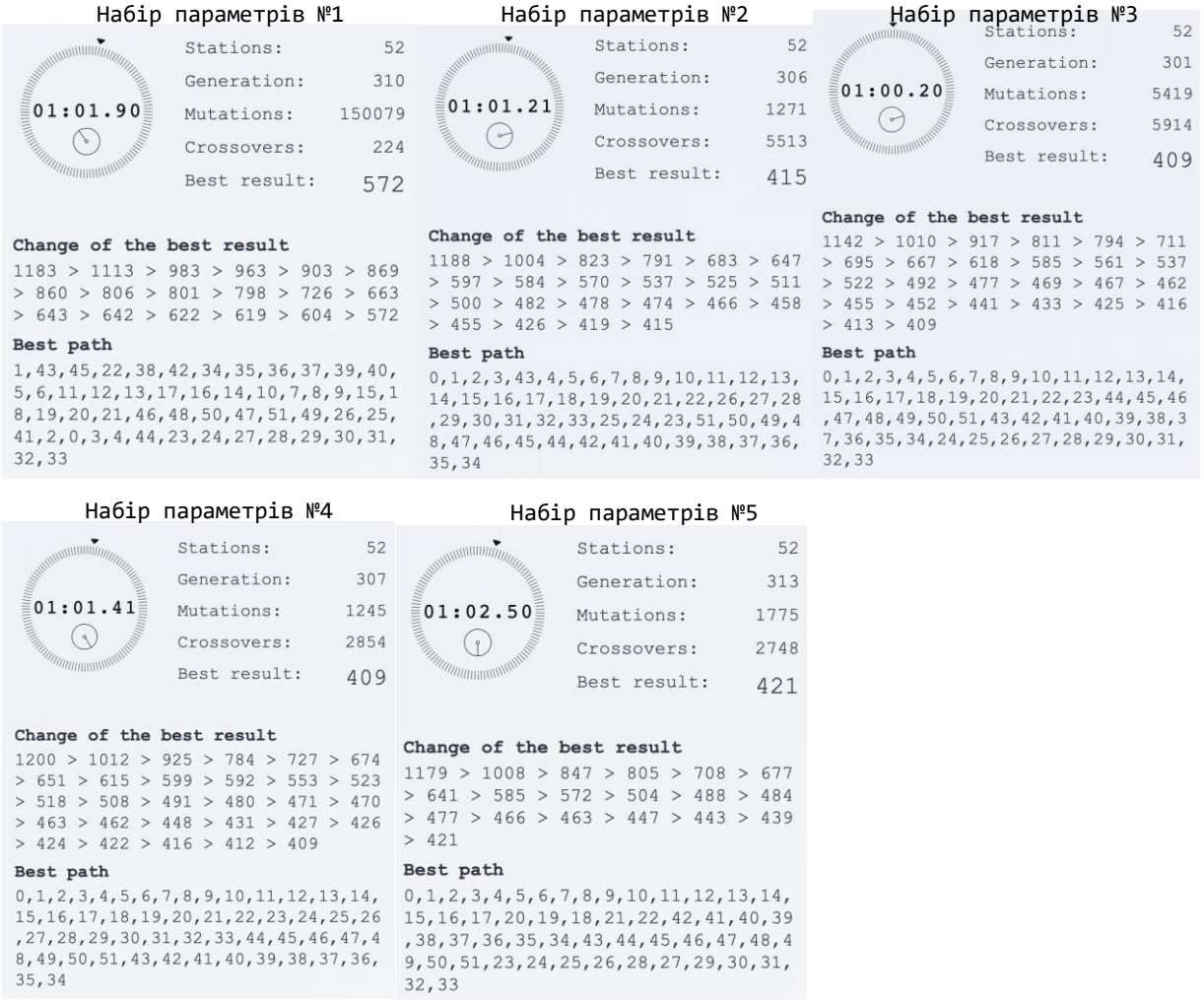


Рисунок 5.6 – Результати експерименту для Київського метрополітену з наборами даних №1-5 відповідно

Таблиця 5.3 – Результати серії експериментів для Київського метрополітену (52 станції, 3 переходи)

Показник	Експеримент номер				
	1	2	3	4	5
Кращий результат, хв	572	415	409	409	421
К-сть мутацій	15079	1271	5419	1245	1775
К-сть схрещувань	224	5513	5914	2854	2748
К-сть поколінь	310	306	301	307	313

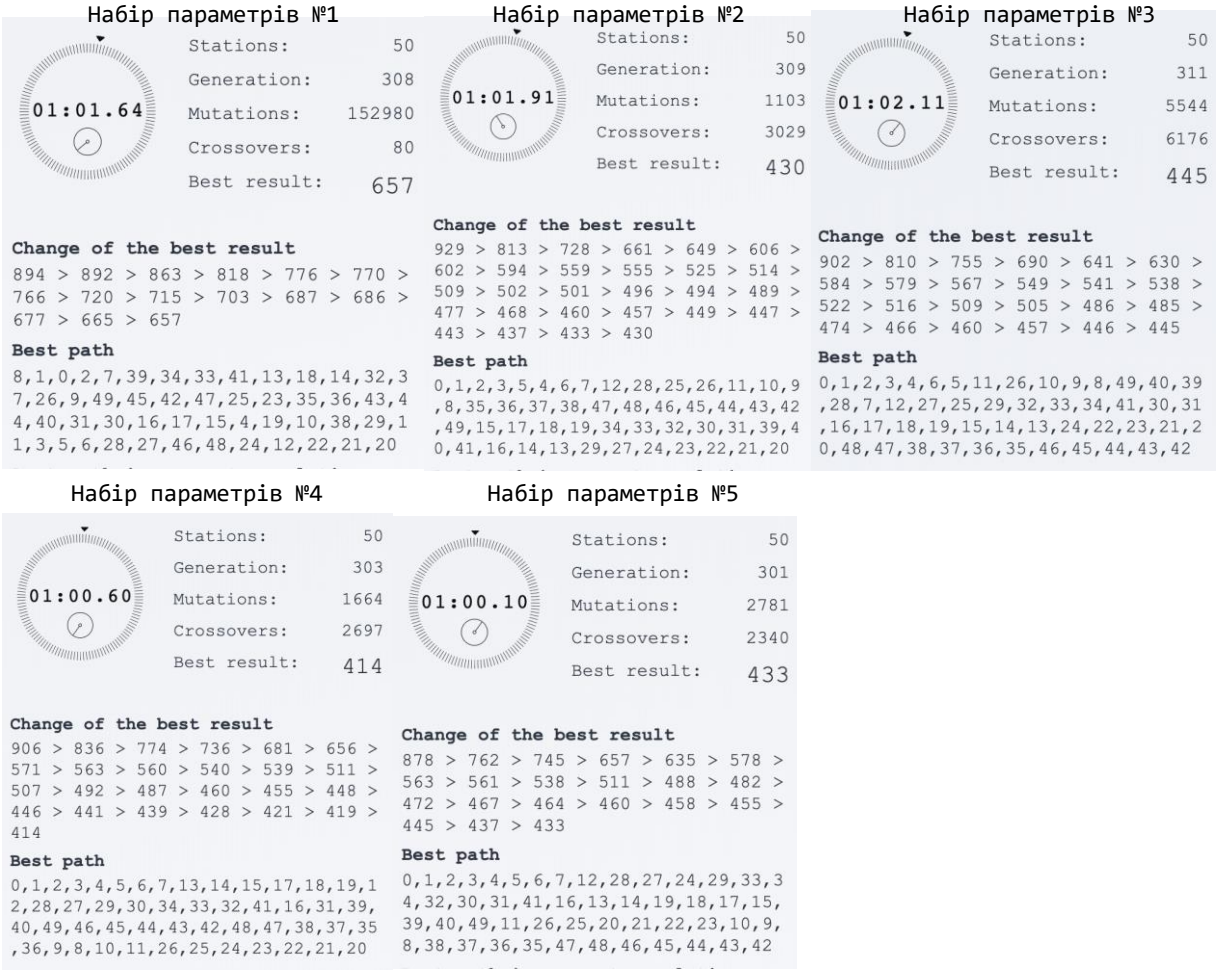


Рисунок 5.7 – Результати експерименту для вигаданого метрополітену з наборами даних №1-5 відповідно

Таблиця 5.4 – Результати серії експериментів для вигаданого метрополітену (50 станцій, 10 переходів)

Показник	Експеримент номер				
	1	2	3	4	5
Кращий результат, хв	657	430	445	414	433
К-сть мутацій	152980	1103	5544	1664	2781
К-сть схрещувань	80	3029	6176	2697	2340
К-сть поколінь	308	309	311	303	301

Аналізуючи на порівнюючи отримані дані, можна зробити висновок, що найоптимальнішою виявився набір параметрів номер 4, як і було прогнозовано. Досить непогані результати показали 2 та 3 набори. В деяких випадках 3 набір даних показував найкращий результат серед всіх наборів. Стратегія, протилежна оптимальній, показала в середньому на 4% гірший результат, що дорівнює 16 хвилинам різниці з найкращим результатом.

Набір даних №1, де ймовірність мутації була найвищою, показав найгірший результат. Це пояснюється тим, що особини дуже швидко мутували, не встигаючи обмінятися між собою генами. Таким чином результат дуже швидко потрапляв в локальний оптимум з якого було дуже складно вибратися без схрещування та з такою великою кількістю мутацій.

Другою частиною експерименту є порівняння результату звичайної роботи алгоритму з початковими параметрами та роботи з ручним коректуванням параметрів. Тестування буде проводитися на протязі трьох хвилин. В результаті ми очікуємо невеликого, але зменшення фінального результату при ручному коректуванні параметрів. Тестування буде проводитися з третім графом (50 станцій, 10 переходів) бо він є найскладнішим для пошуку оптимального шляху.

Серед наборів параметрів із першого етапу експерименту, які включають в себе лише статичні значення, було обрано набір №3. Цей набір показав найкращі значення серед наборів, чиї параметри не змінювалися в ході роботи алгоритму. Початкові параметри для другого етапу тестування вказано в таблиці 5.5.

Таблиця 5.5 – Початкові параметри для другого етапу тестування

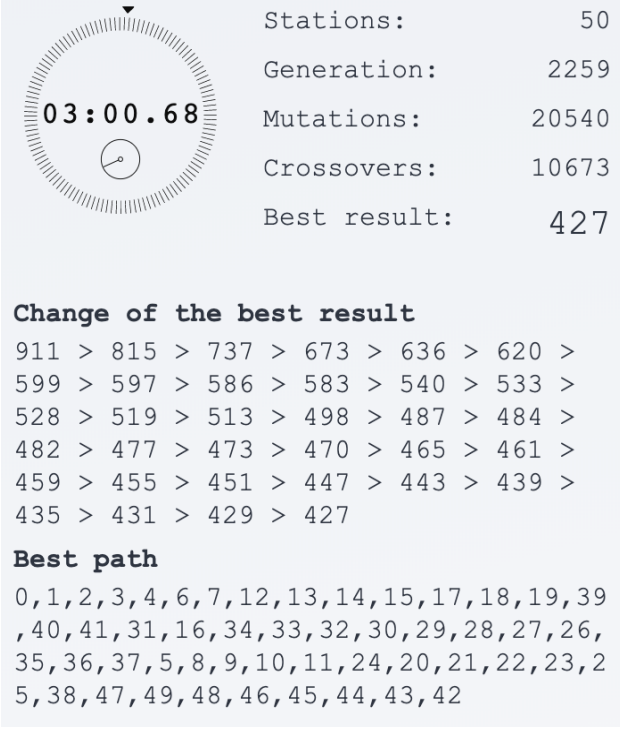
Ймовірності					К-сть особин в популяції, шт
Мутації взагалі	Мутації «Перестановка пар»	Мутації випадкового перетасування	Мутації дзеркального відображення	Схрещування	
10%	10%	10%	5%	80%	50

В першому випадку ці параметри будуть незмінними на протязі всього експерименту. В другому випадку. При роботі алгоритму після швидкого зменшення результату та його стагнації буде збільшено ймовірність мутації дзеркального відображення та ймовірність схрещування з 5% до 30% та з 80% до 98% відповідно. Результати другого етапу експерименту зображено на рисунку 5.8.

За результатами другого етапу експерименту можна зробити висновок, що зміна параметрів хоч і не суттєво, але впливає на фінальний результат. Із-за зміни параметрів збільшилася кількість мутацій та схрещувань, але був знайдений маршрут на 17 хвилин оптимальніший.

Таким чином, було протестовано розроблений генетичний алгоритм на трьох графах, побудованих на основі транспортних маршрутів. Експериментальним шляхом було визначено оптимальні початкові параметри для розробленого генетичного алгоритму та стратегію їх зміни в процесі роботи алгоритму. Було виявлено вплив великої кількості мутацій та схрещувань на отриманий результат.

Без зміни параметрів



Зі зміною параметрів

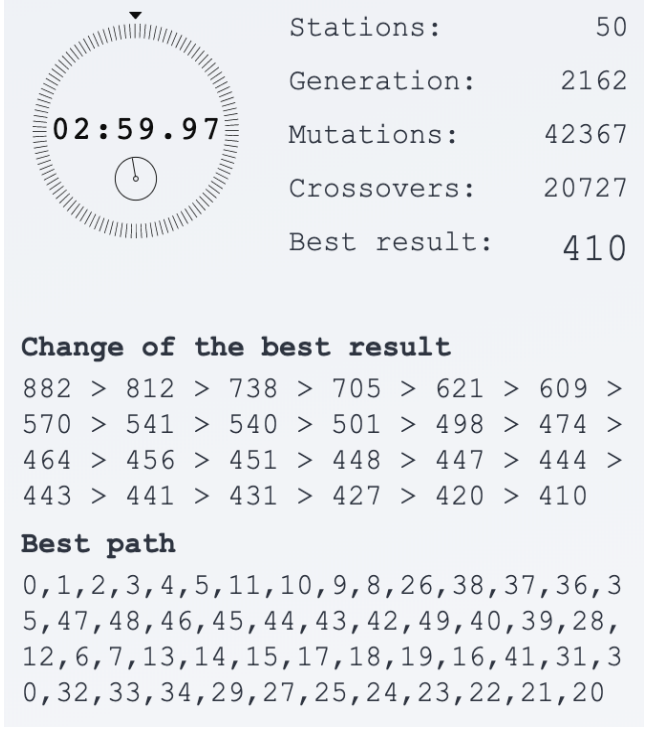


Рисунок 5.8 – Результати другого етапу експерименту

Отже, за результатами проведеного експерименту можна сформулювати наступні рекомендації щодо використання даного алгоритму:

– не задавати високу ймовірність мутації та не змінювати її за межі 10 відсотків. При високій ймовірності проведення мутації, їх кількість стає дуже великою, що призводить до передчасного входу в локальний оптимум;

– мутація з випадковим перетасуванням геному краще працює на початку роботи алгоритму;

– коли значення перестає швидко змінюватися, можна спробувати зменшити ймовірність мутації з випадковою перетасуванням та збільшити ймовірність мутації з перестановкою пар;

– якщо значення довго перебуває в локальному оптимумі, можна спробувати збільшити вірогідність мутації з дзеркальним відображенням або ймовірність схрещування.

ВИСНОВКИ

Під час атестаційної магістерської роботи було досліджено генетичні алгоритми на прикладі створення програмної системи для проведення метромарафонів «Transit challenge».

Було сформовано математичний опис базової моделі метромарафону та сформульовано проблему станцій, яка дала змогу знайти оптимальний маршрут в метро, який би охопив всі стації метрополітену. Така проблема є дуже схожою на популярно проблему комівояжера, яка є NP-повною. Тому її рішення за допомогою генетичних алгоритмів виявилось актуальним.

Також, було сформовано основні принципи використання генетичних алгоритмів в рамках даної проблеми. Було розроблено та застосовано генетичні алгоритми для вирішення цієї проблеми: обрано спосіб задання геному, види мутацій та принципи схрещування. При виборі геному для алгоритму було вирішено взяти перестановку по кількості станцій метрополітену, тобто перейти до повного графу. При цьому знадобилось знайти оптимальні маршрути між кожною парою маршрутів. В решті було сформульовано декілька правил, які б дозволили без проблем схрещувати геноми між собою та вибиратись з локальних оптимумів. Також було обрано спосіб, за яким данні графа будуть зберігатися в системі.

Реалізація системи відбувалася за допомогою мови програмування JavaScript на базі фреймворку Svelte, який надав змогу запрограмувати динамічну зміну параметрів алгоритму в процесі роботи програми. Після реалізації системи було проведено два етапи тестування. В результаті чого було сформовано рекомендації щодо застосування програми, вибору початкових параметрів алгоритму та їх зміни.

В результаті дане дослідження дозволяє знайти досить оптимальний маршрут в метрополітені за більш-менш реальний час, не застосовуючи повний перебір варіантів.

В ході атестаційної роботи магістра було:

– проведено аналіз та моделювання предметної;

– розроблено математичну модель метромарафону;

– розроблено схему бази даних та структуру для збереження інформації та роботи з графами;

– на основі математичної моделі було розроблено генетичний алгоритм для пошуку оптимального маршруту для метромарафону;

– реалізовано веб-додаток для пошуку оптимальних шляхів в метрополітені;

– проведено дослідження генетичних алгоритмів для побудови маршрутів метромарафонів на Харківському та Київському метрополітенах. Також алгоритм було протестовано на вигаданому графу, який імітує більш складнішу транспортну схему, чим українські метрополітени;

– сформовано рекомендації щодо початкових параметрів розробленого алгоритму та їх зміні в процесі роботи програми

– за результатами проведеного дослідження сформувати рекомендації щодо початкових параметрів розробленого алгоритму та їх зміні в процесі роботи програми.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток А).

Було зроблено доповідь на Міжнародному молодіжному форумі. Тези доповіді наведено в додатку Б. Також була написана та подана до опублікування до науково-технічного журналу «Біоніка інтелекту» стаття «Дослідження генетичних алгоритмів для пошуку оптимальних шляхів в системі проведення метромарафонів». (див. додаток В). Лістинг коду наведено в додатку Г.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Larrañaga P., Kuijpers C. M. H., Murga R. H. Genetic algorithms for the travelling salesman problem: a review of representations and operators. Artificial Intelligence Review: SCOPUS, 1999 – vol. 13, no. 2, pp. 129–170.

2. Baaj, M. H., and Mahmassani, H. S. An AI-based approach for transit route system planning and design. Journal of Advanced Transportation, 1990 – p. 287.

3. Nagata Y. and Soler D. A new genetic algorithm for the asymmetric traveling salesman problem. Expert Systems with Applications: SCOPUS, 2012 – vol. 39, no. 10, pp. 847–853.

4. D. Spensieri, R. Bohlin, and J. S. Carlson. Coordination of robot paths for cycle time minimization. In Proceedings of the 9th annual IEEE International Conference on Automation Science and Engineering (CASE ’13), 2013 – p. 527.

5. Soheil Ghafurian, Nikbakhsh Javadian: An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems, Applied Soft Computing 11. 2011 – p. 1262.

6. H. Nazif and L.S. Lee, Optimized CrosSover Genetic Algorithm for Vehicle Routing Problem with Time Windows, American Journal of Applied Sciences 7 (1): 2010 – p. 101.

7. Dijkstra, E.W. A note on two problems in connexion with graphs. Numerische Mathematik. 1, 1959 – p. 471.

8. Fan L. and Mumford C. L. A Metaheuristic Approach to the Urban Transit Routing Problem. Journal of Heuristic. 16, 2010 – p. 372.

9. Ngamchai, S. and Lovell, D. Optimal time transfer in bus transit route network design using a genetic algorithm, Journal of Transportation Engineering. 129 (5), – 2003 – pp. 510–521.

10. R. Braune, S. Wagner and M. Affenzeller, Applying Genetic Algorithms to the Optimization of Production Planning in a real world Manufacturing Environment, Institute of Systems Theory and Simulation Johannes Kepler University: 2005 – p. 43.

11. E. L. Lawler, J. K. Lenstra, A. H. G. RinnooyKan, and D. B. Shmoys, The Traveling Salesman Problem, John Wiley & Sons, Chichester: 1985 – p. 820.

12. Вигерс, К. Розробка вимог до програмного забезпечення [Текст] / К. Вигерс. — М.: Російська редакція Microsoft, 2004. – 575 с.

13. Офіційний сайт КП «Харківський метрополітен» [Електронний ресурс] metro.kharkov.ua/uk : КП «Харківський метрополітен». — Режим доступу: www/URL: http://www.metro.kharkov.ua/uk (дата звернення: 18.10.2019).

14. Схема метро Харькова — Яндекс.Метро [Електронний ресурс] Режим доступу: : www/URL: https://metro.yandex.ru/kharkov/ (дата звернення: 13.10.2019).

15. Карта метро Харькова - интерактивная схема с расчетом времени [Електронний ресурс] URL: http://metrobook.ru/ua_kharkiv/ (дата звернення: 13.10.2019).

16. Кренке, Г. Теорія й практика побудови баз даних [Текст] / Г. Кренке. — П.: Питер, 2001. – 858с.

17. Дейт, К. Введение в системы баз данных. [Текст] / К. Дейт. – М.: Наука, 1980. – 464 с.

18. Система запитань та відповідей Stack Overflow [Електронний ресурс] stackoverflow.com : Stack Overflow FAQ. – Режим доступу: www/URL: http://stackoverflow.com (дата звернення: 21.11.2019).

19. Руткас А. Г. Введение в теорию графов: навч. посіб. — Харків: ХГУ, 1993. — 64 с.

20. Самойленко, М. І. Інформаційні технології в розв’язанні транспортних задач. [Текст] / М. І. Самойленко. – Харків.: ХНАМГ, 2011. — 256 с.

21. Murat Albayrak and Novruz Allahverdi, Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. Expert Systems with Applications, vol. 38, no. 3, 2011 – pp. 1313–1320.

22. Polzin T. and Vahdati S. Daneshmand. Comparison of Algorithms for Solving Traveling Salesman Problem. Discrete Applied Mathematics, 2001 – p. 20.