МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Адаптація генетичного алгоритму для рішення задачі комівояжера в порівнянні з алгоритмом "Мурахи"»

Завідувач кафедри	Довбиш А.С.
Керівник роботи	Шаповалов С.П.
Студент гр. ІН.м - 92	П’ятикоп А.Г.

Суми 2020

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *"Інформатика"*

Затверджую:

зав.кафедрою ______________

____

"____" ______________ 20__р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

П'ятикопу Андрію Геннадійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Адаптація генетичного алгоритму для рішення задачі комівояжера в порівнянні з алгоритмом "Мурахи"*

затверджую наказом по інституту від "____" ______________ 20 ___ р. № ____________

2. Термін здачі студентом закінченого проекту (роботи) ______________

3. Вхідні данні до проекту (роботи) ______________

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1)Аналіз задачі, огляд відомих методів вирішення 2)Обрання і програмна реалізація алгоритмів 3)Дослідження якості роботи алгоритмів в залежності від різних параметрів*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) ______________

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання ____

Керівник ____________________
(підпис)

Завдання прийняв до виконання ____________________
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1	Аналіз задачі		
2	Огляд відомих методів вирішення		
3	Обрання і програмна реалізація алгоритмів		
4	Дослідження якості роботи алгоритмів в залежності від різних параметрів		
5	Оформлення пояснювальної записки до дипломної роботи		

Студент – дипломник ____________________
(підпис)

Керівник проекту ____________________
(підпис)

РЕФЕРАТ

Записка: 66 стор., 51 рис., 2 додатки, 13 джерел інформації.

Об'єкт дослідження – алгоритми вирішення задачі комівояжера.

Мета роботи — провести дослідження якості роботи генетичного алгоритму та алгоритму мурахи в залежності від зміни параметрів.

Методи дослідження – алгоритм мурахи і генетичний алгоритм у вирішенні задачі комівояжера, програмна реалізація алгоритмів.

Результати — проведено порівняльний аналіз алгоритмів, виконана програмна реалізація алгоритмів мовою програмування Java, досліджено якість роботи генетичного алгоритму та алгоритму мурахи в залежності від зміни параметрів.

ЗАДАЧА КОМІВОЯЖЕРА, АЛГОРИТМ МУРАХИ, ГЕНЕТИЧНИЙ АЛГОРИТМ, ПРОГРАМНА РЕАЛІЗАЦІЯ, ДОСЛІДЖЕННЯ ЯКОСТІ РОБОТИ АЛГОРИТМУ, ДОСЛІДЖЕННЯ ЯКОСТІ РОБОТИ АЛГОРИТМУ МУРАХИ, РІВЕНЬ МУТАЦІЇ, РІВЕНЬ ВИПАРОВУВАННЯ ФЕРОМОНУ

ЗМІСТ

ВСТУП ... 6

1 ІНФОРМАЦІЙНИЙ ОГЛЯД ... 8

1.1 Описання відомих методів розв’язку задачі комівояжера ... 8

1.2 Постановка задачі ... 14

1.3 Обгрунтування вибору мови програмування ... 15

2 ВИБІР АЛГОРИТМІВ РОЗВ’ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ ... 16

2.1 Загальні технології розв’язку для алгоритму “мурахи” та генетичного алгоритму ... 16

2.2Технології розв’язку генетичного алгоритму ... 17

2.3 Технології розв’язку алгоритму “мурахи” ... 19

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ... 25

3.1 Описання основних java – класів та констант генетичного алгоритму для задачі комівояжера ... 25

3.2 Ключові функції для реалізації генетичного алгоритму ... 27

3.3 Описання основних java – класів та констант алгоритму “мурахи” для задачі комівояжера ... 31

3.4 Ключові функції для реалізації алгоритму “мурахи” ... 33

3.5 Дослідження роботи генетичного алгоритму в залежності від кількості міст маршруту ... 37

3.6. Дослідження роботи генетичного алгоритму в залежності від рівня мутації та кількості поколінь ... 45

3.7 Дослідження роботи алгоритму “мурахи” в залежності від кількості агентів ... 52

3.8 Дослідження роботи алгоритму “мурахи” в залежності від рівня випаровування феромону ... 58

ВИСНОВКИ ... 61

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 63

ДОДАТОК І ... 65

ДОДАТОК ІІ ... 66

ВСТУП

Задачею про комівояжера називається завдання з математичного програмування за пошуком найкоротшого шляху руху мандрівного торговця (комівояжера), ціль якого зводиться до того, щоб відвідати всі об'єкти, записані в задачі, за найменший термін і з мінімальними витратами. В теорії графів - це знаходження маршруту, який поєднує два або понад вузлів, при використанні критерію оптимальності.

Завдання бродячого торговця полягає в пошуку найбільш оптимального шляху, що пролягає через помічені міста хоча б по одному разу. В умовах задачі зазначається показник вигідності маршруту (найдешевший, найбільш короткий, найменш часозатратний і т. п.) і відповідні матриці вартості, відстаней і т. п. Як правило вказується, що шлях має включати кожне місто маршруту тільки один раз, за такої умови вибір виконується поміж гамільтонових циклів.

Завдання мандрівного торговця - traveling salesman problem (TSP) – це NP-складне завдання дискретної оптимізації. Для нього поки що не винайдено і мабуть немає швидких поліноміальних алгоритмів. У сукупному вигляді на графах задача визначається в такий спосіб: слід знайти найкоротший шлях, що проходить через визначені вузли графа хоча б по одному разу з подальшим поверненням до вузла на початку маршруту [9].

У теперішньому часі задача комівояжера знаходить достатньо велику кількість практичних реалізацій в таких галузях як:

- оптимізація в мережах;
- оптимізація шляхів;
- керування роботами;
- дослідження ДНК.

Деколи в різних завданнях можуть бути залученими як фізичні об'єкти, так і різноманітні сутності або процеси.

Наприклад, задача пошуку найкоротшого маршруту майже без змін може застосовуватися для створення шляху обльоту території літальним апаратом [4].

Тобто, пошук точних і наближених методів розв’язку задачі бродячого торговця є актуальним як з теоретичної, тік і з практичної точки зору. У поточному випадку було використано два алгоритми у вирішенні завдання пошуку найкоротшого маршруту – генетичний алгоритм та алгоритм “мурахи”.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Описання відомих методів розв’язку задачі комівояжера

Метод гілок і меж

Завдання дискретної оптимізації мають кінцеву множину допустимих рішень, які теоретично можна перебрати і вибрати найкраще (що дає мінімум або максимум цільової функції). Практично ж часто це буває нездійсненно навіть для задач невеликої розмірності.

У методах неявного перебору робиться спроба так організувати перебір, використовуючи властивості розглянутої задачі, щоб відкинути частину допустимих рішень. Найбільшого поширення серед схем неявного перебору отримав **метод гілок і меж**, в основі якого лежить ідея послідовного розбиття множини допустимих рішень[11]. На кожному кроці методу елементи розбиття (підмножини) піддаються аналізу – містить дана підмножина оптимальне рішення чи ні. Якщо розглядається задача на мінімум, то перевірка здійснюється шляхом порівняння нижньої оцінки значення цільової функції на даній підмножині з верхньою оцінкою функціоналу. В якості оцінки зверху використовується значення цільової функції на деякому допустимому рішенні. Допустиме рішення, що дає найменшу верхню оцінку, називають рекордом. Якщо оцінка знизу цільової функції на даній підмножині не менша оцінки зверху, то підмножина, що розглядається не містить рішення краще рекорду і може бути відкинутою. Якщо значення цільової функції на черговому рішенні менше рекордного, то відбувається зміна рекорду. Будемо говорити, що підмножина рішень переглянута, якщо встановлено, що вона не містить рішення краще рекорду.

Якщо переглянуті всі елементи розбиття, алгоритм завершує роботу, а поточний рекорд є оптимальним рішенням. В іншому випадку серед непереглянутих елементів розбиття вибирається множина, що є в певному сенсі перспективною. Вона піддається розбиттю (розгалуженню). Нові підмножини аналізуються за описаною вище схемою. Процес триває до тих пір, доки не будуть переглянуті всі елементи розбиття [10].

Алгоритм Дейкстри

Алгоритм Дейкстри – алгоритм на графах, винайдений Е. Дейкстрою. Знаходить найкоротшу відстань від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер з негативною вагою. Алгоритм широко застосовується в програмуванні і технологіях. Відомий також під назвою найкоротший шлях – перший (Shortest Path First) [7].

Формулювання завдання

Дано простий зважений граф G (V, E) без петель і дуг з негативною вагою. Потрібно знайти найкоротші шляхи від деякої вершини **а** графа **G** до всіх інших вершин цього графа.

Ініціалізація

Кожній вершині з **V** зіставимо мітку – мінімальна відома відстань від цієї вершини до вершини **а**. Алгоритм працює покроково – на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані. Мітка самої вершини **а** покладається рівною 0, мітки інших вершин – рівним нескінченності. Це відображає те, що відстані від вершини **а** до інших вершин поки невідомі. Всі вершини графа позначаються як ще не відвідані[11].

Крок алгоритму

Якщо всі вершини відвідані, алгоритм завершується. В іншому випадку з ще не відвіданих вершин вибирається вершина **u**, що має мінімальну позначку. Ми розглядаємо різні маршрути, в яких **u** є передостаннім пунктом. Вершини, з’єднані з вершиною **u** ребрами, назвемо сусідами цієї вершини. Для кожного сусіда розглянемо нову довжину шляху, що дорівнює сумі поточної мітки **u** і довжини ребра, що з’єднує **u** з цим сусідом. Якщо отримана довжина менше мітки сусіда, замінимо мітку цієї довжиною. Розглянувши всіх сусідів, позначимо вершину **u** як відвідану і повторимо крок [8].

Позначення алгоритму:

- V — множина вершин графа
- E — множина ребер графа
- w[ij] — вага (довжина) ребра ij
- a — вершина, від якої шукають відстані до інших вершин
- U — множина відвіданих вершин
- d[u] — по закінченні роботи алгоритму дорівнює довжині найкоротшого шляху з вершини a до вершини u
- p[u] — по закінченні роботи алгоритму містить найкоротший шлях з вершини a до вершини u

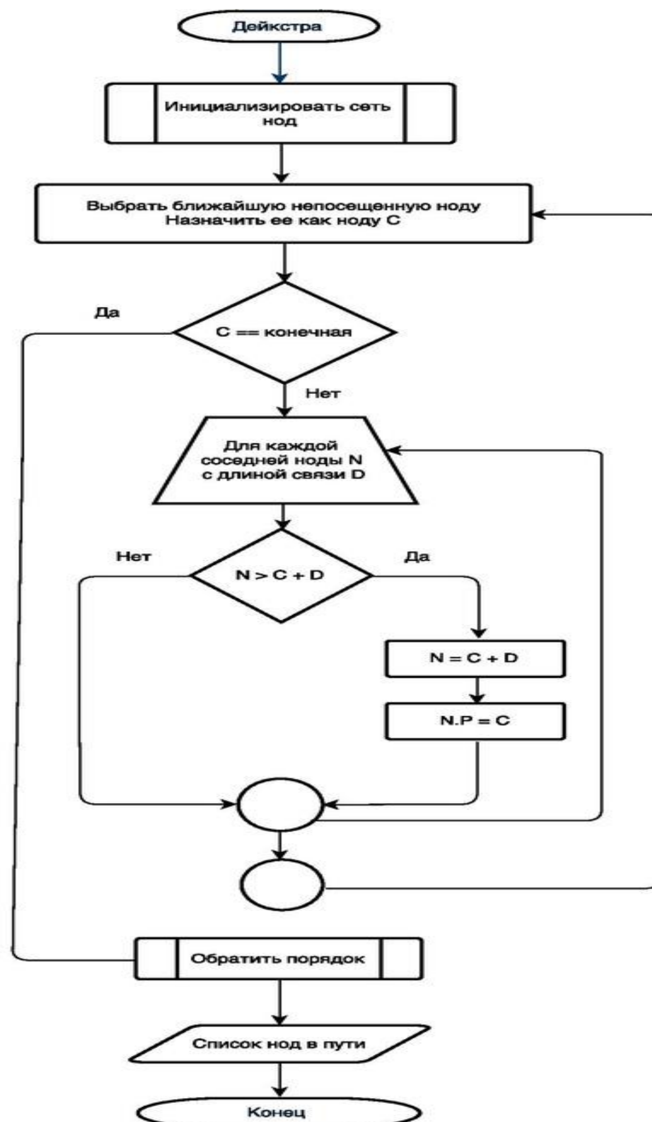


Рисунок 1.1 – Блок-схема алгоритма Дейкстри

Генетичний алгоритм

Генетичний алгоритм – це в першу чергу еволюційний алгоритм, іншими словами, основна особливість алгоритму – схрещування (комбінування). Як нескладно здогадатися ідея алгоритму нахабним чином взята у природи. Так ось, шляхом перебору і найголовніше відбору виходить правильна «комбінація».

Алгоритм ділиться на три етапи:

- схрещування;
- селекція (відбір);
- формування нового покоління.

Якщо результат нас не влаштовує, ці кроки повторюються до тих пір, доки результат нас не почне задовільняти або станеться одна з нижче перерахованих умов:

- кількість поколінь (циклів) досягне заздалегідь обраного максимуму;
- вичерпано час на мутацію[11].

Сам алгоритм складається з декількох кроків.

0. Підготовчий крок – формування початкової популяції (початкового набору рішень). Алгоритм для формування може бути різним, але найчастіше використовують випадковий процес з метою охопити більшу різноманітність для пошуку рішень. Можливо застосування інших способів формування, - наприклад, із заздалегідь відомими властивостями, але слід мати на увазі, що це може вплинути на хід розвитку системи надалі.

1. Відбір – важливий етап в алгоритмі, відповідає за вибір напрямку розвитку популяцій, найчастіше відкидаються рішення з низьким значенням функції пристосованості (fitnessfunction), що сприяє поліпшенню середньої пристосованості всієї популяції.

2. Схрещування – етап, на якому відбувається утворення нових рішень в популяції, що пройшла через відбір, для відновлення чисельності. Особливість його в тому, що при використанні схрещування беруться два або

більше існуючих рішень в популяції, а з них – складові частини (гени) і з’єднуються в новому рішенні, яке залишається в популяції.

3. Схрещування не дозволяє в повній мірі охопити всі можливі варіанти сполучень і значень генів, тому не менш важливий процес мутації. Він полягає в тому, що в деяких рішеннях з популяції відбуваються випадкові зміни в генах. Цей процес сприяє збільшенню різноманітності в популяції.

4. Оцінка рішень і зупинка алгоритму – в більшості випадків; якщо для вирішення завдання необхідно застосовувати генетичний алгоритм, то немає критерію зупину, заснованого на самих рішеннях, замість нього застосовується підхід з числом обчислень (числом створюваних популяцій). Іноді останов можна виробляти наперед, якщо можливий випадок виродження популяцій [6].

Основними кроками алгоритму є кроки з 1 по 4, один прохід по яким «створює нову популяцію».

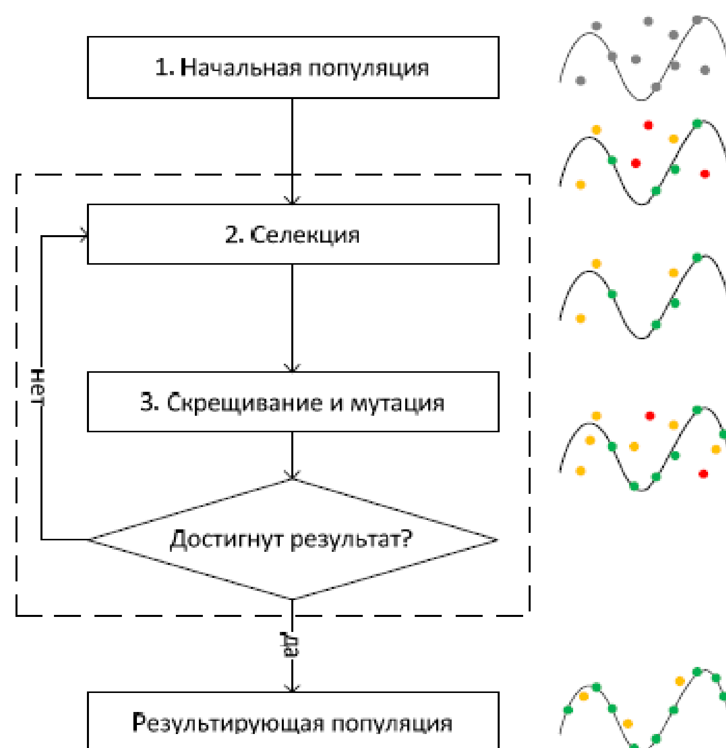


Рисунок 1.2 – Схема роботи генетичного алгоритму різноманітності в популяції

Алгоритм “Мурахи”(Ant Colony Optimization)

Алгоритм ACO(Ant colony optimization) – це спосіб оптимізації на основі популяції, який був розроблений Марко Дориго і успішно застосовується для вирішення кількох NP-важких задач комбінаторної оптимізації. Цей алгоритм був натхненний поведінкою колоній мурашок (або “агентів”), особливо їх поведінкою в процесі пошуку їжі в реальному житті. Зазвичай мурахи, коли залишають свої гнізда, починають безладно переміщатися навколо районів, що оточують їх гнізда в пошуках їжі. Якщо будь-якому з мурах попадається їжа, він спочатку збирає деякі шматочки їжі і на своєму шляху назад в гніздо розпорошує хімічну речовину під назвою феромон – як спосіб спілкування з його колегами, що він знайшов джерело їжі. Інші прилеглі мурахи сприймають аромат феромонів і починають рухатися в напрямку сліду. Після того, як вони виявляють джерело їжі, то оновлюють феромонами слід для попередження інших мурах. Повертаючись, протягом часу кілька мурах отримують цю інформацію і слідують по утвореному шляху [11].

Ще одна цікава частина поведінки мурах полягає в тому, що як тільки вони повертаються в гніздо, вони оптимізують свій маршрут. За короткий час мурахи створюють більш короткий шлях до джерела їжі, ніж попередні. Крім того, якщо на більш короткий маршрут помістити перешкоду, роблячи рух неможливим, мурахи здатні знайти ще один короткий шлях з доступних варіантів обходів перешкоди [1].

Були різні модифікації мурашиних алгоритмів, починаючи з вихідної Ant System (AS), переходячи до Ant Colony System (ACS), потім Min-Max System Ant (MMAC), а потім Ant Colony Optimization (ACO) алгоритмам і так далі.

1.2 Постановка задачі

Задача комівояжера може бути представлена у вигляді математичної моделі – графа G = (V, A) . Множина вершин $V = \{v_0, v_1, \dots v_{n-1}\}$ в графі відповідають містам в задачі комівояжера, множина ребер $A = \{(v_i, v_j) | v_i, v_j \epsilon V, i \neq j\}$ – позначають дороги , що з'єднують пари міст. Кожне ребро з множини A має вагу – d_{ij}. Залежно від формулювання завдання, вага ребра (v_i, v_j) з графа G може означати час переходу, вартість поїздки або відстань між відповідними вершинами v_i і v_j. Контур, що включає кожну вершину графа G хоча б один раз, називається маршрутом комівояжера. Так само контур, що включає кожну вершину графа G рівно один раз, називається контуром Гамільтона. Тоді завданням комівояжера є задача пошуку найменшої загальної довжини Гамільтонового контуру [11].

Розглянемо метричну задачу комівояжера, коли відстані між містами можна обчислити (аналог точок на площині). При даній постановці завдання ми маємо: число міст, координати кожного міста на площині:

$$\{g = (x, y)\}, i = \overline{1, n}. \qquad (1.1)$$

Таким чином, відстань між містами можна знаходити як відстань між двома точками:

$$S(g_i, g_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \qquad (1.2)$$

Необхідно знайти такий шлях через міста $g_{(i)}$, щоб сумарна відстань була мінімальною.

Метою роботи є вирішення завдання комівояжера і побудова мінімального маршруту при заданій конфігурації перешкод і міст за допомогою алгоритму мурахи та генетичного алгоритму. Маршрут не повинен перетинати кордони перешкод, але може проходити по їх кордонах.

1.3 Обгрунтування вибору мови програмування

Не так багато технологій можуть похвалитися тим, що вони є актуальними вже більше 20 років. У 2019 році Java посіла п'яте місце в списку найбільш популярних технологій, поступившись тільки незаперечним лідерам: JavaScript, HTML, CSS і SQL. Java займає 18-е місце в рейтингу улюблених технологій (за результатами опитування StackOverflow).

Java включає в себе об'єктно-орієнтоване програмування (ООП) – концепцію, в якій можна не тільки визначати тип даних і його структуру, а й набір функцій, що застосовуються до нього. Таким чином, структура даних стає об'єктом, яким можна управляти для створення відносин між різними об'єктами.

ООП більш ефективно організовує структуру програм, в тому числі великих та спрощує обслуговування і модернізацію старого коду.

Java забезпечує незалежність від платформи , що дає змогу запускати програму на різних операційних системах. Щоб максимально ефективно використовувати час процесора, Java дозволяє запускати потоки одночасно, що називається багатопоточністю [11].

2 ВИБІР АЛГОРИТМІВ РОЗВ’ЯЗКУ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Загальні технології розв’язку для алгоритму “мурахи” та генетичного алгоритму

Для того щоб знайти в задачі комівояжера довжину маршруту між містами за їх географічними координатами , слід використати одну з формул для обчислення сферичної відстані великого кола.

Існує три способи розрахунку сферичної відстані великого кола між містами з маршруту за даними їх довготи та широти.

1. **Сферична теорема косинусів**. У випадку маленьких відстаней і невеликої розрядності обчислення (кількість знаків після коми) використання формули може призводити до значних помилок, пов'язаних з округленням.

φ1, λ1; φ2, λ2 - широта і довгота двох точок в радіанах, Δλ - різниця координат по довготі, Δδ - кутова різниця, Δδ = arccos {sin φ1 sin φ2 + cos φ1 cos φ2 cos Δλ}. Для переведення кутової відстані в метричну потрібно кутову різницю помножити на радіус Землі (6372795 метрів); одиниці кінцевої відстані дорівнюватимуть одиницям, в яких виражений радіус.

2. **Формула гаверсинусів** - використовується, щоб уникнути проблем з невеликими відстанями.

$$\Delta\sigma = 2\arcsin\left\{\sqrt{\sin^2\left(\frac{\phi_2-\phi_1}{2}\right)+\cos\phi_1\cos\phi_2\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right\}$$

(2.1)

3.**Модифікація для антиподів** (протилежні точки земної кулі – однакова довгота , але різні широти). Попередня формула також схильна до проблеми точок-антиподів. Для того, щоб її вирішити використовується наступна модифікація.

$$\Delta\sigma = \arctan\left\{\frac{\sqrt{\left[\cos\phi_2 \sin\Delta\lambda\right]^2 + \left[\cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda\right]^2}}{\sin\phi_1 \sin\phi_2 + \cos\phi_1 \cos\phi_2 \cos\Delta\lambda}\right\}$$

(2.2)

Дана формула була застосована для підрахунку довжини маршруту між містами у генетичному алгоритмі та алгоритмі “мурахи”[11].

2.2Технології розв’язку генетичного алгоритму

Для застосування генетичного алгоритму необхідно визначити основні структурні елементи: вид елемента популяції, процес схрещування, мутації і вид фітнес-функції.

За елемент популяції (одне можливе рішення) приймаємо маршрут через усі міста. Кожен такий маршрут є можливим рішенням і не суперечить умовам завдання, хоча може бути зовсім не оптимальним. Припускаємо, що всі елементи популяції коректні, тобто всі вони - потенційні вирішення поставленого завдання і не є суперечливими.

Як фітнес-функцію ми приймаємо функцію виду:

$$S = \sum_{i,j \in P} S(g_i, g_j),$$

де **P** - безліч всіх зв'язків в маршруті. (2.3)

Ця функція визначає, що мінімізується параметр і дозволяє оцінювати одержувані рішення. Метод мутації реалізується в такий спосіб: вибираємо випадкове місто; знаходимо зв'язки, що відповідають цьому місту; з'єднуємо сусідні міста прямим зв'язком (виключаємо обране місто з цього зв'язку); вставляємо місто у випадкове місце [5].

Припустимо, що рішення до мутації має вигляд, представлений на рис. 2.1.

Рисунок 2.1 - Рішення до мутації

Застосовуючи до нього алгоритм мутації, можемо отримати рішення у вигляді, представленому на рис. 2.2.

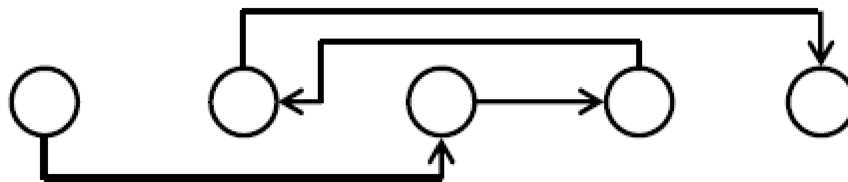


Рисунок 2.2 - Рішення після мутації

Метод схрещування реалізується складніше, так як вимагає додаткових перевірок на коректність одержуваного маршруту.

Алгоритм методу схрещування:

I. вибираємо два маршрути з популяції, які будуть виступати в ролі батьків;
II. в утворені маршрути переносимо зв'язки, які існують в обох маршрутах-батьках; при незбіжних зв'язках перевага віддається зв'язку в першому батьку для міст з парними номерами; якщо її застосувати неможливо, то беремо зв'язки другого батька (надається перевага зв'язкам з другого маршруту для міст з непарними номерами); якщо це також неможливо, то беремо з першого маршруту;
III. при такому завданні схрещування для отримання двох рішень можна застосовувати даний метод двічі, змінюючи маршрути-батьки місцями;
IV. тоді другий нащадок буде отримано при умові, що при розбіжності зв'язків перевага надаватиметься другому маршруту [3].

На рис. 2.3 представлений приклад застосування даного алгоритму схрещування для двох рішень:

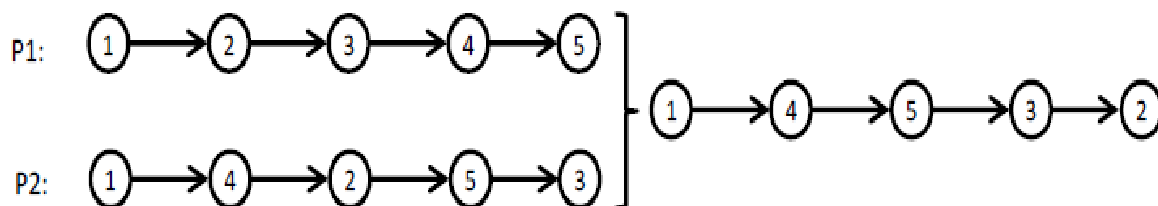


Рисунок 2.3 - Приклад застосування алгоритму схрещування для двох рішень

2.3 Технології розв'язку алгоритму "мурахи"

В АСО колонія мурашок на кожній ітерації вираховує ймовірність, на основі якої мураха в вузлі **i** вибирає наступний вузол **j**, до якого буде рухатися далі. Вибір вузла залежить від значення сліду феромону $\boldsymbol{\tau}_{ij}(t)$ і доступній евристичній η_{ij} У TSP $\eta_{ij} = 1 / d_{ij}$.

Тобто мураха переміщається від місця *i* до *j* з імовірністю:

$$P_{ij}^{t} = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_{i^k}}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}, & if\ j \in N_{i^k} \\ 0, & \text{інакше} \end{cases}$$

(2.4)

У формулі (2.4) $\boldsymbol{\tau}_{ij}(t)$ являє собою слід феромону, η_{ij} представляє локальну евристичну інформацію, t являє собою ітерацію N_ik, що являє собою безліч вузлів, в які мураха може піти, а α і β є параметрами, які керують зміною сліду феромону [11]. До кінця ітерації слід феромону на кожному ребрі ij буде модернізований за допомогою наступного рівняння:

$$\tau_{ij}(t+1) = (1-p) * \tau_{ij}(t) + \Delta\tau_{ij}^{best}(t)$$

(2.5)

У формулі (2.5), $\boldsymbol{\tau}_{ij}(t + 1)$ являє собою слід феромону в ітерації t + 1, параметр ρ∈ [0.1,0.9] відповідає за швидкість випаровування феромону. Кількість феромону, залишена кращим мурахою, представлена у вигляді:

$$\Delta\tau_{ij}^{best}(t) = \begin{cases} \dfrac{1}{f\left(s^{best}(t)\right)}, & \text{якщо краща мураха пройшла в ітерації } t \text{ по дузі } ij \\ 0, & \text{інакше} \end{cases}$$

(2.6)

В формулі (2.6), $f\left(s^{best}(t)\right)$ являє собою вартість кращого рішення $s^{best}(t)$.

Схема роботи алгоритму мурахи представлена на рис.2.4:

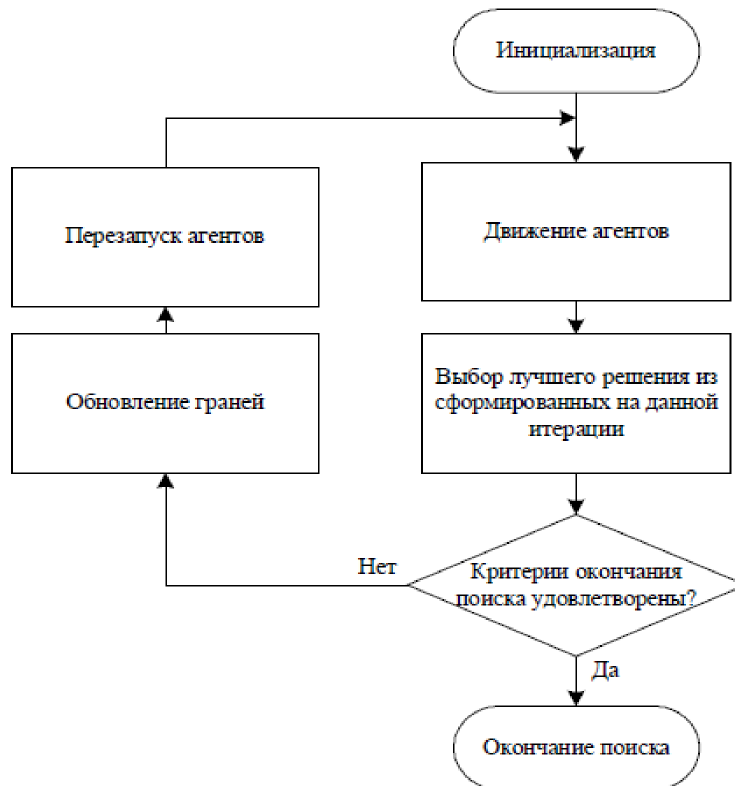


Рисунок 2.4 - Схема роботи алгоритму мурахи

Початкова популяція

Після створення популяція мурах порівну розподіляється по вузлах мережі. Необхідно рівне розділення мурах між вузлами, щоб всі вузли мали однакові шанси стати відправною точкою. Якщо всі мурашки почнуть рух з однієї точки, це буде означати, що дана точка є оптимальною для старту, але насправді ми цього не знаємо.

Для кожної мурахи перехід з міста i в місто j залежить від трьох складових: пам'яті мурашки - списку TabuList, видимості і віртуального сліду феромону.

TabuList (пам'ять мурашки) - це список, в якому зберігається бітовий масив, за допомогою якого визначаються відвідані і не відвідані вузли. Таким чином, агент повинен проходити через кожний вузол тільки один раз. Вузли

в списку "поточної подорожі" Path розташовуються в тому порядку, в якому агент відвідував їх. Пізніше список використовується для визначення довжини шляху між вузлами. Зрозуміло, що список tabu зростає при здійсненні маршруту і обнуляється на початку кожної ітерації алгоритму. Позначимо через $J_{i,k}$ список міст, які ще необхідно відвідати мурасі, що знаходиться в місті.

Видимість - величина, зворотна відстані: $\eta_{ij} = 1/L_{ij}$, де L_{ij} - відстань між містами ***i*** та ***j*** . Видимість - це локальна статична інформація, що виражає евристичне бажання відвідати місто ***j*** з міста ***i*** - чим ближче місто, тим більше бажання відвідати його. Використання тільки видимості, звичайно, є недостатнім для знаходження оптимального маршруту.

Віртуальний слід феромону на ребрі(***ij***) (представляє підтверджене мурашиним досвідом бажання відвідати місто ***j*** з міста ***i)***. Цей параметр характеризує перевагу вибору даної грані в порівнянні з іншими при переміщенні. Інформація про феромони граней змінюється в процесі складання рішень.

При цьому кількість феромонів, що залишається агентами, пропорційна якості рішення, складеного відповідним агентом: чим менше шлях, тим більше буде залишено феромону, і навпаки, чим довший шлях, тим менше буде залишено феромону на відповідних ребрах. Такий підхід дозволяє забезпечити безпосередній пошук в напрямку знаходження кращого рішення.

На відміну від видимості, слід феромону є більш глобальною і динамічною інформацією - вона змінюється після кожної ітерації алгоритму, відображаючи придбаний мурахами досвід. Кількість віртуального феромону на ребрі (***ij***) на ітерації ***t*** позначимо як $\tau_{ij}(t)$ [11].

Рух мурахи

Рух мурашки грунтується на одному вірогідно-пропорційному правилі, яке визначає ймовірність переходу $\boldsymbol{k}$-го мурашки з міста $\boldsymbol{i}$ в місто $\boldsymbol{j}$, якщо мураха ще не закінчив шлях (path), тобто не відвідав усі вузли мережі:

$$\begin{cases} P_{ij,k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{l \in J_{i,k}} [\tau_{ij}(t)]^{б} \cdot [з_{ij}(t)]^{в}} < Rand(1), \text{если } j \in J_{i,k}, \\ P_{ij,k}(t) = 0, \text{если } j \notin J_{i,k} \end{cases} \quad (2.7)$$

де:

- $P_{ij,k}(t)$ - ймовірність того, що мураха переміститися в вузол $\boldsymbol{j}$ з вузла $\boldsymbol{i}$.
- $Rand(1)$ - випадкове число в інтервалі (0; 1);
- *б* - параметр, що задає ваги сліду феромону при виборі маршруту. При *б*=0 буде обране найближче місто, що відповідає жадібному алгоритму в класичній теорії оптимізації.
- *в* - параметр, що задає ваги видимості (відстаней). Якщо *в*=0, тоді працює лише феромонне посилення, що тягне за собою швидке виродження маршрутів до одного субоптимального рішення.

Звернемо увагу, що правило (2.7) визначає лише можливості вибору того чи іншого міста. Сам вибір міста здійснюється за принципом «колеса рулетки»: кожне місто на своєму секторі зі своєю площею, пропорційної ймовірності. Для вибору міст потрібно відмовитися від гонорару - спродукувати випадкове число, і визначити сектор, на якому ця кулька зупиниться. Звернемо увагу, що хоча правило (2.7) не змінюється впродовж ітерації, значення ймовірностей $P_{i,j,k}(t)$ для двох мурах в одному і тому ж місті можуть відрізнятися, так як $P_{i,j,k}(t)$ – функція від $J_{i,k}$ - списку ще не відвіданих міст мурахою k.

Пройдений мурахою шлях закінчується, коли мураха відвідає всі вузли графа. Зверніть увагу, що цикли заборонені, оскільки в алгоритм включений список табу. Після завершення довжина шляху може бути підрахована - вона дорівнює сумі всіх граней, за якими подорожував мураха. Рівняння (2.8) показує кількість феромону, який був залишений на кожній грані шляху для мурахи k. Змінна Q є константою.

$$\Delta\phi_{ij,k}(t) = \frac{Q}{L_k(t)} = \begin{cases} \frac{Q}{L_k(t)}, \text{якщо } (ij) \in T_k(t) \\ 0, \text{якщо } (ij) \notin T_k(t) \end{cases}, \tag{2.8}$$

де $T_k(t)$ — маршрут, пройдений мурахою k на ітерації t; $L_k(t)$ — довжина цього маршруту; Q - регульований параметр, значення якого вибирають одного порядку з довжиною оптимального маршруту.

Результат рівняння є засобом вимірювання шляху (короткий шлях характеризується високою концентрацією феромону, а довший шлях - більш низькою). Потім отриманий результат використовується в рівнянні (2.9), щоб збільшити кількість феромону уздовж кожної грані пройденого мурахою шляху.

$$\phi_{ij,k}(t+1) = \phi_{ij}(t) + \mathrm{c} \cdot \sum_{k=1}^{N_{ij}} \Delta\phi_{ij}(t), \tag{2.9}$$

де c – коефіцієнт випаровування феромону, $c \in [0, 1]$; i і j - вузли, що утворюють грані, які відвідала k-та мураха; N_{ij} – загальна кількість клієнтів, що відвідали грань(ij) [11].

Звернемо увагу, що дане рівняння застосовується до всього шляху, при цьому кожна грань позначається феромоном пропорційно довжині шляху. Тому слід дочекатися, поки мураха закінчить подорож і тільки потім оновити рівні феромону; в іншому випадку справжня довжина шляху залишиться невідомою. На початку шляху у кожної грані є шанс виграти. Щоб поступово

видалити межі, які входять в гірші шляхи мережі, до всіх граней застосовується процедура випаровування феромону (Pheromone evaporation). Використовуючи константу з рівняння (2.9), ми отримуємо формулу (2.10).

$$\phi_{ij}(t) = \phi_{ij}(t) \cdot (1 - c)$$

(2.10)

На початку оптимізації кількість феромону приймається рівною невеликому позитивному числу $initialPheromone$. Загальна кількість мурах в колонії залишається сталою впродовж виконання алгоритму. Численна колонія призводить до швидкого посилення кількості субоптимальних маршрутів, а коли мурах мало, виникає небезпека втрати кооперативності поведінки через обмежену взаємодію і швидке випаровування феромону. Зазвичай число мурах призначають рівною кількості міст - кожен мураха починає маршрут зі свого міста. Після того як шлях мурашки завершений, грані оновлені відповідно до довжини шляху і відбулося випаровування феромону на всіх гранях, алгоритм запускається повторно. Список табу очищується, і довжина шляху обнуляється. Мурашкам дозволяється переміщатися по мережі, обгрунтовуючи вибір межі на рівнянні. Цей процес може виконуватися для постійної кількості шляхів або до моменту, коли впродовж декількох запусків не було відзначено повторних змін. Потім визначається кращий шлях, який і є рішенням [2].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Описання основних java – класів та констант генетичного алгоритму для задачі комівояжера

Приведено код програми мовою Java, що вирішує задачу комівояжера за допомогою генетичного алгоритму. Було створено 5 java – класів:

1. **City.java –** представляє місто на маршруті
2. **Route.java** - представляє рішення для кандидата на маршрут, який буде починатися з одного з міст і проходить через всі міста і повертається до початку шляху
3. **Population.java** – популяція маршруту
4. **GeneticAlgorithm.java –** основна реалізація процесів генетичного алгоритму – селекції, мутації, схрещування, елітизму.
5. **Driver.java –** основний клас, в якому зберігається масив міст маршруту та виводиться інформація на екран про результати роботи програми [11].

Основні константи, що використовуються в програмі

Клас **genetic algorithm** зі статичними незмінними полями (константами)

public class genetic algorithm {

public static final double MUTATION_RATE = 0.25; //**рівень мутації**

public static final int TOURNAMENT_SELECTION_SIZE = 3;//**для схрещування хромосом(шляхів) в поколінні**

public static final int POPULATION_SIZE = 8**; //кількість маршрутів у поколінні**

public static final int NUMBER_OF_ELITE_ROUTES = 1; //**кількість елітних маршрутів**

public static final int NUMBER_OF_GENERATIONS = 30; //**кількість поколінь -від 0 до 29 покоління** }

Класс **City** (створюємо місто маршруту) зі статичними незмінними полями (константами) та інші поля класу, що відповідають за довготу, широту та назву міста.

Аналогічні константи і поля використовуються в класі **AntCity.java** для алгоритму “мурахи”

```
public class City {
    private static final double EARTH_EQUATORIAL_RADIUS = 6378.1370D;
    // константа, що збрегігає значення екваторіального радіуса
    private static final double CONVERT_DEGREES_TO_RADIANS = Math.PI/180D; // константа для переведення градусів у радіани
    private static final double CONVERT_KM_TO_MILES = 0.621371;
    //константа для переведення кілометрів у милі
    private double longtitude;//довгота міста
    private double latitude;//широта міста
    private String name;//ім'я міста
     …………. }
```

private static final double CONVERT_KM_TO_MILES = 0.621371; //**переведення кілометрів у милі** [11].

3.2 Ключові функції для реалізації генетичного алгоритму

Функція классу Route.java для підрахунку загальної відстані між містами

public double calculateTotalDistance(){//сумуємо відстань між сусідніми містами та між останнім містом і початком маршруту

int citiesSize = this.cities.size();//рахуємо кількість міст маршруту

return (int)(this.cities.stream().mapToDouble(x -> {

int cityIndex = this.cities.indexOf(x);//індекс першого входження конкретного символу

double returnValue = 0;

if (cityIndex < citiesSize - 1)

returnValue = x.measureDistance(this.cities.get(cityIndex+1));

return returnValue;

}).sum() + this.cities.get(0).measureDistance(this.cities.get(citiesSize - 1)));}

Функція класу City.java для підрахунку відстані між двома містами

public double measureDistance(City city){//вимірюємо відстань між цим містом і минулим містом

double deltaLongtitude = (city.getLongtitude()-this.getLongtitude ());

double a1 = Math.sqrt(Math.pow(Math.cos(city.getLatitude())* Math.sin(deltaLongtitude),2D) +

Math. pow (Math.cos(this. getLatitude ())* Math.sin(city. getLatitude ()) - Math.sin(this.getLatitude())*Math.cos(city.getLatitude())*Math.cos(deltaLongtitude),2D));

double b1 = Math.sin(this.getLatitude())*Math.sin(city.getLatitude()) + Math.cos(this.getLatitude()) *

Math.cos(city. getLatitude ())*Math.cos(deltaLongtitude);

return CONVERT_KM_TO_MILES * EARTH_EQUATORIAL_RADIUS * Math.atan2(a1,b1);

}

Аналогічні функції використовуються для алгоритму “мурахи”.

Функція класу GeneticAlgorithm.java для схрещування популяції

```
Population corssoverPopulation(Population population){
Population crossoverPopulation = new Population(population.getRoutes().size(),this);
//створюємо нову популяцію; її розмір - передаємо в екземпляр класу Population - (crossoverPopulation)
IntStream.range(0,NUMBER_OF_ELITE_ROUTES).forEach(x -> crossoverPopulation.getRoutes().set(x,population.getRoutes().get(x)));
IntStream.range(NUMBER_OF_ELITE_ROUTES,crossoverPopulation.getRoutes().size()).forEach(x ->{//записуємо попердню популяцію
Route route1 = selectTournamentPopulation(population).getRoutes().get(0);
Route route2 = selectTournamentPopulation(population).getRoutes().get(0);
crossoverPopulation.getRoutes().set(x,crossoverRoute(route1, route2));
});
return crossoverPopulation;}
```

Функція класу GeneticAlgorithm.java для схрещування маршрутів

```
Route crossoverRoute (Route route1,Route route2){
Route crossoverRoute = new Route (this);
Route tempRoute1 = route1;
Route tempRoute2 = route2;
if (Math.random() < 0.5){
tempRoute1 = route2;
tempRoute2 = route1;}
for (int x=0;x < crossoverRoute.getCities().size()/2;x++)
crossoverRoute. getCities ().set(x,tempRoute1.getCities().get(x));
return fillNullsInCrossoverRoute(CrossoverRoute,tempRoute2);
```

```
}
private Route fillNullsInCrossoverRoute(Route crossoverRoute,Route route){// route – route2
route.getCities().stream().filter( x ->!crossoverRoute.getCities().contains(x)).forEach(cityX ->{
for (int y=0; y < route.getCities().size();y++){
if (crossoverRoute.getCities().get(y) == null){
crossoverRoute.getCities().set(y,cityX);
//перевіряємо чи є вже наявне місто з даного маршруту
// в схрещуваному маршруті і додаємо місто, якщо воно відсутнє
break;
}}
});
return crossoverRoute;
}
```

Функція класу GeneticAlgorithm.java для мутації в популяції

```
Population mutatePopulation(Population population){
population.getRoutes().stream().folter(x -> population.getRoutes().indexOf(x) >= NUMBER_OF_ELITE_ROUTES).forEach(x->mutateRoute(x));
return population;//для кожного з неелітних маршрутів ми викликаємо метод mutateRoute ()
}
```

Функція класу GeneticAlgorithm.java, що повертає найкращу популяцію відповідно до фітнес - функції

```
Population selectTornamentPopulation(Population population){
Population tournamentPopulation = new Population(TOURNAMENT_SELECTION_SIZE,this);
IntStream.range(0,TOURNAMENT_SELECTION_SIZE).forEach(x -> tournamentPopulation.getRoutes().set(
```

```
x.population.getRoutes().get((int)(Math.random() *
population.getRoutes ().size())));
tournamentPopulation.sortRoutesByFitness ();
return tournamentPopulation;}
```

Функція класу Route.java, що повертає значення фітнес - функції

```
public double getFitness(){
if (isFitnessChanged == true){
fitness = (1/calculateTotalDistance()) * 10000;//для зручності множимо значення на 10000
isFitnessChanged = false;
}
return fitness;
}
```

3.3 Описання основних java – класів та констант алгоритму "мурахи" для задачі комівояжера

Приведено код програми мовою Java, що вирішує задачу комівояжера за допомогою **алгоритму "мурахи"**. Було створено 6 java – класів:

1. **AntCity.java –** представляє місто на маршруті.
2. **AntRoute.java** - представляє рішення для мурахи – маршрут, який буде починатися з одного з міст і проходить через усі міста і повертається до початку шляху.
3. **Ant.java**– опис алгоритму проходження маршруту для мурахи.
4. **AntColonyOptmz.java –** опис методів для застосувння алгоритму мурахи для кожної особи.
5. **DriverAnt.java –** основний клас, в якому зберігається масив міст маршруту та виводиться інформація на екран про результати роботи програми.
6. **AtomicDouble.java –** допоміжний (службовий) клас[11].

Основні константи, що використовуються в програмі

Клас **DriverAnt** зі статичними незмінними полями (константами) та масивом, що містить список міст маршруту.

```
public class DriverAnt {
   static final int NUMBER_OF_ANTS = 500; // константа, що зберігає кількість мурах, які будуть шукати найкоротший шлях
   static final double PROCESSING_CYCLE_PROBABILITY = 0.8;
// константа, що зберігає значення ймовірністі циклу обробки
static ArrayList<AntCity> initialRoute = new ArrayList< AntCity>(Arrays.AsList(
            new AntCity(34.8003 ,50.9216,"Sumy"),
            new AntCity(30.5238,50.4547,"Kyiv"),
            new AntCity(36.2527,49.9808,"Kharkiv"),
            new AntCity (34.5407,49.5937,"Poltava"),
            new AntCity (24.0232 ,49.8383,"Lviv"),
```

new AntCity (30.7326,46.4775,"Odessa"),

new AntCity (37.8022,48.0230,"Donetsk"),

new AntCity (28.6767,50.2649,"Zhytomyr"),

new AntCity (25.6056,49.5559,"Ternopil"),

new AntCity (34.9833,48.4500,"Dnipro")

));

Клас **Ant** зі статичними незмінними полями (константами) та іншими полями , що відповідають за об’єкт “мураха”, її номер, маршрут[11].

public class Ant implements Callable< Ant > {

public static final double Q = 0.005;//**Значення між 0 та 1, константа, що зберігає значення для регулювання рівня залишеного феромону**

public static final double RHO = 0.2;//**константа , що зберігає значення для зміни рівня випаровування феромону (від 0 до 1)**

public static final double ALPHA = 0.01;//**константа , що зберігає значення для контролю важливості сліду феромону**

public static final double BETA = 9.5;//**константа, що зберігає значення для контролю важливості відстані між джерелом та містом призначення**

static int invalidCityIndex = -1; // **константа, що зберігає значення для невірно вибраного міста**

static int numbOfAntCities = DriverAnt.initialRoute.size();//**константа, що зберігає значення кількості міст маршруту ……** }

Клас **AntColonyOptmz** містить поля , що зберігають матрицю відстаней між містами та рівень феромону, що виділяє мураха.

public class AntColonyOptmz {

private AtomicDouble [][] PheromoneLevelMatrix = null;//матриця, в якій зберігається величина феромону мурахи

private double[][] distanceMatrix = null;//матриця відстаней між містами …..}

3.4 Ключові функції для реалізації алгоритму “мурахи”

Метод **call()** класу **Ant**, який будує маршрут для кожної мурахи

```
public Ant call() throws Exception {
int originatingAntCityIndex = ThreadLocalRandom.current().nextInt(numbOfAntCities);
ArrayList<AntCity> routeAntCities = new ArrayList<AntCIty>(numbOfAntCities); //всі міста маршруту для поточної мурахи
HashMap<String,Boolean> visitedAntCities = new HashMap<String, Boolean>( numbOfAntCities);
IntStream.range(0,numbOfAntCities).forEach(x-> visitedAntCities.put(DriverAnt.initialRoute.get(x).getName(),false));
//для кожної мурахи первіряється список всіх міст маршруту і для НЕ відвіданих маршрутів ставиться мітка False
int numbOfVisitedAntCities = 0;
visitedAntCities.put(DriverAnt.initialRoute.get(originatingAntCityIndex).getName(),true);
double RouteDistance = 0.0;
int x = originatingAntCityIndex;
int y = invalidCityIndex;
if (numbOfVisitedAntCities != numbOfAntCities) y = getY(x, visitedAntCities);
while ( y!= invalidCityIndex){
routeAntCities.add(numbOfVisitedAntCities++,DriverAnt.initialRoute.get(x));
routeDistance += aco.getDistanceMatrix()[x][y];
adjustPheromoneLevel(x,y,routeDistance);//метод регуляції рівня феромону
visitedAntCities.put(DriverAnt. initialRoute.get(y).getName(),true);
```

```
x = y;//індекс кінця маршруту прирівнюємо до індексу початку маршруту
if(numbOfVisitedAntCities!= numbOfAntCities)
y = getY(x,visitedAntCities);
else y = invalidCityIndex;
}
routeDistance += aco.getDistanceMatrix()[x][originatingAntCitIndex];
routeAntCities.add(numbOfVisitedAntCities,DriverAnt.initialRoute.get(x));//
додаємо останнє місто в список відвіданих міст
routeAnt = new RouteAnt(routeAntCities,routDistance);
return this;
}
```

Метод **adjustPheromoneLevel ()** класу **Ant**, який регулює рівень феромону після кожного пройденого мурахою маршруту

```
private void adjustPheromoneLevel(int x ,int y , double distance){//регулювати рівень феромону
boolean flag = false;
while (!flag){
double currentPheromoneLevel = aco.getPheromoneLevelMatrix()[x][y].doubleValue();
double updatedPheromoneLevel = (1-RHO)*currentPheromoneLevel + Q/distance;
if (updatedPheromoneLevel < 0.00) flag = aco.getPheromoneLevelMatrix()[x][y].compareAndSet(0);
//якщо феромон випарувася, то задаємо рівень феромону = 0
else flag = aco.getPheromoneLevelMatrix()[x][y].compareAndSet(updatedPheromoneLevel);
}
```

```
}
```

Метод **int getY ()** класу **Ant**, який будує оптимальний маршрут на основі ймовірності переходу між містами

```
private int getY(int x ,HashMap<String,Boolean> visitedAntCities){//індекс міста початку маршруту, хеш мапу всіх відвіданих міст
    //і метод поверне індекс наступного міста маршруту
    int returnY = invalidCityIndex;
    double random = ThreadLocalRandom.current().nextDouble();
    ArrayList<Double> transitionProbabilities = getTransitionProbavilities(x,visitedAntCities);
    for (int y =0;y < numberOfAntCities;y++){
        if (transitionProbabilities.get(y) > random){
            returnY = y;//індекс кінцевого міста маршруту
            break;
        }else random -=transitionProbabilities.get(y);
    }
    return returnY;
}
```

Метод **getTransitionProbabilities** класу **Ant**, який обчислює ймовірність переходу мурахи між містами на основі рівня феромону.

```
private ArrayList<Double> getTransitionProbabilities(int x,HashMap<String,Boolean> visitedAntCities) {
    //ймовірність переходу мурахи з одного міста в інше (від міста X до інших міст)
    ArrayList<Double> transitionProbabilities = new ArrayList<Double>(numbOfAntCities);
    IntStream.range(0,numbofAntCities).forEach(i->
transitionProbabilities.add(0.0));
```

```
double denominator = getTPDEnomonator(transitionProbabilities,x,visitedAntCities);//x - індекс початку маршруту
IntStream.range(0,numbOfAntCities).forEach(y->
transitonProbabilities.set(y,transitionProbavilities.get(y)/denomintor));
return  transitonProbabilities;
}[11].
```

3.5 Дослідження роботи генетичного алгоритму в залежності від кількості міст маршруту

Дослідимо залежність якості рішення генетичного алгоритму від різноманітних параметрів.

Спочатку будемо змінювати кількість міст маршруту **від 10 до 25**

1) Базові параметри запуску алгоритму:
   - 30 поколінь – **10 міст**;
   - рівень мутації = **0.25**;
   - кількість поколінь у маршруті = **8**.

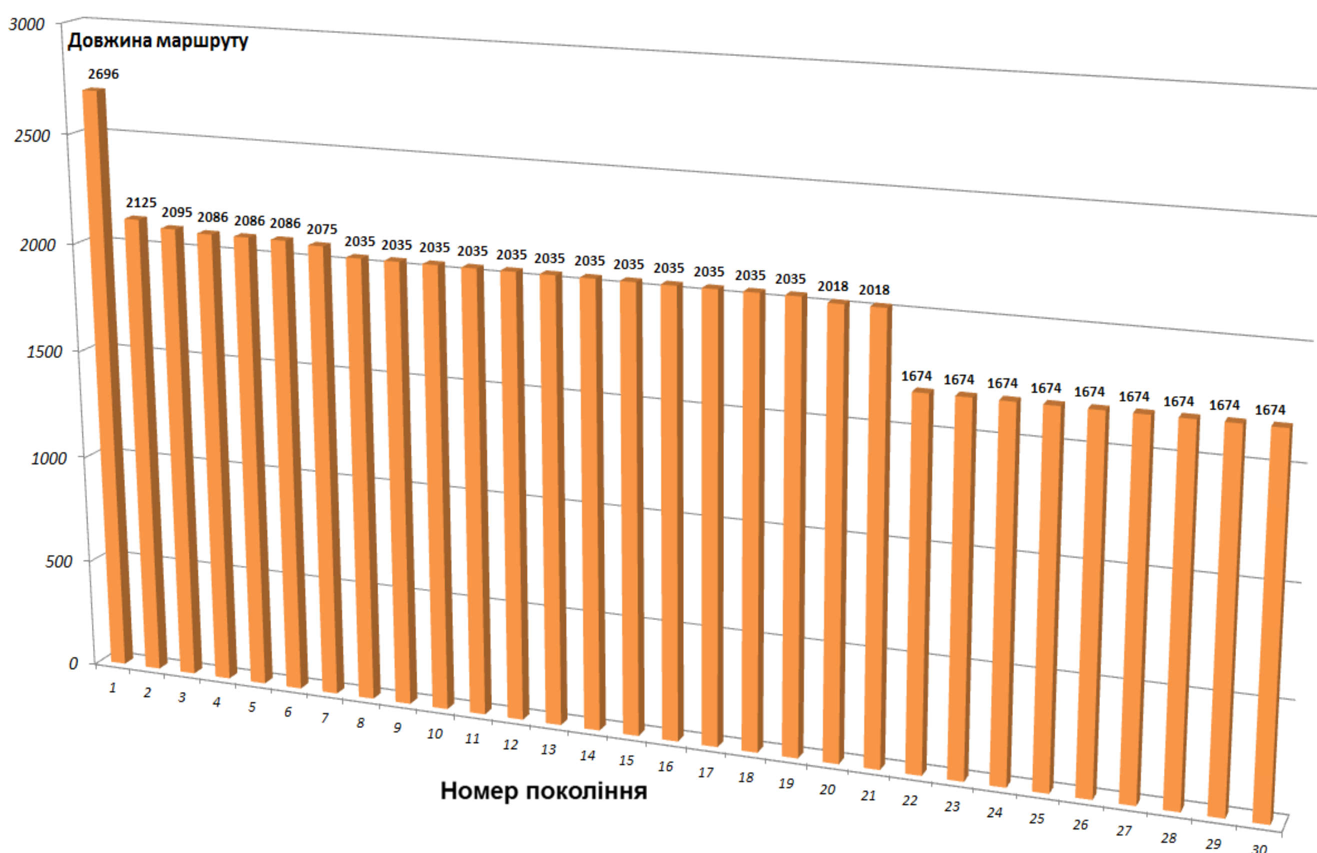


Рисунок 3.1 – Графік залежності найкоротшої відстані від номеру покоління

Скріншоти запуску програмного коду з результатами виконня програми для генетичного алгоритму:

```
> Generation #0
Route| Fitness | Distance(in miles)
-------------------------------------------------------------------------------------------------------------
[Lviv, Kyiv, Zhytomyr, Poltava, Kharkiv, Dnipro, Odessa, Ternopil, Sumy, Donetsk]|3,709199|2696,000000
[Kharkiv, Sumy, Odessa, Poltava, Lviv, Ternopil, Kyiv, Donetsk, Zhytomyr, Dnipro]|3,649635|2740,000000
[Lviv, Donetsk, Dnipro, Odessa, Kharkiv, Poltava, Zhytomyr, Ternopil, Kyiv, Sumy]|3,633721|2752,000000
[Kyiv, Ternopil, Poltava, Zhytomyr, Donetsk, Kharkiv, Dnipro, Odessa, Sumy, Lviv]|3,357958|2978,000000
[Zhytomyr, Donetsk, Sumy, Odessa, Ternopil, Kharkiv, Dnipro, Lviv, Poltava, Kyiv]|3,123048|3202,000000
[Ternopil, Dnipro, Lviv, Kharkiv, Donetsk, Zhytomyr, Odessa, Kyiv, Sumy, Poltava]|3,020236|3311,000000
[Poltava, Ternopil, Sumy, Lviv, Dnipro, Donetsk, Odessa, Zhytomyr, Kharkiv, Kyiv]|2,991325|3343,000000
[Donetsk, Dnipro, Sumy, Lviv, Poltava, Ternopil, Kharkiv, Zhytomyr, Odessa, Kyiv]|2,948983|3391,000000
```

Рисунок 3.2 - Скріншот запуску програми генетичного алгоритму для 1 покоління

```
> Generation #29
Route| Fitness | Distance(in miles)
-------------------------------------------------------------------------------------------------------------
[Kyiv, Dnipro, Donetsk, Kharkiv, Sumy, Poltava, Odessa, Ternopil, Lviv, Zhytomyr]|5,973716|1674,000000
[Kyiv, Dnipro, Donetsk, Sumy, Kharkiv, Poltava, Lviv, Ternopil, Odessa, Zhytomyr]|4,965243|2014,000000
[Kyiv, Donetsk, Dnipro, Kharkiv, Odessa, Poltava, Sumy, Ternopil, Lviv, Zhytomyr]|4,703669|2126,000000
[Dnipro, Sumy, Poltava, Kharkiv, Zhytomyr, Ternopil, Lviv, Odessa, Donetsk, Kyiv]|4,448399|2248,000000
[Kyiv, Zhytomyr, Poltava, Kharkiv, Sumy, Donetsk, Odessa, Ternopil, Lviv, Dnipro]|4,444444|2250,000000
[Donetsk, Zhytomyr, Kyiv, Kharkiv, Sumy, Poltava, Odessa, Ternopil, Lviv, Dnipro]|4,405286|2270,000000
[Kyiv, Kharkiv, Donetsk, Odessa, Sumy, Zhytomyr, Dnipro, Ternopil, Lviv, Poltava]|3,494060|2862,000000
[Kyiv, Odessa, Kharkiv, Donetsk, Ternopil, Dnipro, Zhytomyr, Poltava, Lviv, Sumy]|2,862869|3493,000000
```

Рисунок 3.3 - Скріншот запуску програми генетичного алгоритму для 30 покоління

```
Найкращий маршрут знайдений до цих пір: [Kyiv, Dnipro, Donetsk, Kharkiv, Sumy, Poltava, Odessa, Ternopil, Lviv, Zhytomyr]
 довжиною в : 1674,000000 миль
```

Рисунок 3.4 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

2) Базові параметри запуску алгоритму:

- 30 поколінь – **15 міст**;
- рівень мутації = 0.25;
- кількість поколінь у маршруті = 8.

```
new City( longitude: 7.4660, latitude: 51.5149, name: "Dortmund"),

new City( longitude: 11.5754, latitude: 48.1374, name: "Munich"),

new City( longitude: 6.9843, latitude: 51.0303, name: "Leverkusen"),

new City( longitude: 12.3713, latitude: 51.3396, name: "Leipzig"),

new City( longitude: 8.2791, latitude: 49.9842, name: "Mainz")
```

Рисунок 3.5 - Скріншот доданих міст для виконання генетичного алгоритму

Скріншоти запуску програмного коду з результатами виконання програми для генетичного алгоритму:

```
> Generation #0
Route| Fitness | Distance(in miles)
------------------------------------------------------------------------------------------------------------------------------------
[Ternopil, Leverkusen, Sumy, Zhytomyr, Munich, Lviv, Poltava, Kharkiv, Donetsk, Dnipro, Leipzig, Kyiv, Mainz, Dortmund, Odessa]|1,131862|8835,000000
[Leipzig, Munich, Poltava, Donetsk, Zhytomyr, Sumy, Leverkusen, Dortmund, Dnipro, Kyiv, Ternopil, Mainz, Kharkiv, Lviv, Odessa]|1,116819|8954,000000
[Kharkiv, Kyiv, Dortmund, Donetsk, Dnipro, Poltava, Odessa, Lviv, Leverkusen, Munich, Ternopil, Mainz, Sumy, Leipzig, Zhytomyr]|1,093135|9148,000000
[Poltava, Dnipro, Dortmund, Kyiv, Zhytomyr, Odessa, Mainz, Munich, Donetsk, Leverkusen, Lviv, Kharkiv, Ternopil, Leipzig, Sumy]|1,003915|9961,000000
[Sumy, Kyiv, Dortmund, Leipzig, Lviv, Odessa, Dnipro, Ternopil, Mainz, Kharkiv, Poltava, Leverkusen, Donetsk, Zhytomyr, Munich]|1,003915|9961,000000
[Poltava, Kharkiv, Munich, Ternopil, Odessa, Leipzig, Mainz, Zhytomyr, Leverkusen, Kyiv, Dnipro, Donetsk, Dortmund, Sumy, Lviv]|0,998502|10015,000000
[Zhytomyr, Dnipro, Donetsk, Ternopil, Leverkusen, Poltava, Leipzig, Mainz, Sumy, Munich, Lviv, Kyiv, Dortmund, Kharkiv, Odessa]|0,979720|10207,000000
[Mainz, Kharkiv, Leipzig, Kyiv, Poltava, Lviv, Dnipro, Odessa, Dortmund, Leverkusen, Sumy, Ternopil, Munich, Donetsk, Zhytomyr]|0,957763|10441,000000
```

Рисунок 3.6 - Скріншот запуску програми генетичного алгоритму для 1 покоління

```
> Generation #29
Route| Fitness | Distance(in miles)
------------------------------------------------------------------------------------------------------------------------------------
[Kharkiv, Kyiv, Sumy, Poltava, Lviv, Munich, Ternopil, Zhytomyr, Mainz, Leverkusen, Dortmund, Leipzig, Dnipro, Donetsk, Odessa]|1,826150|5476,000000
[Kharkiv, Odessa, Donetsk, Poltava, Lviv, Munich, Ternopil, Zhytomyr, Mainz, Dortmund, Leverkusen, Leipzig, Dnipro, Sumy, Kyiv]|1,770538|5648,000000
[Odessa, Kyiv, Zhytomyr, Dortmund, Munich, Leverkusen, Mainz, Ternopil, Dnipro, Kharkiv, Donetsk, Lviv, Poltava, Leipzig, Sumy]|1,459002|6854,000000
[Ternopil, Leipzig, Lviv, Odessa, Dnipro, Munich, Zhytomyr, Kharkiv, Poltava, Sumy, Leverkusen, Mainz, Dortmund, Donetsk, Kyiv]|1,336362|7483,000000
[Sumy, Kyiv, Kharkiv, Dnipro, Lviv, Dortmund, Ternopil, Donetsk, Mainz, Odessa, Poltava, Leverkusen, Munich, Leipzig, Zhytomyr]|1,167815|8563,000000
[Kharkiv, Mainz, Sumy, Poltava, Lviv, Munich, Ternopil, Leipzig, Dnipro, Zhytomyr, Odessa, Kyiv, Leverkusen, Dortmund, Donetsk]|1,080847|9252,000000
[Mainz, Kyiv, Ternopil, Poltava, Leverkusen, Munich, Lviv, Leipzig, Kharkiv, Odessa, Donetsk, Sumy, Dortmund, Dnipro, Zhytomyr]|1,018226|9821,000000
[Odessa, Dnipro, Dortmund, Poltava, Leverkusen, Kharkiv, Ternopil, Munich, Donetsk, Zhytomyr, Mainz, Lviv, Sumy, Leipzig, Kyiv]|0,829944|12049,000000
```

Рисунок 3.7 - Скріншот запуску програми генетичного алгоритму для 30 покоління

```
Найкращий маршрут знайдений до цих пір: [Kharkiv, Kyiv, Sumy, Poltava, Lviv, Munich, Ternopil, Zhytomyr, Mainz, Leverkusen, Dortmund, Leipzig, Dnipro, Donetsk, Odessa]
 довжиною в : 5476,000000 миль
Час виконання: 325 мілісекунд(-и)
```

Рисунок 3.8 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

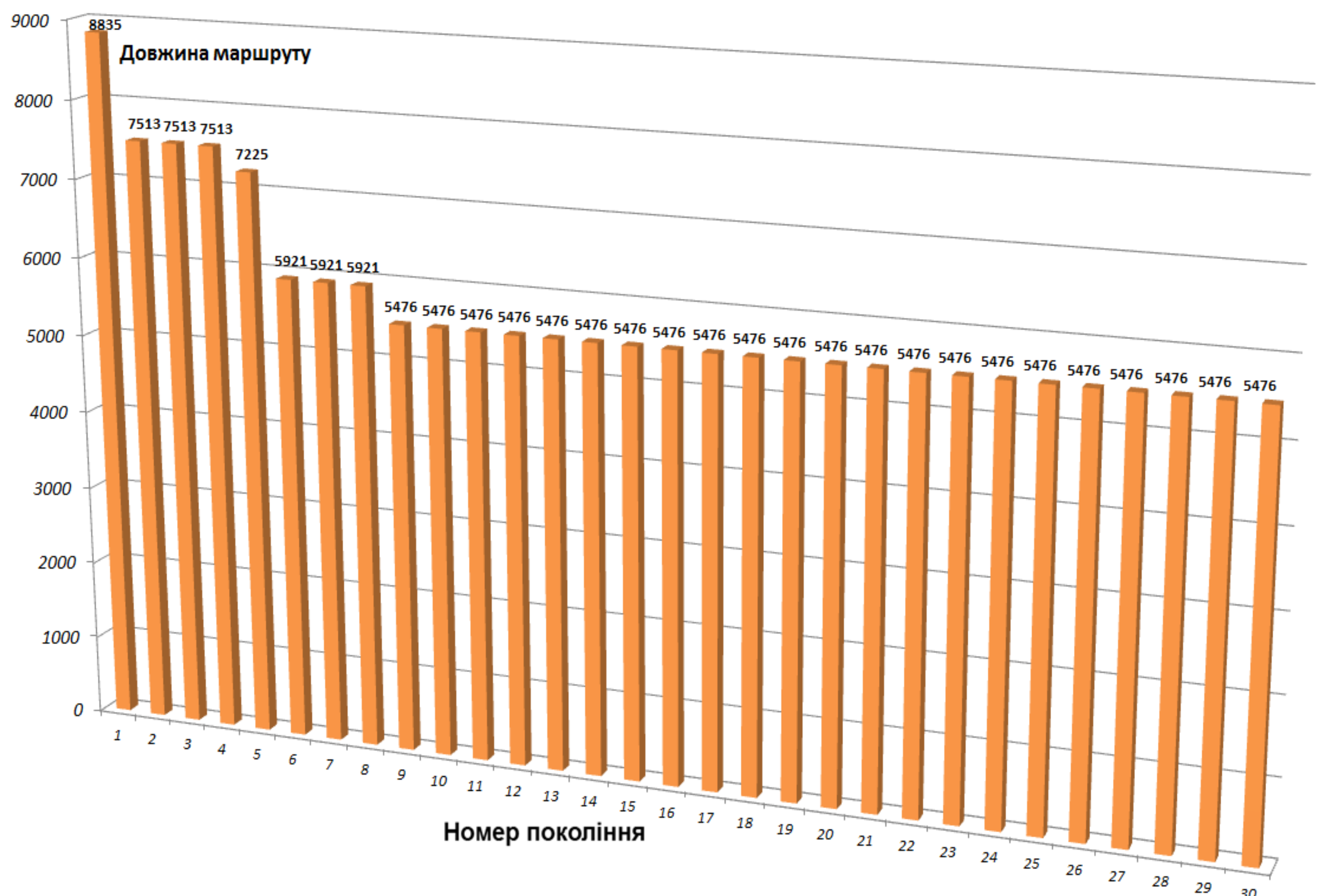


Рисунок 3.9 – Графік залежності найкоротшої відстані від номеру покоління

Як можна побачити з поточного графіку на рисунку 3.9 для 15 міст найкоротший маршрут вдалося знайти вже на 9 популяції, тож можливо було слід зупинити виконання алгоритму саме на ній, адже відбулося виродження популяцій.

3) Базові параметри запуску алгоритму:

- 30 поколінь – **25 міст**;
- рівень мутації = 0.25;
- кількість поколінь у маршруті = 8

```
new City( longitude: 2.1589, latitude: 41.3888, name: "Barcelona"),
new City( longitude: 12.5113, latitude: 41.8919, name: "Rome"),
new City( longitude: 2.3488, latitude: 48.8534, name: "Paris"),
new City( longitude: 16.9299, latitude: 52.4069, name: "Poznan"),
new City( longitude: 10.7460, latitude: 59.9127, name: "Oslo"),
new City( longitude: 13.0007, latitude: 55.6058, name: "Malme"),
new City( longitude: 25.2798, latitude: 54.6891, name: "Vilnius"),
new City( longitude: 14.4207, latitude: 50.0880, name: "Prague"),
new City( longitude: 18.0649, latitude: 59.3325, name: "Stockholm"),
new City( longitude: 4.8896, latitude: 52.3740, name: "Amsterdam")
```

Рисунок 3.10 - Скріншот доданих міст для виконання генетичного алгоритму

Скріншоти запуску програмного коду з результатами виконня програми для генетичного алгоритму:

```
> Generation #0
Route                                                                  |   Fitness | Distance(in miles)
-------------------------------------------------------------------------------------------------------
[Rome, Leipzig, Malme, Paris, Zhytomyr, Poltava, Odessa, Dnipro, Dortmund, Amsterdam, Ternopil, Sumy, Kharkiv, Donetsk, Munich, ]|0,676133|14790,000000
[Poltava, Amsterdam, Sumy, Donetsk, Vilnius, Odessa, Leverkusen, Stockholm, Dortmund, Malme, Poznan, Kyiv, Oslo, Leipzig, Prague]|0,601287|16631,000000
[Kharkiv, Mainz, Prague, Kyiv, Vilnius, Barcelona, Zhytomyr, Poltava, Sumy, Oslo, Ternopil, Poznan, Leipzig, Rome, Amsterdam, St]|0,593049|16862,000000
[Oslo, Sumy, Ternopil, Kyiv, Kharkiv, Munich, Leipzig, Mainz, Leverkusen, Rome, Vilnius, Odessa, Lviv, Donetsk, Prague, Malme, Dr]|0,581193|17206,000000
[Malme, Barcelona, Prague, Rome, Dnipro, Sumy, Odessa, Donetsk, Amsterdam, Leipzig, Munich, Lviv, Leverkusen, Zhytomyr, Ternopil]|0,563095|17759,000000
[Malme, Ternopil, Leipzig, Dnipro, Odessa, Poltava, Sumy, Leverkusen, Lviv, Rome, Amsterdam, Prague, Kyiv, Barcelona, Poznan, Zh]|0,549813|18188,000000
[Kharkiv, Poltava, Oslo, Ternopil, Dnipro, Poznan, Paris, Sumy, Vilnius, Amsterdam, Kyiv, Leipzig, Prague, Leverkusen, Stockholm]|0,527343|18963,000000
[Prague, Dnipro, Oslo, Paris, Leipzig, Kharkiv, Mainz, Poltava, Leverkusen, Rome, Lviv, Donetsk, Sumy, Malme, Kyiv, Ternopil, Do]|0,482649|20719,000000
```

Рисунок 3.11 - Скріншот запуску програми генетичного алгоритму

для 1 покоління

```
> Generation #29
Route                                                                                     |
-----------------------------------------------------------------------------------------------------------
[Rome, Leipzig, Paris, Malme, Vilnius, Kharkiv, Odessa, Lviv, Dortmund, Amsterdam, Mainz, Oslo, Prague, Stockholm, Poznan, Zhyt
[Barcelona, Leipzig, Leverkusen, Malme, Vilnius, Lviv, Odessa, Kharkiv, Dortmund, Amsterdam, Mainz, Oslo, Paris, Stockholm, Pol
[Rome, Mainz, Paris, Malme, Vilnius, Zhytomyr, Odessa, Lviv, Dortmund, Amsterdam, Leipzig, Munich, Prague, Stockholm, Poznan, K
[Rome, Leipzig, Ternopil, Dortmund, Malme, Kharkiv, Donetsk, Kyiv, Prague, Amsterdam, Mainz, Barcelona, Paris, Odessa, Poltava,
[Zhytomyr, Leipzig, Ternopil, Malme, Vilnius, Kharkiv, Kyiv, Barcelona, Dortmund, Amsterdam, Mainz, Oslo, Sumy, Odessa, Lviv, P
[Barcelona, Leipzig, Paris, Oslo, Poznan, Lviv, Sumy, Kharkiv, Vilnius, Stockholm, Dortmund, Malme, Rome, Ternopil, Zhytomyr, L
[Rome, Leipzig, Paris, Malme, Vilnius, Donetsk, Odessa, Amsterdam, Dortmund, Lviv, Mainz, Oslo, Barcelona, Poznan, Poltava, Kha
[Rome, Leipzig, Paris, Oslo, Vilnius, Kharkiv, Odessa, Lviv, Dortmund, Sumy, Poltava, Malme, Leverkusen, Donetsk, Amsterdam, Mu
```

```
Fitness | Distance(in miles)
----------------------------
]|0,783392|12765,000000
]|0,749794|13337,000000
]|0,704722|14190,000000
]|0,670601|14912,000000
]|0,666001|15015,000000
]|0,617971|16182,000000
]|0,604705|16537,000000
]|0,556235|17978,000000
```

Рисунок 3.12 - Скріншот запуску програми генетичного алгоритмудля 30 покоління

```
Найкращий маршрут знайдений до цих пір: [Rome, Leipzig, Paris, Malme, Vilnius, Kharkiv, Odessa, Lviv, Dortmund, Amsterdam, Mainz, Oslo, Prague, Stoc
 довжиною в : 12765,000000 миль
Час виконання: 360 мілісекунд(-и)
```

Рисунок 3.13 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

Найкращий маршрут знайдений до цих пір: [Rome, Leipzig, Paris, Malme, Vilnius, Kharkiv, Odessa, Lviv, Dortmund, Amsterdam, Mainz, Oslo, Prague, Stockholm, Poznan, Zhytomyr, Sumy, Poltava, Ternopil, Donetsk, Kyiv, Dnipro, Munich, Barcelona, Leverkusen] довжиною в 12765,000000 миль.

Як можна побачити з графіку на рисунку 3.14 для 25 міст сталася аналогічна ситуація зі знаходженням найкоротшого маршруту, але вже на 7 популяції, тож можливо було слід зупинити виконання алгоритму саме на ній, адже відбулося виродження популяцій.

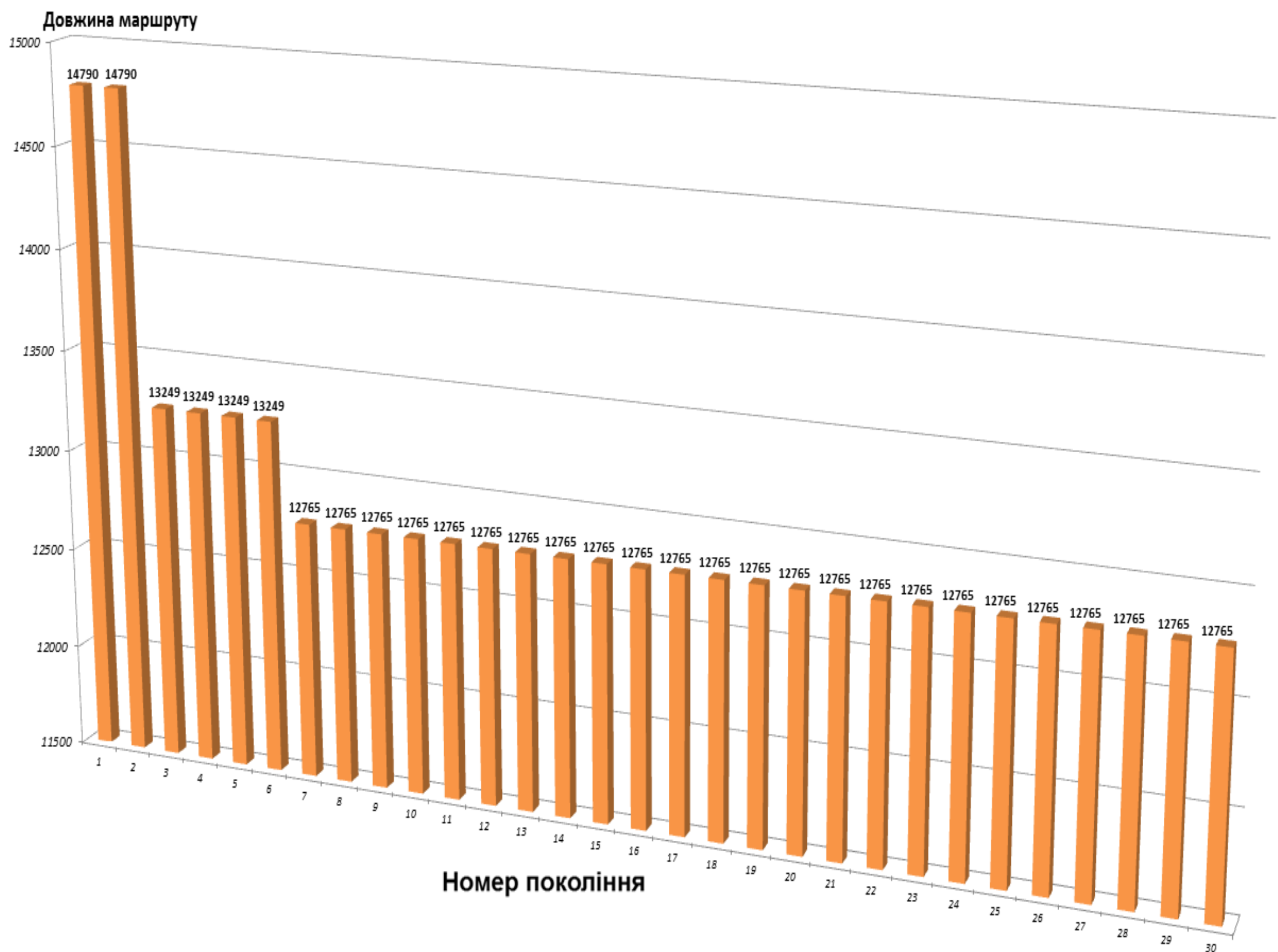


Рисунок 3.14 – Графік залежності найкоротшої відстані від номеру покоління

Внаслідок додавання міст до маршруту збільшилася його довжина та зменшилося значення функції пристосованості (на рисунку 3.15), внаслідок чого більше рішень з низьким значенням цього параметру могло бути відкинуто, що поліпшило швидкість знаходження найкоротшого маршруту та середню пристосованість всієї популяції.

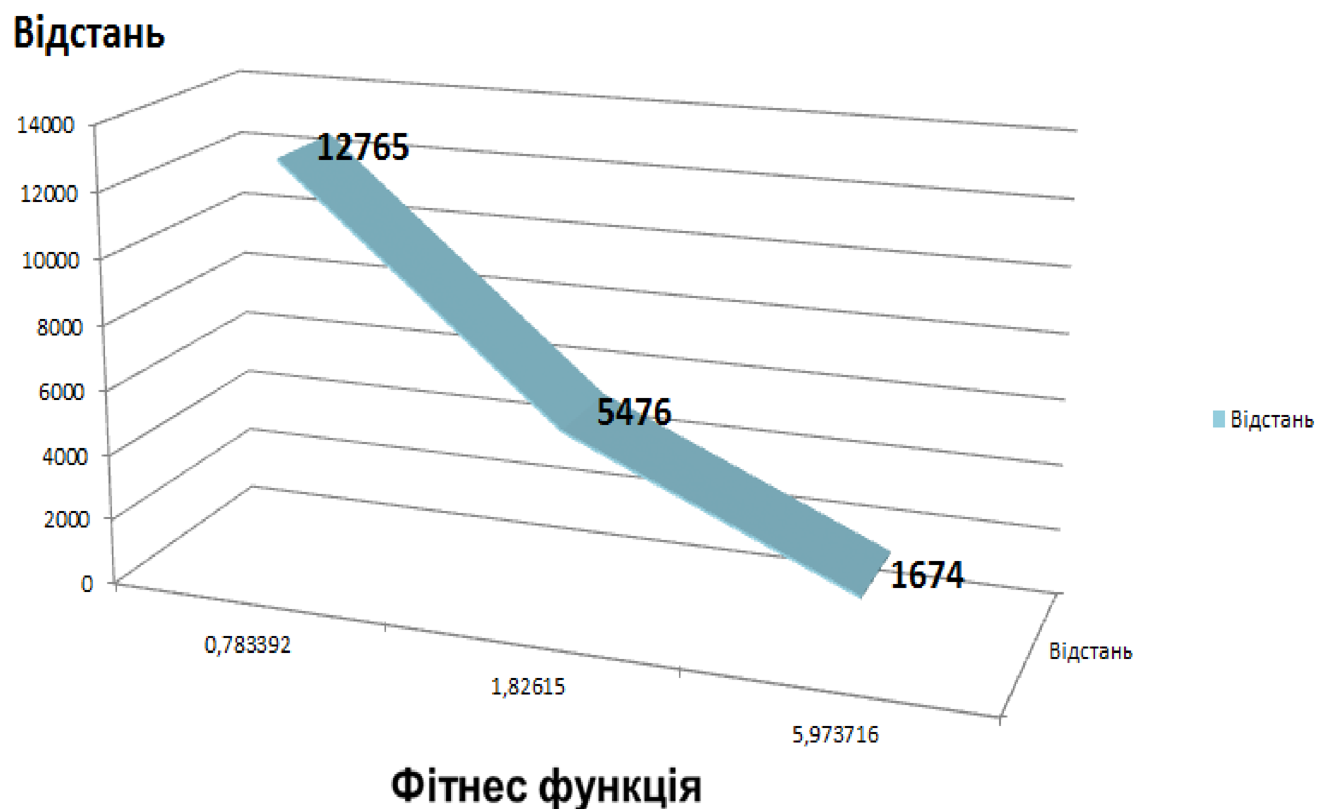


Рисунок 3.15 – Графік залежності найкоротшої відстані від значення функції пристосованості

3.6. Дослідження роботи генетичного алгоритму в залежності від рівня мутації та кількості поколінь

Тепер змінимо рівень мутації та поступово будемо збільшувати кількість поколінь від 30 до 45

1) Базові параметри запуску алгоритму:
   - 30 поколінь
   - **17 міст**;
   - рівень мутації = **0.4**;
   - кількість поколінь у маршруті = **8**.

Скріншоти запуску програмного коду з результатами виконання програми для генетичного алгоритму:

```
> Generation #0
Route                                                                                                    |                      Fitness | Distance(in miles)
-------------------------------------------------------------------------------------------------------------------------------------------------------------
[Barcelona, Paris, Ternopil, Lviv, Dnipro, Odessa, Sumy, Munich, Kyiv, Poltava, Dortmund, Zhytomyr, Donetsk, Kharkiv, Amsterdam, Rome, Stockholm]|0,807950|12377,000000
[Donetsk, Lviv, Kharkiv, Sumy, Ternopil, Dortmund, Stockholm, Paris, Dnipro, Zhytomyr, Odessa, Amsterdam, Munich, Barcelona, Kyiv, Rome, Poltava]|0,806257|12403,000000
[Zhytomyr, Paris, Dortmund, Lviv, Poltava, Odessa, Stockholm, Munich, Barcelona, Dnipro, Donetsk, Kyiv, Ternopil, Amsterdam, Sumy, Rome, Kharkiv]|0,776578|12877,000000
[Paris, Sumy, Dortmund, Barcelona, Stockholm, Amsterdam, Kharkiv, Lviv, Dnipro, Poltava, Donetsk, Ternopil, Munich, Odessa, Rome, Zhytomyr, Kyiv]|0,739098|13530,000000
[Paris, Ternopil, Stockholm, Poltava, Dortmund, Lviv, Zhytomyr, Dnipro, Amsterdam, Barcelona, Donetsk, Munich, Rome, Kyiv, Odessa, Kharkiv, Sumy]|0,718391|13920,000000
[Odessa, Stockholm, Sumy, Amsterdam, Rome, Dortmund, Kyiv, Paris, Ternopil, Munich, Zhytomyr, Poltava, Kharkiv, Dnipro, Lviv, Donetsk, Barcelona]|0,699301|14300,000000
[Kharkiv, Lviv, Stockholm, Kyiv, Barcelona, Rome, Zhytomyr, Munich, Dnipro, Dortmund, Poltava, Amsterdam, Ternopil, Donetsk, Sumy, Odessa, Paris]|0,644205|15523,000000
[Rome, Dnipro, Amsterdam, Donetsk, Zhytomyr, Munich, Sumy, Paris, Kharkiv, Lviv, Stockholm, Kyiv, Poltava, Barcelona, Dortmund, Odessa, Ternopil]|0,621157|16099,000000
```

Рисунок 3.16 - Скріншот запуску програми генетичного алгоритму для 1 покоління

```
> Generation #29
Route                                                                                                    |                      Fitness | Distance(in miles)
-------------------------------------------------------------------------------------------------------------------------------------------------------------
[Stockholm, Ternopil, Odessa, Poltava, Donetsk, Dnipro, Kharkiv, Zhytomyr, Rome, Kyiv, Sumy, Paris, Barcelona, Lviv, Munich, Dortmund, Amsterdam]|1,090988|9166,000000
[Lviv, Amsterdam, Paris, Dortmund, Barcelona, Sumy, Poltava, Odessa, Ternopil, Donetsk, Munich, Kyiv, Kharkiv, Zhytomyr, Rome, Dnipro, Stockholm]|0,861475|11608,000000
[Barcelona, Ternopil, Lviv, Poltava, Donetsk, Dnipro, Sumy, Amsterdam, Paris, Kharkiv, Zhytomyr, Munich, Odessa, Dortmund, Stockholm, Rome, Kyiv]|0,773455|12929,000000
[Amsterdam, Munich, Rome, Lviv, Stockholm, Odessa, Kharkiv, Poltava, Donetsk, Paris, Ternopil, Barcelona, Dnipro, Zhytomyr, Sumy, Dortmund, Kyiv]|0,744325|13435,000000
[Stockholm, Rome, Donetsk, Dortmund, Paris, Dnipro, Kharkiv, Poltava, Ternopil, Lviv, Munich, Sumy, Zhytomyr, Barcelona, Odessa, Amsterdam, Kyiv]|0,702593|14233,000000
[Kharkiv, Ternopil, Odessa, Paris, Lviv, Dnipro, Munich, Zhytomyr, Sumy, Barcelona, Rome, Kyiv, Amsterdam, Stockholm, Donetsk, Poltava, Dortmund]|0,687569|14544,000000
[Stockholm, Lviv, Poltava, Barcelona, Donetsk, Kyiv, Dortmund, Zhytomyr, Rome, Dnipro, Munich, Paris, Odessa, Ternopil, Sumy, Kharkiv, Amsterdam]|0,677094|14769,000000
[Zhytomyr, Dortmund, Odessa, Poltava, Munich, Sumy, Kharkiv, Amsterdam, Kyiv, Paris, Donetsk, Lviv, Rome, Ternopil, Barcelona, Dnipro, Stockholm]|0,595948|16780,000000
```

Рисунок 3.17 - Скріншот запуску програми генетичного алгоритму для 30 покоління

```
Найкращий маршрут знайдений до цих пір: [Stockholm, Ternopil, Odessa, Poltava, Donetsk, Dnipro, Kharkiv, Zhytomyr, Rome, Kyiv, Sumy, Paris, Barcelona, Lviv, Munich, Dortmund, Amsterdam]
 довжиною в : 9166,000000 миль
Час виконання: 312 мілісекунд(-и)
```

Рисунок 3.18 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

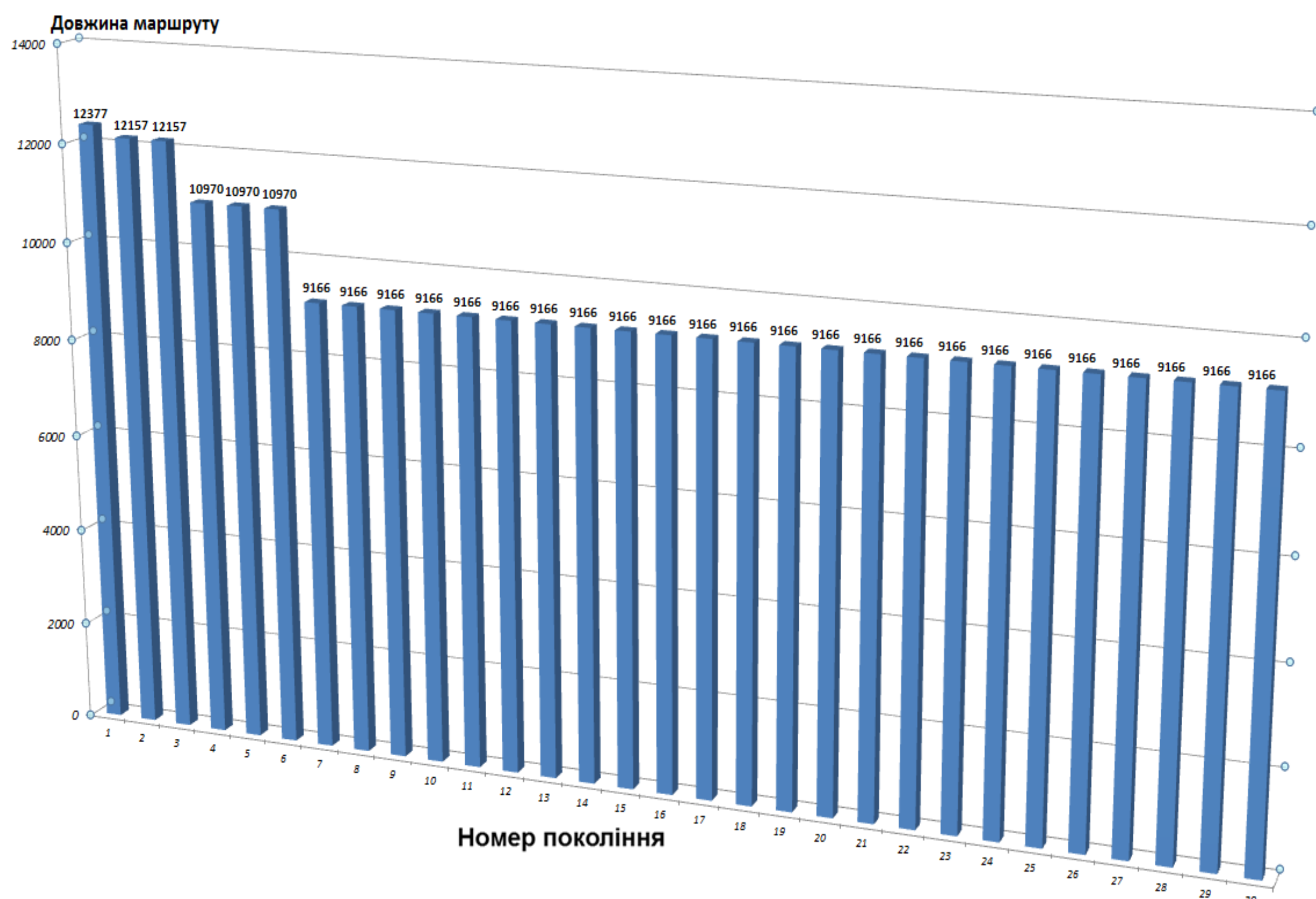


Рисунок 3.19 – Графік залежності найкоротшої відстані від номеру покоління

Як можна побачити з графіку на рисунку 3.19 найкоротший маршрут було знайдено вже у 7 поколінні, з подальшим виродженням популяцій.

2) Базові параметри запуску алгоритму:

- 35 поколінь
- **17 міст**;
- рівень мутації = **0.4**;
- кількість поколінь у маршруті = **8**.

Скріншоти запуску програмного коду з результатами виконня програми для генетичного алгоритму:

```
> Generation #0
Route                                                                                                   |                     Fitness | Distance(in miles)
---------------------------------------------------------------------------------------------------------------------------------------------------------
[Zhytomyr, Munich, Ternopil, Poltava, Stockholm, Amsterdam, Sumy, Dnipro, Kharkiv, Kyiv, Rome, Barcelona, Paris, Lviv, Dortmund, Donetsk, Odessa]|0,901469|11093,000000
[Paris, Rome, Stockholm, Ternopil, Sumy, Zhytomyr, Dortmund, Kharkiv, Kyiv, Amsterdam, Odessa, Poltava, Donetsk, Lviv, Dnipro, Barcelona, Munich]|0,801925|12470,000000
[Poltava, Zhytomyr, Stockholm, Munich, Paris, Lviv, Sumy, Odessa, Ternopil, Barcelona, Amsterdam, Kyiv, Rome, Dnipro, Kharkiv, Donetsk, Dortmund]|0,794281|12590,000000
[Rome, Donetsk, Odessa, Stockholm, Poltava, Ternopil, Dortmund, Kyiv, Munich, Amsterdam, Dnipro, Zhytomyr, Sumy, Lviv, Barcelona, Paris, Kharkiv]|0,713369|14018,000000
[Paris, Kyiv, Lviv, Ternopil, Poltava, Stockholm, Kharkiv, Barcelona, Amsterdam, Dnipro, Rome, Donetsk, Munich, Odessa, Sumy, Zhytomyr, Dortmund]|0,703185|14221,000000
[Poltava, Dnipro, Kharkiv, Rome, Amsterdam, Lviv, Ternopil, Munich, Zhytomyr, Kyiv, Dortmund, Odessa, Paris, Sumy, Barcelona, Donetsk, Stockholm]|0,661726|15112,000000
[Zhytomyr, Poltava, Paris, Rome, Dnipro, Amsterdam, Donetsk, Stockholm, Dortmund, Kharkiv, Munich, Kyiv, Barcelona, Lviv, Odessa, Ternopil, Sumy]|0,644122|15525,000000
[Lviv, Donetsk, Odessa, Dortmund, Poltava, Sumy, Barcelona, Dnipro, Amsterdam, Rome, Kyiv, Paris, Zhytomyr, Munich, Kharkiv, Stockholm, Ternopil]|0,623441|16040,000000
```

Рисунок 3.20 - Скріншот запуску програми генетичного алгоритму для 1 покоління

```
> Generation #5
Route                                                                                                   |                     Fitness | Distance(in miles)
---------------------------------------------------------------------------------------------------------------------------------------------------------
[Amsterdam, Rome, Barcelona, Stockholm, Paris, Dortmund, Munich, Kharkiv, Donetsk, Odessa, Ternopil, Lviv, Sumy, Poltava, Kyiv, Zhytomyr, Dnipro]|1,133530|8822,000000
[Zhytomyr, Rome, Barcelona, Poltava, Amsterdam, Dortmund, Munich, Kharkiv, Donetsk, Odessa, Ternopil, Kyiv, Paris, Stockholm, Sumy, Lviv, Dnipro]|0,874738|11432,000000
[Kyiv, Dortmund, Zhytomyr, Donetsk, Stockholm, Lviv, Dnipro, Odessa, Rome, Amsterdam, Ternopil, Kharkiv, Barcelona, Paris, Munich, Sumy, Poltava]|0,826378|12101,000000
[Amsterdam, Odessa, Barcelona, Stockholm, Paris, Dortmund, Poltava, Kharkiv, Donetsk, Rome, Ternopil, Lviv, Sumy, Munich, Kyiv, Zhytomyr, Dnipro]|0,762544|13114,000000
[Stockholm, Poltava, Zhytomyr, Munich, Amsterdam, Paris, Donetsk, Odessa, Sumy, Dnipro, Dortmund, Kharkiv, Barcelona, Ternopil, Kyiv, Lviv, Rome]|0,758783|13179,000000
[Dnipro, Paris, Odessa, Amsterdam, Barcelona, Lviv, Ternopil, Rome, Sumy, Kyiv, Dortmund, Zhytomyr, Stockholm, Munich, Donetsk, Poltava, Kharkiv]|0,745935|13406,000000
[Ternopil, Munich, Sumy, Dnipro, Paris, Poltava, Barcelona, Odessa, Amsterdam, Kharkiv, Dortmund, Rome, Stockholm, Lviv, Zhytomyr, Kyiv, Donetsk]|0,640738|15607,000000
[Munich, Zhytomyr, Stockholm, Poltava, Paris, Dnipro, Sumy, Odessa, Barcelona, Ternopil, Rome, Lviv, Kyiv, Amsterdam, Donetsk, Dortmund, Kharkiv]|0,592698|16872,000000
```

Рисунок 3.21 - Скріншот запуску програми генетичного алгоритму для 5 покоління

```
> Generation #34
Route                                                                                                   |                     Fitness | Distance(in miles)
---------------------------------------------------------------------------------------------------------------------------------------------------------
[Amsterdam, Rome, Barcelona, Stockholm, Paris, Dortmund, Munich, Kharkiv, Donetsk, Odessa, Ternopil, Lviv, Sumy, Poltava, Kyiv, Zhytomyr, Dnipro]|1,133530|8822,000000
[Rome, Paris, Barcelona, Dortmund, Amsterdam, Ternopil, Stockholm, Zhytomyr, Dnipro, Donetsk, Lviv, Sumy, Munich, Poltava, Kharkiv, Odessa, Kyiv]|1,009999|9901,000000
[Amsterdam, Rome, Odessa, Stockholm, Paris, Dortmund, Dnipro, Kharkiv, Donetsk, Sumy, Lviv, Ternopil, Kyiv, Poltava, Barcelona, Zhytomyr, Munich]|0,906865|11027,000000
[Amsterdam, Rome, Lviv, Dortmund, Paris, Barcelona, Sumy, Zhytomyr, Stockholm, Dnipro, Ternopil, Munich, Kyiv, Odessa, Poltava, Kharkiv, Donetsk]|0,904486|11056,000000
[Ternopil, Dnipro, Sumy, Kyiv, Poltava, Stockholm, Rome, Barcelona, Odessa, Lviv, Amsterdam, Kharkiv, Dortmund, Paris, Munich, Zhytomyr, Donetsk]|0,869490|11501,000000
[Ternopil, Barcelona, Poltava, Dnipro, Dortmund, Stockholm, Kharkiv, Lviv, Sumy, Zhytomyr, Amsterdam, Paris, Kyiv, Odessa, Munich, Rome, Donetsk]|0,756487|13219,000000
[Amsterdam, Donetsk, Odessa, Stockholm, Rome, Ternopil, Munich, Kharkiv, Paris, Barcelona, Lviv, Dortmund, Dnipro, Poltava, Kyiv, Zhytomyr, Sumy]|0,725900|13776,000000
[Amsterdam, Kharkiv, Barcelona, Stockholm, Paris, Poltava, Zhytomyr, Rome, Donetsk, Odessa, Kyiv, Lviv, Sumy, Dortmund, Ternopil, Dnipro, Munich]|0,678058|14748,000000
```

Рисунок 3.22 - Скріншот запуску програми генетичного алгоритму для 35 покоління

```
Найкращий маршрут знайдений до цих пір: [Amsterdam, Rome, Barcelona, Stockholm, Paris, Dortmund, Munich, Kharkiv, Donetsk, Odessa, Ternopil, Lviv, Sumy, Poltava, Kyiv, Zhytomyr, Dnipro]
 довжиною в : 8822,000000 миль
Час виконання: 453 мілісекунд(-и)
```

Рисунок 3.23 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

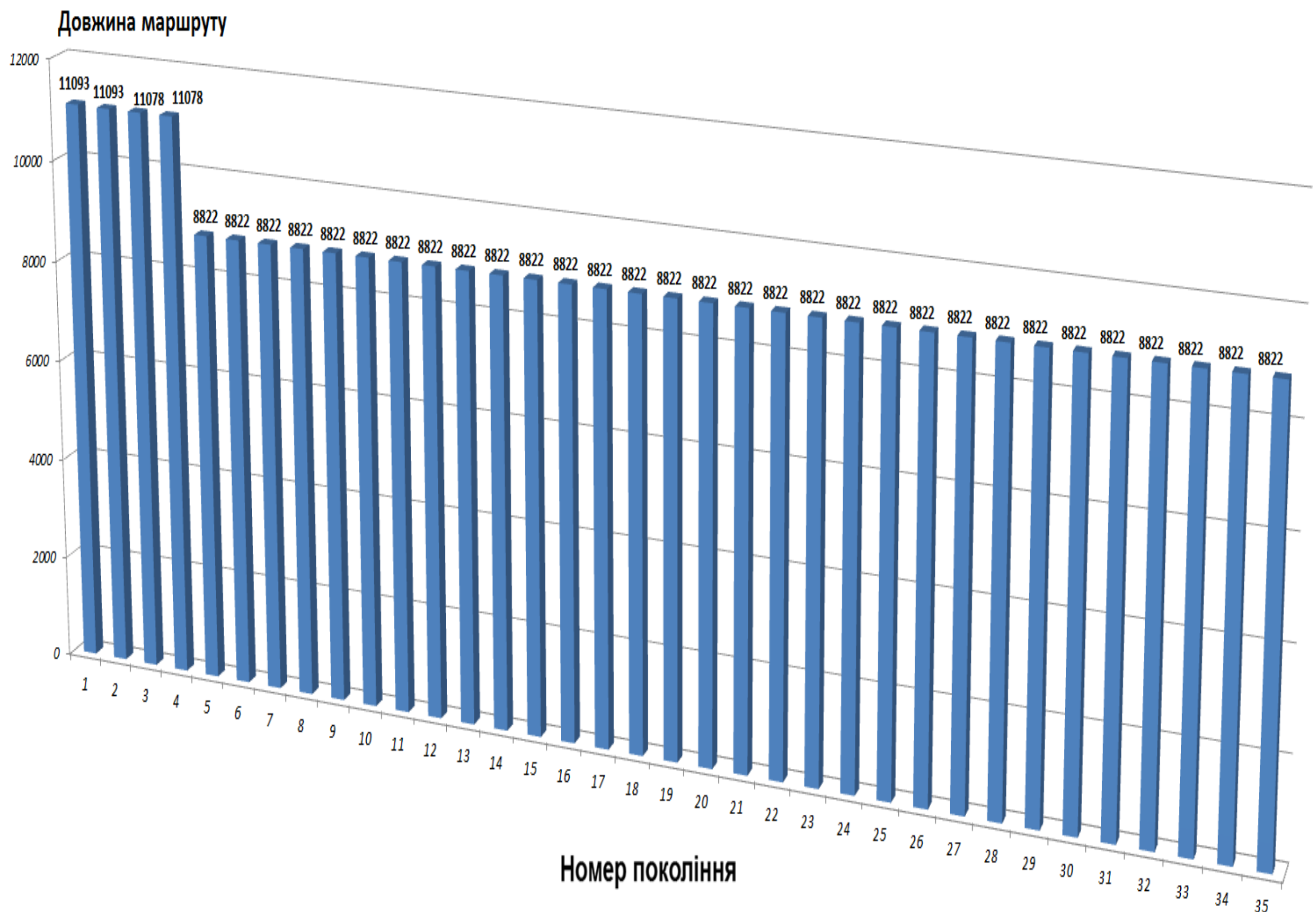


Рисунок 3.24 – Графік залежності найкоротшої відстані від номеру покоління

Як ми можемо побачити з графіку на рисунку 3.24, завдяки збільшенню кількості поколінь до 35, найкоротший маршрут вдалося знайти вже на 5 поколінні, тобто на цьому етапі можна було застосувати критерій зупину алгоритму.

3) Збільшимо кількість поколінь до 45 та рівень мутації до 0.5

Базові параметри запуску алгоритму:

- 45 поколінь
- **17 міст**;
- рівень мутації = **0.5**;
- кількість поколінь у маршруті = **8**.

Скріншоти запуску програмного коду з результатами виконання програми для генетичного алгоритму:

```
> Generation #0
Route                                                                                                    |                        Fitness | Distance(in miles)
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
[Dnipro, Kharkiv, Poltava, Ternopil, Dortmund, Munich, Kyiv, Amsterdam, Rome, Barcelona, Lviv, Paris, Donetsk, Odessa, Zhytomyr, Sumy, Stockholm]|0,861549|11607,000000
[Paris, Ternopil, Kharkiv, Zhytomyr, Barcelona, Amsterdam, Donetsk, Kyiv, Sumy, Dnipro, Odessa, Lviv, Dortmund, Stockholm, Munich, Rome, Poltava]|0,822030|12165,000000
[Kharkiv, Poltava, Lviv, Stockholm, Paris, Kyiv, Dnipro, Sumy, Barcelona, Rome, Odessa, Ternopil, Munich, Amsterdam, Donetsk, Zhytomyr, Dortmund]|0,797957|12532,000000
[Stockholm, Donetsk, Ternopil, Barcelona, Rome, Dortmund, Paris, Kharkiv, Munich, Dnipro, Poltava, Amsterdam, Odessa, Sumy, Lviv, Kyiv, Zhytomyr]|0,783822|12758,000000
[Ternopil, Kyiv, Rome, Amsterdam, Sumy, Kharkiv, Zhytomyr, Stockholm, Munich, Barcelona, Poltava, Paris, Donetsk, Dortmund, Lviv, Odessa, Dnipro]|0,720305|13883,000000
[Dnipro, Kharkiv, Sumy, Paris, Dortmund, Kyiv, Barcelona, Poltava, Donetsk, Stockholm, Ternopil, Lviv, Amsterdam, Zhytomyr, Rome, Odessa, Munich]|0,715154|13983,000000
[Amsterdam, Paris, Lviv, Kharkiv, Donetsk, Ternopil, Stockholm, Sumy, Dortmund, Zhytomyr, Munich, Poltava, Dnipro, Barcelona, Odessa, Rome, Kyiv]|0,696040|14367,000000
[Rome, Kharkiv, Paris, Zhytomyr, Lviv, Odessa, Ternopil, Poltava, Barcelona, Donetsk, Sumy, Stockholm, Dnipro, Amsterdam, Dortmund, Kyiv, Munich]|0,683667|14627,000000
```

Рисунок 3.25 - Скріншот запуску програми генетичного алгоритму для 1 покоління

```
> Generation #23
Route                                                                                                    |                        Fitness | Distance(in miles)
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
[Poltava, Zhytomyr, Kyiv, Donetsk, Dnipro, Sumy, Ternopil, Lviv, Rome, Amsterdam, Stockholm, Munich, Barcelona, Dortmund, Paris, Kharkiv, Odessa]|1,183152|8452,000000
[Poltava, Ternopil, Kyiv, Sumy, Dnipro, Kharkiv, Donetsk, Lviv, Dortmund, Rome, Amsterdam, Stockholm, Paris, Munich, Barcelona, Zhytomyr, Odessa]|1,129433|8854,000000
[Poltava, Donetsk, Kharkiv, Odessa, Rome, Sumy, Ternopil, Lviv, Dnipro, Dortmund, Barcelona, Munich, Kyiv, Paris, Amsterdam, Stockholm, Zhytomyr]|0,942329|10612,000000
[Odessa, Donetsk, Ternopil, Rome, Poltava, Lviv, Dortmund, Munich, Amsterdam, Paris, Kharkiv, Stockholm, Zhytomyr, Kyiv, Barcelona, Dnipro, Sumy]|0,825764|12110,000000
[Stockholm, Ternopil, Sumy, Amsterdam, Dnipro, Poltava, Kharkiv, Donetsk, Lviv, Dortmund, Zhytomyr, Odessa, Paris, Munich, Kyiv, Rome, Barcelona]|0,814133|12283,000000
[Poltava, Ternopil, Paris, Sumy, Zhytomyr, Amsterdam, Odessa, Donetsk, Kyiv, Dnipro, Lviv, Stockholm, Kharkiv, Dortmund, Barcelona, Rome, Munich]|0,795608|12569,000000
[Amsterdam, Rome, Kharkiv, Kyiv, Lviv, Paris, Sumy, Donetsk, Poltava, Ternopil, Barcelona, Dnipro, Zhytomyr, Munich, Dortmund, Odessa, Stockholm]|0,768049|13020,000000
[Poltava, Ternopil, Dortmund, Donetsk, Stockholm, Zhytomyr, Paris, Lviv, Munich, Kyiv, Amsterdam, Rome, Barcelona, Kharkiv, Odessa, Sumy, Dnipro]|0,758093|13191,000000
```

Рисунок 3.26 - Скріншот запуску програми генетичного алгоритму для 23 покоління

```
> Generation #44
Route                                                                                                    |                        Fitness | Distance(in miles)
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
[Poltava, Zhytomyr, Kyiv, Donetsk, Dnipro, Sumy, Ternopil, Lviv, Rome, Amsterdam, Stockholm, Munich, Barcelona, Dortmund, Paris, Kharkiv, Odessa]|1,183152|8452,000000
[Kharkiv, Munich, Odessa, Zhytomyr, Dnipro, Poltava, Kyiv, Lviv, Ternopil, Rome, Sumy, Dortmund, Barcelona, Amsterdam, Stockholm, Paris, Donetsk]|0,876040|11415,000000
[Lviv, Kyiv, Sumy, Ternopil, Barcelona, Rome, Odessa, Kharkiv, Munich, Stockholm, Amsterdam, Zhytomyr, Dnipro, Dortmund, Paris, Donetsk, Poltava]|0,848896|11780,000000
[Poltava, Lviv, Sumy, Stockholm, Kyiv, Donetsk, Dortmund, Amsterdam, Dnipro, Zhytomyr, Rome, Ternopil, Barcelona, Munich, Paris, Kharkiv, Odessa]|0,806192|12404,000000
[Rome, Sumy, Amsterdam, Kharkiv, Odessa, Kyiv, Lviv, Zhytomyr, Munich, Ternopil, Donetsk, Dnipro, Dortmund, Stockholm, Poltava, Paris, Barcelona]|0,801989|12469,000000
[Munich, Amsterdam, Stockholm, Barcelona, Ternopil, Paris, Dnipro, Dortmund, Lviv, Odessa, Rome, Zhytomyr, Poltava, Donetsk, Kharkiv, Kyiv, Sumy]|0,787278|12702,000000
[Lviv, Poltava, Paris, Barcelona, Ternopil, Rome, Zhytomyr, Dortmund, Sumy, Munich, Kyiv, Dnipro, Odessa, Stockholm, Kharkiv, Donetsk, Amsterdam]|0,691324|14465,000000
[Stockholm, Kharkiv, Paris, Lviv, Zhytomyr, Munich, Rome, Donetsk, Amsterdam, Sumy, Poltava, Ternopil, Dortmund, Kyiv, Barcelona, Dnipro, Odessa]|0,638162|15670,000000
```

Рисунок 3.27 - Скріншот запуску програми генетичного алгоритму для 45 покоління

```
Найкращий маршрут знайдений до цих пір: [Poltava, Zhytomyr, Kyiv, Donetsk, Dnipro, Sumy, Ternopil, Lviv, Rome, Amsterdam, Stockholm, Munich, Barcelona, Dortmund, Paris, Kharkiv, Odessa]
 довжиною в : 8452,000000 миль
Час виконання: 531 мілісекунд(-и)
```

Рисунок 3.28 - Скріншот запуску програми генетичного алгоритму зі знайденим найкоротшим маршрутом

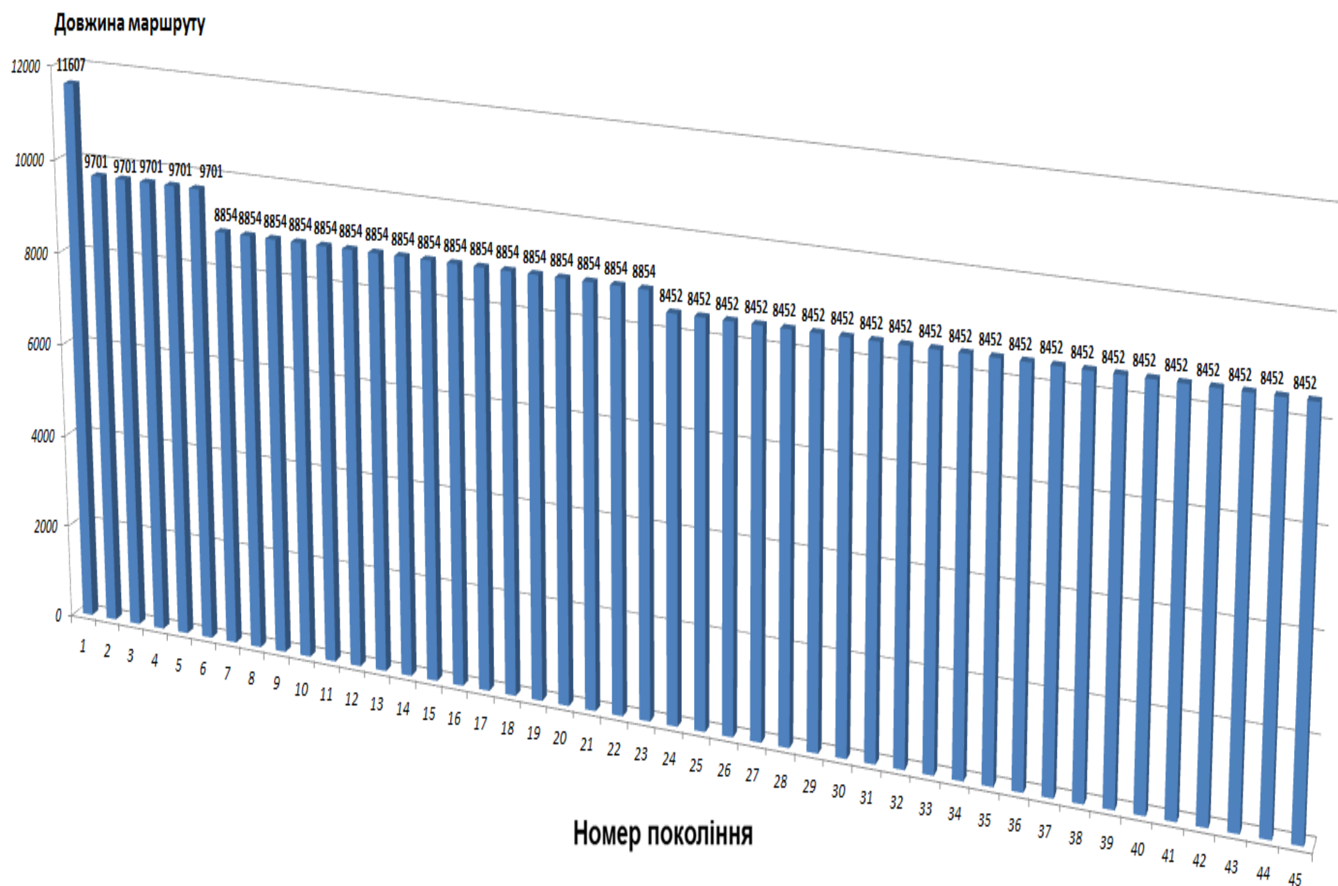


Рисунок 3.29 – Графік залежності найкоротшої відстані від номеру покоління

З даного графіку на рисунку 3.29 ми можемо спостерігати, що зі збільшенням кількості поколінь до 45 та збільшенням рівня мутації, вдалося зменшити проблему виродження популяцій та знайти маршрут, що коротший за знайдені при 30 та 35 поколіннях.

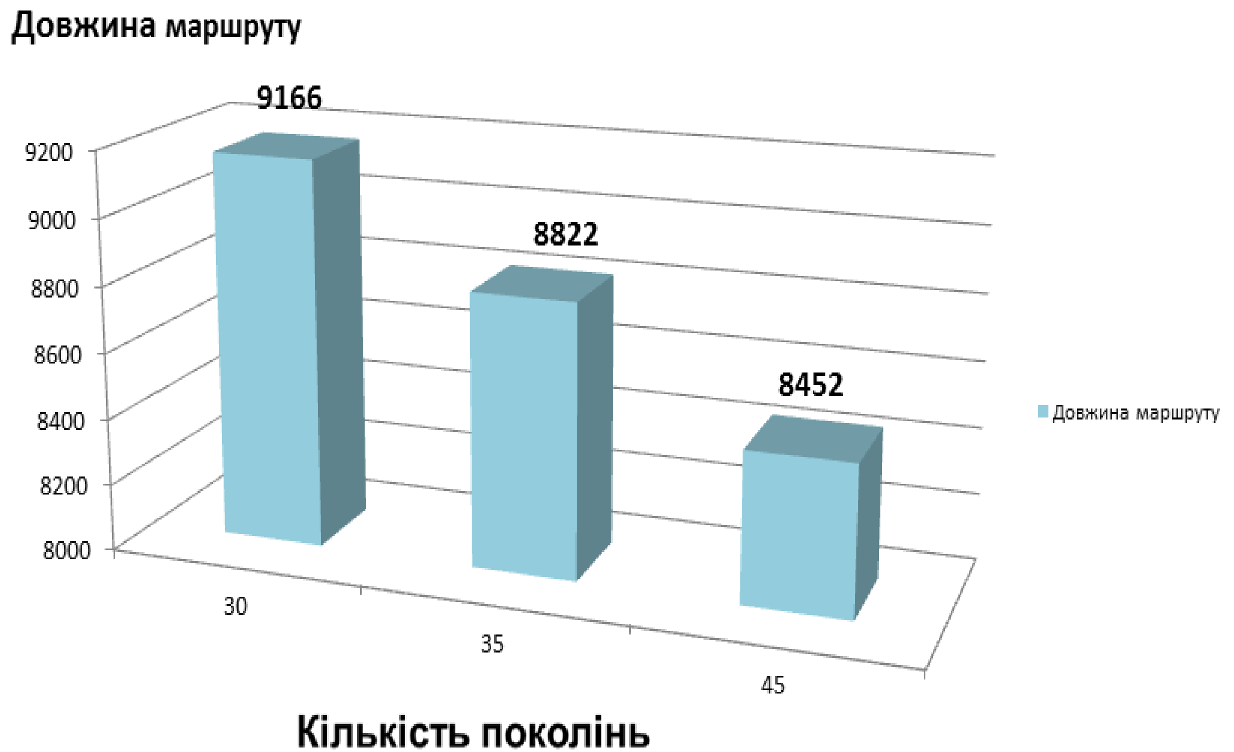


Рисунок 3.30 – Графік залежності найкоротшої відстані від кількості поколінь в алгоритмі

Висновки по генетичному алгоритму

1) Внаслідок додавання міст до маршруту збільшилася його довжина та зменшилося значення функції пристосованості, в результаті чого більше рішень з низьким значенням цього параметру могло бути відкинуто, що поліпшило швидкість знаходження найкоротшого маршруту та середню пристосованість всієї популяції.

2) Зі збільшенням кількості поколінь спостерігалася проблема виродження популяцій, коли слід було застосувати критерій зупину. Завдяки зростанню рівня мутації та кількості поколінь в алгоритмі до 45, вдалося зменшити поточну проблему виродження та знайти маршрут, що коротший за знайдені при 30 та 35 поколіннях.

3.7 Дослідження роботи алгоритму “мурахи” в залежності від кількості агентів

Дослідимо залежність якості рішення алгоритму “мурахи” від різноманітних параметрів.

Спочатку будемо змінювати кількість мурах **від 100 до 500**

1) Базові параметри запуску алгоритму:
   - 100 мурах
   - **17 міст**;
   - рівень випаровування феромону = **0.2**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 100 Artificial Ants ...
                                                    Route                                                                  |Distance(in miles) |      ant #
-------------------------------------------------------------------------------------------------------------------------------------------------------------
[Donetsk, Dnipro, Poltava, Kharkiv, Sumy, Kyiv, Zhytomyr, Ternopil, Lviv, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |      6257,57          |1
[Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Dnipro, Donetsk, Odessa, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |      5751,51          |3
[Amsterdam, Dortmund, Munich, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Stockholm] |      5724,02          |9
[Amsterdam, Dortmund, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |      5486,01          |40


Найкращий маршрут знайдений до цих пір довжиною  : 5486.010933707894 миль
```

Рисунок 3.31 - Скріншот запуску програми алгоритму “мурахи”

для 100 агентів зі знайденим найкоротшим маршрутом

Як можна побачити на рисунку 3.31 , вже 48 агенту зі 100 вдалося знайти найкоротший маршрут.

2) Базові параметри запуску алгоритму:

- 200 мурах
- **17 міст**;
- рівень випаровування феромону = **0.2**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 200 Artificial Ants ...
                                  Route                                                                                  |Distance(in miles) |    ant #
---------------------------------------------------------------------------------------------------------------------------------------------------------
[Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Kyiv, Zhytomyr, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |    5889,68    |1
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |    5636,43    |5
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Stockholm] |    5596,31    |45
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Donetsk, Dnipro, Odessa, Stockholm] |    5561,05    |70
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Stockholm] |    5481,60    |71
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |    5442,42    |143
```

Рисунок 3.32 - Скріншот запуску програми алгоритму “мурахи” для 200 агентів

```
Найкращий маршрут знайдений до цих пір довжиною  : 5442.424545426586 миль
Час виконання: 148 мілісекунд(-и)
```

Рисунок 3.33 - Скріншот запуску програми алгоритму “мурахи” зі знайденим найкоротшим маршрутом

Як можна побачити на рисунку 3.32, вже 143 агенту з 200 вдалося знайти найкоротший маршрут.

3) Базові параметри запуску алгоритму:

- 300 мурах
- **17 міст**;
- рівень випаровування феромону = **0.2**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 300 Artificial Ants ...
                                   Route                                                                  |Distance(in miles) |     ant #
-----------------------------------------------------------------------------------------------------------------------------------------------
[Ternopil, Lviv, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5905,68      |1
[Lviv, Ternopil, Zhytomyr, Kyiv, Odessa, Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Stockholm, Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich] |     5585,00      |2
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Lviv, Ternopil, Stockholm] |     5553,76      |143
[Stockholm, Amsterdam, Dortmund, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa] |     5437,64      |147


Найкращий маршрут знайдений до цих пір довжиною  : 5437.637301460103 миль
Час виконання: 172 мілісекунд(-и)
```

Рисунок 3.34 - Скріншот запуску програми алгоритму “мурахи” для 300 агентів зі знайденим найкоротшим маршрутом

Як можна побачити на рисунку 3.34 , вже 147 агенту з 300 вдалося знайти найкоротший маршрут.

4) Базові параметри запуску алгоритму:

- 400 мурах
- **17 міст**;
- рівень випаровування феромону = **0.2**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 400 Artificial Ants ...
                                   Route                                                                  |Distance(in miles) |     ant #
-----------------------------------------------------------------------------------------------------------------------------------------------
[Rome, Munich, Dortmund, Amsterdam, Paris, Barcelona, Odessa, Dnipro, Poltava, Kharkiv, Sumy, Kyiv, Zhytomyr, Ternopil, Lviv, Donetsk, Stockholm] |     7068,75      |1
[Kharkiv, Poltava, Dnipro, Donetsk, Sumy, Kyiv, Zhytomyr, Lviv, Ternopil, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     6254,00      |3
[Paris, Amsterdam, Dortmund, Munich, Rome, Barcelona, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Stockholm] |     6080,27      |2
[Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Zhytomyr, Kyiv, Ternopil, Lviv, Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Stockholm] |     5857,22      |13
[Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Dnipro, Donetsk, Odessa, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5743,25      |29
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |     5644,68      |33
[Ternopil, Lviv, Zhytomyr, Kyiv, Poltava, Dnipro, Donetsk, Kharkiv, Sumy, Odessa, Rome, Barcelona, Paris, Amsterdam, Dortmund, Munich, Stockholm] |     5625,47      |84
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |     5434,17      |159
[Sumy, Poltava, Kharkiv, Dnipro, Donetsk, Odessa, Kyiv, Zhytomyr, Ternopil, Lviv, Munich, Rome, Barcelona, Paris, Amsterdam, Dortmund, Stockholm] |     5365,63      |209
```

Рисунок 3.35 - Скріншот запуску програми алгоритму “мурахи” для 400 агентів

```
Найкращий маршрут знайдений до цих пір довжиною  : 5365.633018642673 миль
Час виконання: 248 мілісекунд(-и)
```

Рисунок 3.36 - Скріншот запуску програми алгоритму “мурахи” зі знайденим найкоротшим маршрутом

Як можна побачити на рисунку 3.35 , вже 289 агенту з 400 вдалося знайти найкоротший маршрут.

5) Базові параметри запуску алгоритму:

- 500 мурах
- **17 міст**;
- рівень випаровування феромону = **0.2**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 500 Artificial Ants ...
                                  Route                                                                                  |Distance(in miles) |    ant #
---------------------------------------------------------------------------------------------------------------------------------------------------------
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Ternopil, Lviv, Zhytomyr, Kyiv, Dnipro, Poltava, Kharkiv, Sumy, Donetsk, Odessa, Stockholm] |    5865,78    |4
[Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Donetsk, Dnipro, Odessa, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |    5667,88    |9
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Odessa, Dnipro, Poltava, Sumy, Kharkiv, Donetsk, Kyiv, Zhytomyr, Ternopil, Lviv, Stockholm] |    5542,39    |36
[Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Ternopil, Lviv, Stockholm, Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich] |    5443,57    |63
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Donetsk, Dnipro, Odessa, Stockholm] |    5367,05    |185
[Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Ternopil, Lviv, Munich, Rome, Barcelona, Paris, Amsterdam, Dortmund, Stockholm] |    5291,52    |302
```

Рисунок 3.37 - Скріншот запуску програми алгоритму “мурахи” для 500 агентів

```
Найкращий маршрут знайдений до цих пір довжиною  : 5291.520801877303 миль
Час виконання: 281 мілісекунд(-и)
```

Рисунок 3.38 - Скріншот запуску програми алгоритму “мурахи” зі знайденим найкоротшим маршрутом

Як можна побачити на рисунку 3.37 , вже 382 агенту з 500 вдалося знайти найкоротший маршрут.

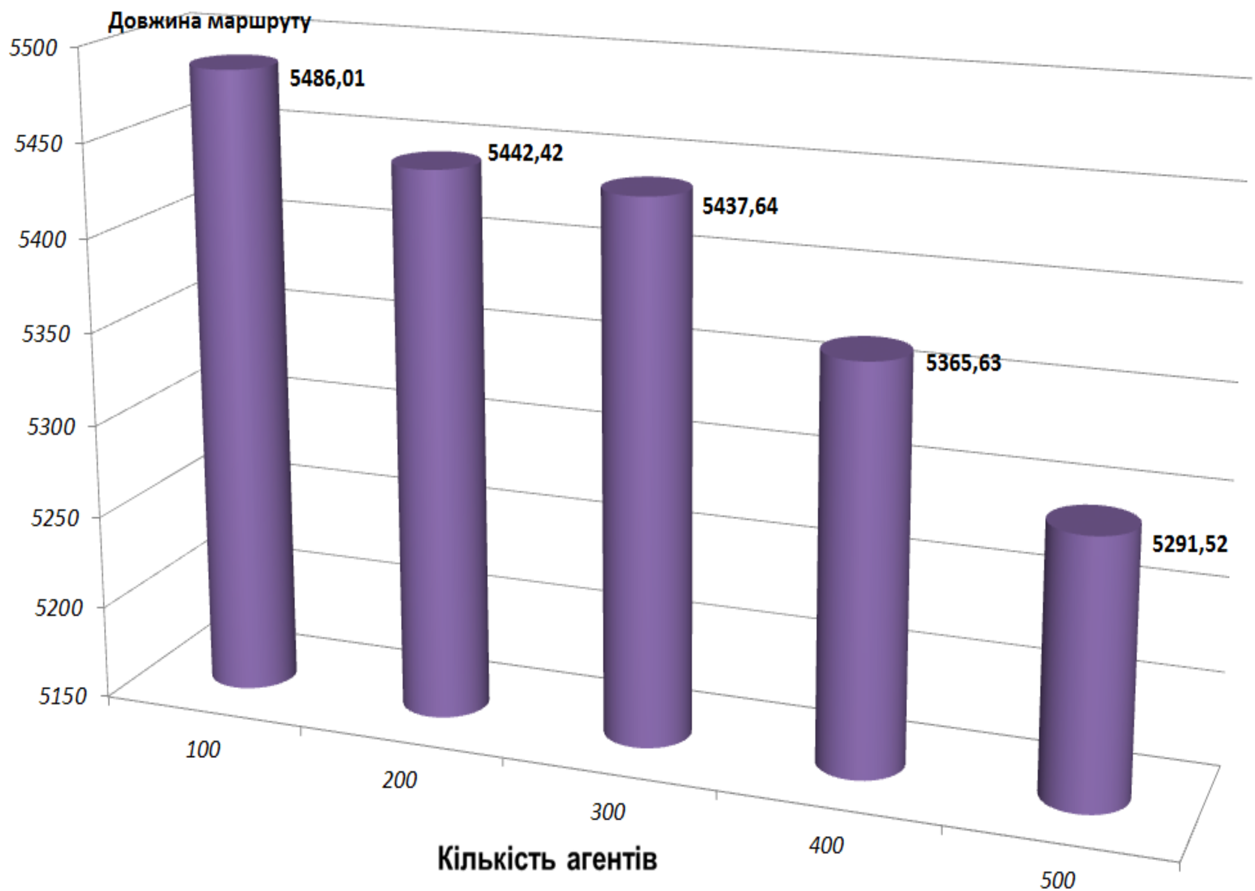


Рисунок 3.39 – Графік залежності найкоротшої відстані від кількості агентів в алгоритмі

З даного графіку на рисунку 3.39 ми можемо спостерігати, що зі збільшенням кількості мурах до 500 довжина маршруту поступово зменшувалася, та вдалося знайти маршрут, що коротший за знайдені при кількості агентів від 100 до 400.

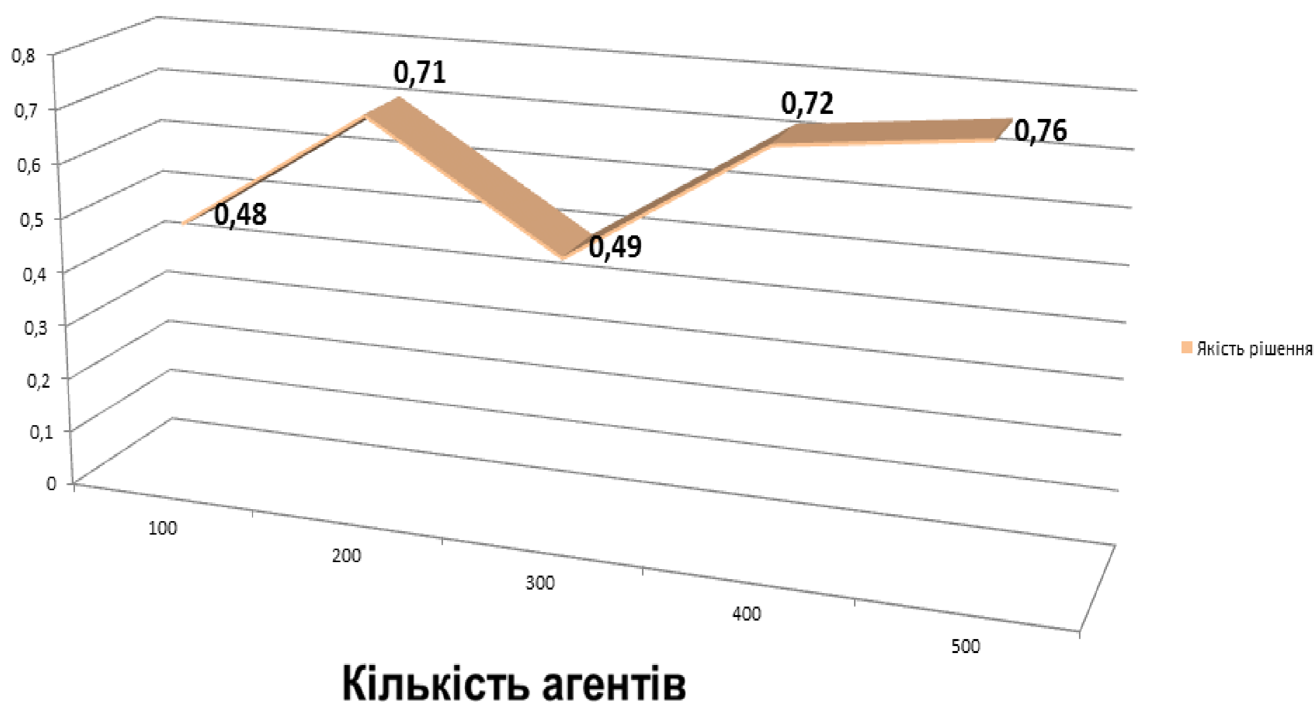


Рисунок 3.40 – Графік залежності якості рішення від кількості мурах в алгоритмі

З даного графіку на рисунку 3.40 ми можемо спостерігати, що зі збільшенням кількості мурах до 500, в основному якість рішення алгоритму покращується (номер мурахи, що знайшла найкоротший маршрут першою, ділимо на загальну кількість мурах в поточній ітерації), а довжина знайденого маршруту оптимізується (зменшується). У випадку 300 агентів, якість рішення не відчутно вплинула на результат, адже довжина маршруту була коротшою, ніж у випадку з 200 агентами.

3.8 Дослідження роботи алгоритму "мурахи" в залежності від рівня випаровування феромону

Посилаючись на рисунок 3.31, можна помітити, що при кількості агентів – 100 та коефіцієнту випаровування феромону = **0,2** найкоротша відстань була знайдена довжиною в **5486,81 милю**.

Тепер будемо змінювати рівень випаровування феромону від 0.3 до 0.7

1) Базові параметри запуску алгоритму:
    - 100 мурах
    - **17 міст**;
    - рівень випаровування феромону = **0.3**;

Скріншоти запуску програмного коду з результатами виконння програми для алгоритму "мурахи":

```
> 100 Artificial Ants ...
                                  Route                                                                   |Distance(in miles) |     ant #
--------------------------------------------------------------------------------------------------------------------------------------------
[Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5800,31          |1
[Stockholm, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome] |     5800,31          |23
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Stockholm] |     5683,86          |44
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Stockholm] |     5481,60          |45


Найкращий маршрут знайдений до цих пір довжиною  : 5481.604279935056 миль
Час виконання: 342 мілісекунд(-и)
```

Рисунок 3.41 - Скріншот запуску програми алгоритму "мурахи" для 100 агентів з рівнем випаровування феромону = 0.3

2) Базові параметри запуску алгоритму:

- 100 мурах
- **17 міст**;
- рівень випаровування феромону = **0.45**;

Скріншоти запуску програмного коду з результатами виконання програми для алгоритму “мурахи”:

```
> 100 Artificial Ants ...
                                        Route                                                    |Distance(in miles) |      ant #
-----------------------------------------------------------------------------------------------------------------------------------
[Dnipro, Poltava, Sumy, Kharkiv, Donetsk, Odessa, Zhytomyr, Kyiv, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5991,55      |1
[Poltava, Kharkiv, Sumy, Dnipro, Donetsk, Odessa, Zhytomyr, Kyiv, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5974,82      |6
[Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Poltava, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm, Dortmund, Amsterdam, Paris, Munich, Rome, Barcelona] |     5969,13      |8
[Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Kyiv, Zhytomyr, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |     5737,13      |10
[Barcelona, Munich, Paris, Amsterdam, Dortmund, Stockholm, Ternopil, Lviv, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Rome] |     5676,77      |13
[Amsterdam, Dortmund, Munich, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Stockholm] |     5636,47      |38
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Ternopil, Lviv, Zhytomyr, Kyiv, Poltava, Sumy, Kharkiv, Dnipro, Donetsk, Odessa, Stockholm] |     5574,60      |39
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Stockholm] |     5394,05      |81
```

Рисунок 3.42 - Скріншот запуску програми алгоритму “мурахи” для 100 агентів з рівнем випаровування феромону = 0.45

```
Найкращий маршрут знайдений до цих пір довжиною  : 5394.050913178793 миль
Час виконання: 294 мілісекунд(-и)
```

Рисунок 3.43 - Скріншот запуску програми алгоритму “мурахи” зі знайденим найкоротшим маршрутом

3) Базові параметри запуску алгоритму:

- 100 мурах
- **17 міст**;
- рівень випаровування феромону = **0.7**;

Скріншоти запуску програмного коду з результатами виконня програми для алгоритму “мурахи”:

```
> 100 Artificial Ants ...
                                          Route                                                                  |Distance(in miles) |       ant #
----------------------------------------------------------------------------------------------------------------------------------------------------
[Ternopil, Lviv, Zhytomyr, Kyiv, Poltava, Dnipro, Kharkiv, Sumy, Donetsk, Odessa, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |      6051,55         |1
[Kharkiv, Sumy, Poltava, Dnipro, Donetsk, Odessa, Zhytomyr, Kyiv, Ternopil, Lviv, Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Stockholm] |      5924,96         |2
[Munich, Dortmund, Amsterdam, Paris, Barcelona, Rome, Lviv, Ternopil, Zhytomyr, Kyiv, Poltava, Dnipro, Kharkiv, Sumy, Donetsk, Odessa, Stockholm] |      5742,18         |3
[Dortmund, Amsterdam, Paris, Barcelona, Rome, Munich, Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Stockholm] |      5394,05         |7
[Lviv, Ternopil, Zhytomyr, Kyiv, Sumy, Kharkiv, Poltava, Dnipro, Donetsk, Odessa, Munich, Rome, Barcelona, Paris, Amsterdam, Dortmund, Stockholm] |      5388,70         |24


Найкращий маршрут знайдений до цих пір довжиною  : 5388.701887819593 миль
```

Рисунок 3.44 - Скріншот запуску програми алгоритму “мурахи” для 100 агентів з рівнем випаровування феромону = 0.7

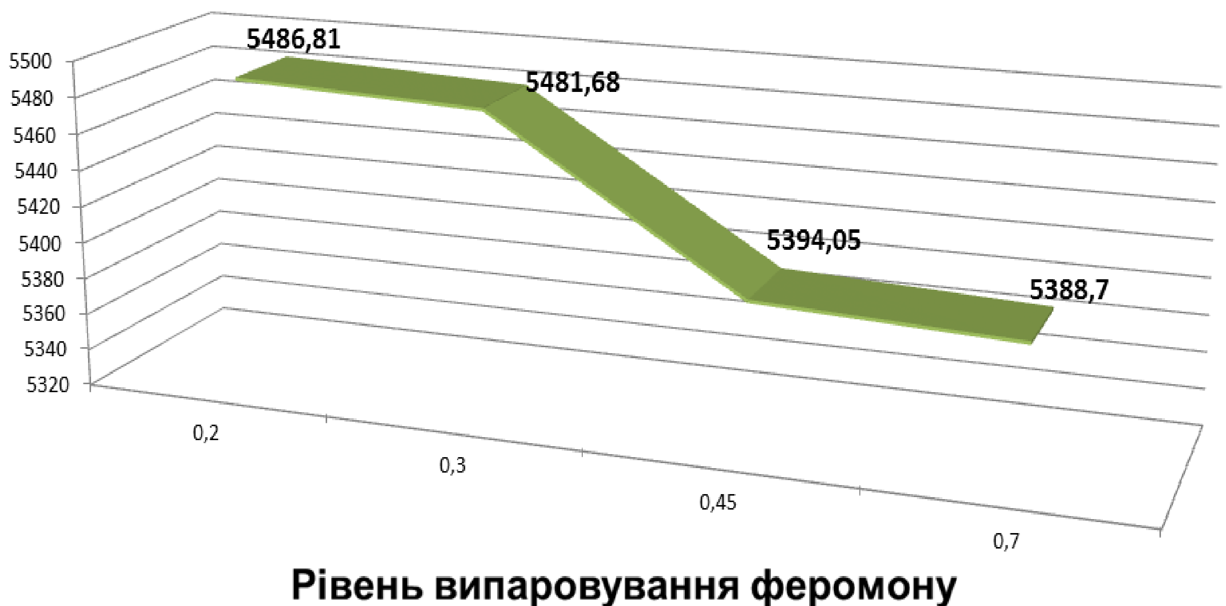


Рисунок 3.45 – Графік залежності найкоротшої відстані від рівня випаровування феромону

З даного графіку на рисунку 3.45 ми можемо спостерігати, що зі збільшенням рівня випаровування феромону, довжина маршруту поступово оптимізується (зменшується).

ВИСНОВКИ

В ході роботи були розглянуті відомі методи вирішення задачі комівояжера, такі як алгоритм Дейкстри, метод гілок і меж, генетичний алгоритм та алгоритм “мурахи”, проведено аналіз літератури. Був проведений порівняльний аналіз алгоритмів розв’язку задачі комівояжера.

Для генетичного алгоритму та алгоритму «мурахи» проведений порівняльний аналіз та отримані результати виконання програм алгоритмів, проведена апробація результатів, що відображено на скріншотах виконання програм та відповідних графіках залежності для обох алгоритмів. Мовою програмування було обрано - Java.

У генетичному алгоритмі внаслідок додавання міст до маршруту збільшилася його довжина та зменшилося значення функції пристосованості, внаслідок чого більше рішень з низьким значенням цього параметру могло бути відкинуто, що поліпшило швидкість знаходження найкоротшого маршруту та середню пристосованість всієї популяції.

Зі збільшенням кількості поколінь спостерігалася проблема виродження популяцій генетичного алгоритму, коли слід було застосувати критерій зупину. Завдяки зростанню рівня мутації та кількості поколінь в алгоритмі до 45, вдалося зменшити поточну проблему виродження та знайти маршрут, що коротший за знайдені при 30 та 35 поколіннях.

Зі збільшенням кількості мурах до 500, в основному якість рішення алгоритму покращується (номер мурахи, що знайшла найкоротший маршрут першою, ділимо на загальну кількість мурах в поточній ітерації), а довжина знайденого маршруту оптимізується (зменшується). У випадку 300 агентів, якість рішення не відчутно вплинула на результат , адже довжина маршруту була коротшою, ніж у випадку з 200 агентами.

Досліджуючи зміни випаровування феромону, вдалося встановити, що при збільшенні поточного параметру довжина маршруту оптимізується. Кількість ітерацій поступово зменшується внаслідок того, що більше мурах обирають найкоротший шлях, збільшуючи рівень виділення феромону. Таким чином, агенти знаходять маршрут швидше.

Отже, задача комівояжера є однією з найбільш досліджуваних задач комбінаторної оптимізації. Отримані результати можуть бути використані для подальшого дослідження та покращення якості роботи алгоритмів в залежності від комбінації різних параметрів для знаходження найкоротшого маршруту з меншими витратами ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

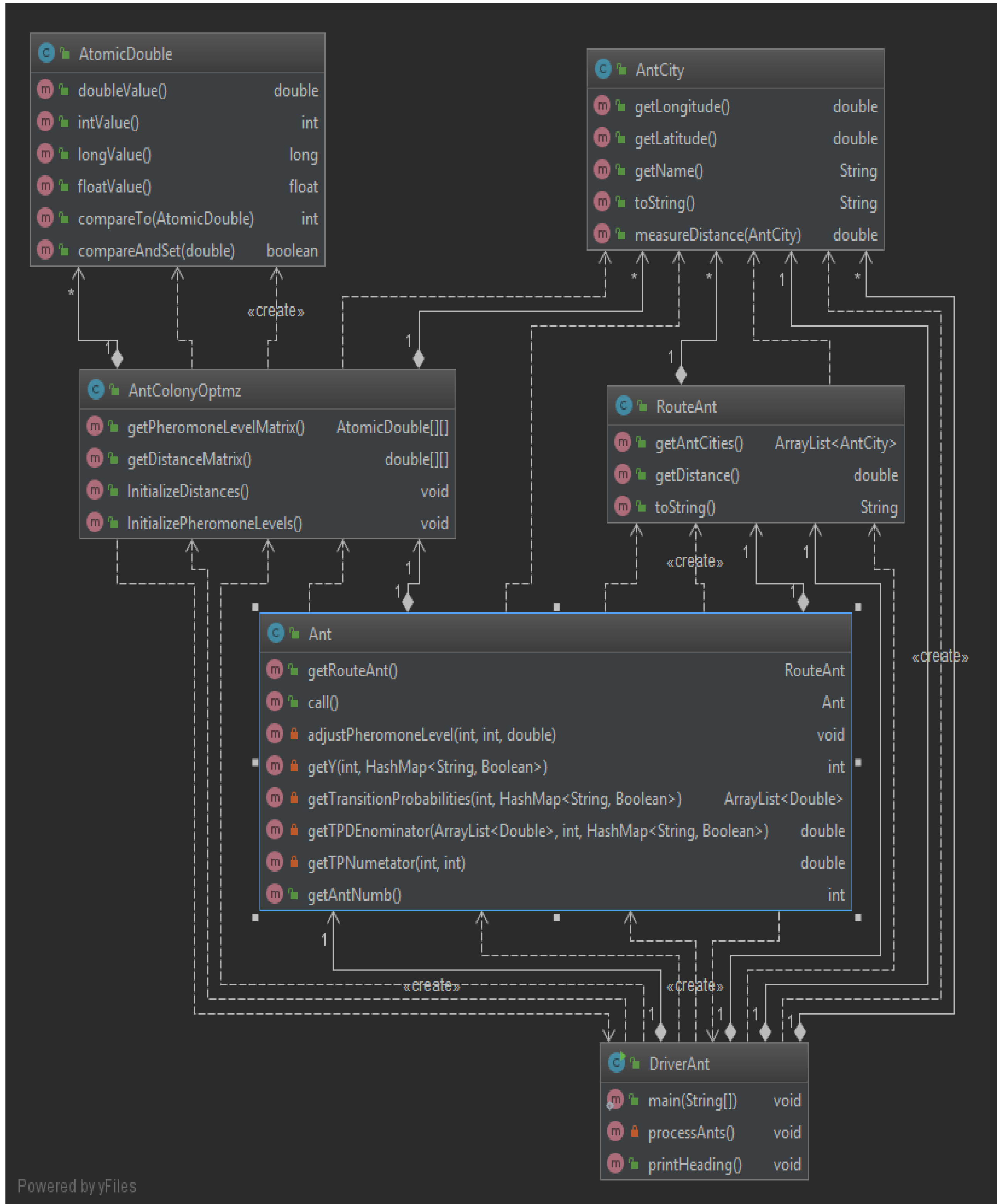
1. Біологічні основи мурашиних колоній – [Електронний ресурс]. URL: http://posibniki.com.ua/post-prikladni-sistemi-kolektivnogo-intelektu-swarm-intelligence (Дата звернення 15.10.2020).
2. Алгоритм мурахи - [Електронний ресурс]. URL: http://dn.khnu.km.ua/dn/k_default.aspx?M=k1113&T=14&lng=1&st=0 (Дата звернення 17.10.2020).
3. Основи проектування систем штучного інтелекту та розпізнавання образів: Методичні вказівки/ Скл.: В. В. Жуковський.- Рівне: НУВГП, 2016.- 53 с.
4. Володина Е.В., Студентова Е.А. Практическое применение алгоритма решения задачи коммивояжера.- М.: Инженерный вестник Дона, 2015 . - 250 с.
5. Korte B., Vygen J., Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics) 6th ed., New York, 2018,455 p.
6. Sathya N. , Muthukumaravel A. A review of the Optimization Algorithms on Traveling Salesman Problem. *Indian Journal of Science and Technology*, Vol 8(29), DOI: 10.17485/ijst/2015/v8i1/84652, November 2015
7. Denis M., Mujuni E., Kuznetsov D. Mathematical Formulation Model for a School Bus Routing Problem with Small Instance Data. *Journal Mathematical Theory and Modeling*. Vol.4, No.8, 2015.
8. Студентова Е.А. Алгоритм решения задачи коммивояжера с использованием Microsoft Excel и Open Office Calc // Современные проблемы науки и образования. 2015. №6. (приложение "Технические науки"). - С. 40.
9. Вільна енциклопедія Вікіпедія, стаття "Задача коммивояжера" – [Електронний ресурс]. – [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D0%B2

%D0%BE%D1%8F%D0%B6%D1%91%D1%80%D0%B0 (Дата звернення 01.11.2020).

10. Приближенные алгоритмы для NP-трудных задач : учеб.-метод. пособие / А. В. Кононов, П. А. Кононова ; Новосиб. гос. ун-т. — Новосибирск: РИЦ НГУ, 2015. — 117 с.
11. П’ятикоп, А.Г. Порівняльний аналіз генетичного алгоритму та алгоритму "Мурахи" для задачі комівояжера [Текст]: робота на здобуття кваліфікаційного ступеня бакалавра; спец.: 6.040302 - інформатика / А.Г. П’ятикоп; наук. керівник С.П. Шаповалов. - Суми: СумДУ, 2019. - 71 с.
12. Генетичні алгоритми. Ключові поняття та методи реалізації – [Електронний ресурс]. URL: http://www.znannya.org/?view=ga_general (Дата звернення 19.10.2020).
13. Вільна енциклопедія Вікіпедія, стаття “Муравьиный алгоритм” – [Електронний ресурс]. – [Електронний ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%80%D0%B0%D0%B2%D1%8C%D0%B8%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC#:~:text=%D0%9C%D1%83%D1%80%D0%B0%D0%B2%D1%8C%D0%B8%D0%BD%D1%8B%D0%B9%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20(%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%20%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8%20%D0%BF%D0%BE%D0%B4%D1%80%D0%B0%D0%B6%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC,%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%20%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0%20%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%BE%D0%B2%20%D0%BD%D0%B0%20%D0%B3%D1%80%D0%B0%D1%84%D0%B0%D1%85. (Дата звернення 03.11.2020).

ДОДАТОК І

Діаграма Java – класів для алгоритму “мурахи”



ДОДАТОК II

Діаграма Java – класів для генетичного алгоритму