**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни

«Проєктування алгоритмів»

„**Проєктування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-35 Брага Артем*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2024

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_Toc51260927)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою.

# Завдання

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розвʼязку (значення цільової функції) після кожних 20 ітерацій до 1000 (допускається самостійній вибір кроку та верхньої границі ітерацій) і побудувати графік залежності якості розвʼязку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Задача і алгоритм** |
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 8 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 9 | Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 12 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 31 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

import numpy as np  
from numba import njit  
import matplotlib.pyplot as plt  
import networkx as nx  
  
@njit  
def weighted\_random\_choice(probabilities):  
  
 cumulative\_sum = np.cumsum(probabilities)  
 random\_value = np.random.rand() \* cumulative\_sum[-1]  
 for i in range(len(cumulative\_sum)):  
 if random\_value <= cumulative\_sum[i]:  
 return i  
 return len(cumulative\_sum) - 1  
  
@njit  
def ant\_colony\_optimization(distance\_matrix, num\_ants, num\_elite\_ants, alpha, beta, rho, iterations, step):  
 num\_cities = len(distance\_matrix)  
 pheromone = np.ones((num\_cities, num\_cities))  
 best\_distance = np.inf  
 best\_path = None  
 quality\_over\_time = []  
  
 for iteration in range(iterations):  
 paths = np.zeros((num\_ants, num\_cities), dtype=np.int32)  
 distances = np.zeros(num\_ants)  
  
 for ant in range(num\_ants):  
 visited = np.zeros(num\_cities, dtype=np.bool\_)  
 current\_city = np.random.randint(0, num\_cities)  
 visited[current\_city] = True  
 path = [current\_city]  
  
 for \_ in range(num\_cities - 1):  
 probabilities = np.zeros(num\_cities)  
 for next\_city in range(num\_cities):  
 if not visited[next\_city]:  
 tau = pheromone[current\_city][next\_city] \*\* alpha  
 eta = (1.0 / distance\_matrix[current\_city][next\_city]) \*\* beta  
 probabilities[next\_city] = tau \* eta  
  
 probabilities\_sum = probabilities.sum()  
 if probabilities\_sum == 0:  
 probabilities += 1  
 probabilities\_sum = probabilities.sum()  
 probabilities /= probabilities\_sum  
  
 next\_city = weighted\_random\_choice(probabilities)  
 path.append(next\_city)  
 visited[next\_city] = True  
 current\_city = next\_city  
  
 paths[ant] = path  
 distance = 0  
 for i in range(num\_cities):  
 distance += distance\_matrix[path[i - 1], path[i]]  
 distances[ant] = distance  
  
 if distances[ant] < best\_distance:  
 best\_distance = distances[ant]  
 best\_path = path  
  
  
 pheromone \*= (1 - rho)  
 for ant in range(num\_ants):  
 contribution = 1.0 / distances[ant]  
 for i in range(num\_cities):  
 pheromone[paths[ant, i - 1], paths[ant, i]] += contribution  
  
 for elite in range(num\_elite\_ants):  
 contribution = 2.0 / distances[elite]  
 for i in range(num\_cities):  
 pheromone[paths[elite, i - 1], paths[elite, i]] += contribution  
  
 if iteration % step == 0:  
 quality\_over\_time.append((iteration, best\_distance))  
  
 return best\_distance, best\_path, quality\_over\_time  
  
def generate\_distance\_matrix(num\_cities, min\_distance=1, max\_distance=60):  
  
 distance\_matrix = np.random.randint(min\_distance, max\_distance + 1, size=(num\_cities, num\_cities))  
 np.fill\_diagonal(distance\_matrix, 0)  
 return (distance\_matrix + distance\_matrix.T) // 2  
  
def plot\_quality(quality\_over\_time):  
  
 iterations, distances = zip(\*quality\_over\_time)  
 plt.plot(iterations, distances, marker='o')  
 plt.xlabel("Ітерації")  
 plt.ylabel("Якість розв'язку (довжина шляху)")  
 plt.title("Залежність якості розв'язку від числа ітерацій")  
 plt.grid()  
 plt.show()  
  
def plot\_graph(distance\_matrix, best\_path=None):  
  
 num\_cities = len(distance\_matrix)  
 G = nx.Graph()  
  
  
 for i in range(num\_cities):  
 G.add\_node(i, pos=(np.cos(2 \* np.pi \* i / num\_cities), np.sin(2 \* np.pi \* i / num\_cities)))  
  
  
 for i in range(num\_cities):  
 for j in range(i + 1, num\_cities):  
 G.add\_edge(i, j, weight=distance\_matrix[i, j])  
  
  
 pos = nx.get\_node\_attributes(G, 'pos')  
  
  
 plt.figure(figsize=(10, 10))  
 nx.draw\_networkx\_nodes(G, pos, node\_size=50, node\_color='blue', alpha=0.7)  
 nx.draw\_networkx\_edges(G, pos, alpha=0.5, edge\_color='gray')  
  
  
 if best\_path:  
 path\_edges = [(best\_path[i - 1], best\_path[i]) for i in range(len(best\_path))]  
 nx.draw\_networkx\_edges(G, pos, edgelist=path\_edges, edge\_color='red', width=2)  
  
 plt.title("Граф міст на основі матриці відстаней")  
 plt.axis('off')  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 num\_cities = 300  
 num\_ants = 45  
 num\_elite\_ants = 15  
 alpha = 3  
 beta = 2  
 rho = 0.6  
 iterations = 1000  
 step = 20  
  
 distance\_matrix = generate\_distance\_matrix(num\_cities)  
  
 best\_distance, best\_path, quality\_over\_time = ant\_colony\_optimization(  
 distance\_matrix, num\_ants, num\_elite\_ants, alpha, beta, rho, iterations, step  
 )  
  
 print(f"Найкраща довжина шляху: {best\_distance}")  
 print(f"Найкращий шлях: {best\_path}")  
  
 plot\_quality(quality\_over\_time)  
 plot\_graph(distance\_matrix, best\_path)

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

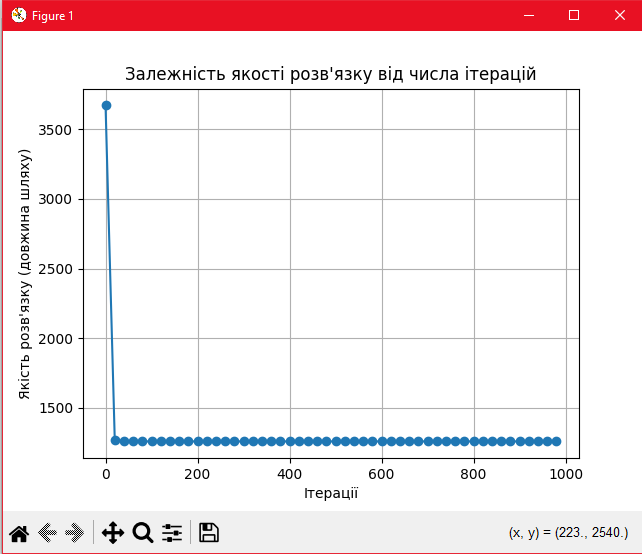
Рисунок 3.1 – Графік знаходження якості розв’язку від числа ітерацій  


Рисунок 3.2 – Граф міст, побудований на основі матриці відстаней

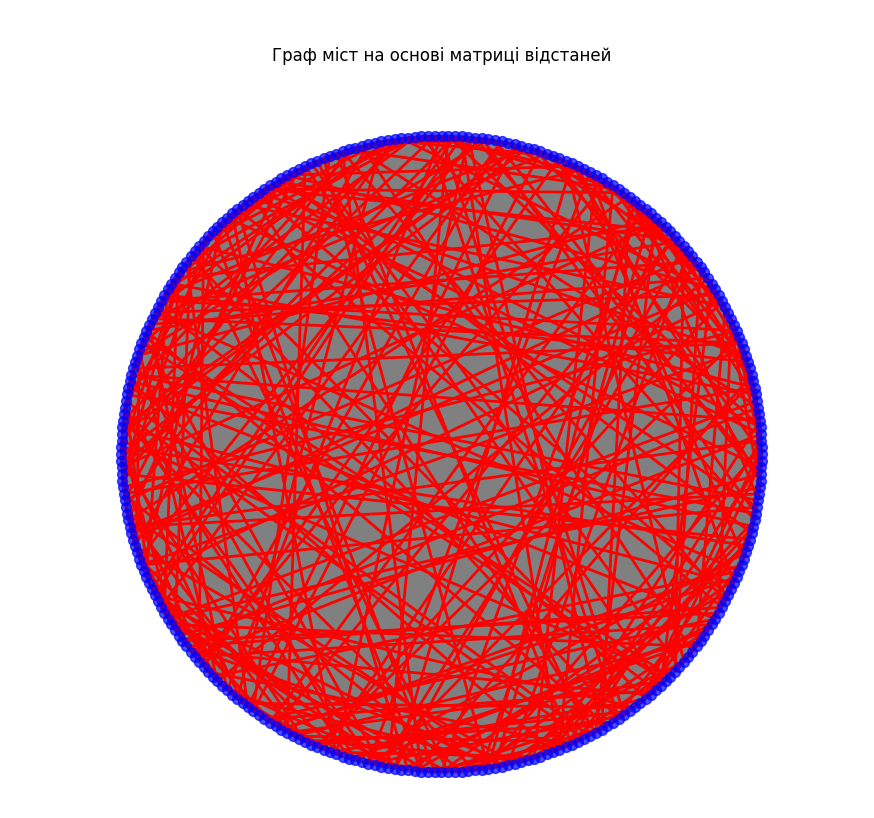
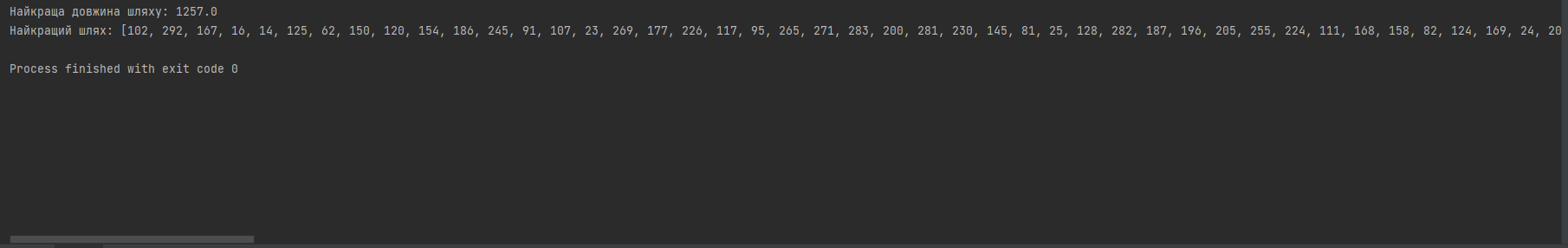


Рисунок 3.3 - Знаходження найкращого шляху



## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

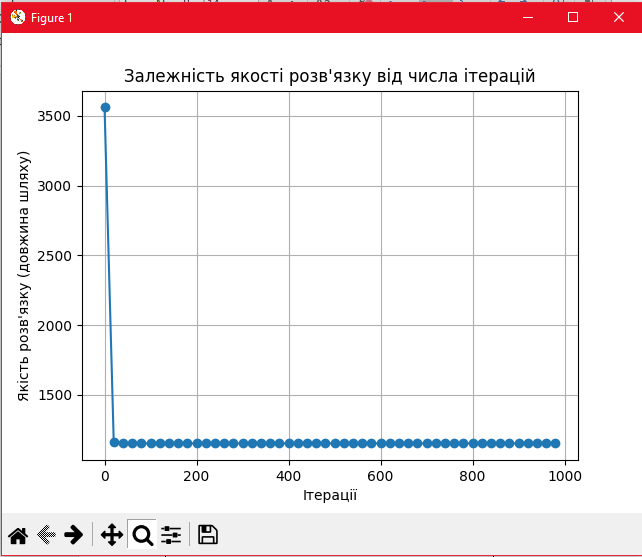
У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

|  |  |
| --- | --- |
| Ітерації | Якість рішення( Довжина шляху) |
| 1 | 3557 |
| 20 | 1158 |
| 40 | 1155 |
| 60 | 1150 |
| 80 | 1150 |
| … | … |
| 1000 | 1150 |

### Графіки залежності розвʼязку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розвʼязку.

Рисунок 3.3 – Графіки залежності розвʼязку від числа ітерацій





Висновок

В рамках даної лабораторної роботи було реалізовано алгоритм мурашиної колонії для вирішення задачі комівояжера. Результати експериментів дозволяють зробити наступні висновки:   
1) Було розроблено програмну реалізацію алгоритму, яка дозволяє знаходити оптимальний або близький до оптимального маршрут у графі міст.

2) На основі побудованого графіка якості рішення було показано, що збільшення кількості ітерацій дозволяє алгоритму знаходити кращі результати. Однак, із часом покращення стає менш вираженим, що свідчить про досягнення стабільності у знаходженні рішення.

3) Граф міст було представлено у вигляді симетричного графа, де вузли відповідають містам, а ребра — відстаням між ними. Найкращий маршрут, знайдений алгоритмом, було виділено червоним кольором.

4) Алгоритм продемонстрував здатність знаходити високоякісні рішення для графів великого розміру. Наприклад, у графі з 300 містами було знайдено найкоротший маршрут за 60 ітерацій, що свідчить про ефективність даного методу для задач такого типу.

Критерії оцінювання

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.