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
«Проектування алгоритмів»

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

Виконав(ла) *ІП-13 Бабашев О.Д* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О.О.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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 розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 M = 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 M = 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 M = 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 M = 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 M = 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

main.py

```
import random
from Ant import Ant

TOTAL_CITIES = 100
ITERATIONS = 100
M = 30
ALPHA = 2
BETA = 4
P = 0.4


def born_ants(total_ants, total_cities):
    ants = []
    occupied_cities = []

    for i in range(total_ants):
        city = random.randrange(0, total_cities)
        while city in occupied_cities:
            city = random.randrange(0, total_cities)
        occupied_cities.append(city)
        ant = Ant(city, TOTAL_CITIES)
        ants.append(ant)
    return ants


def generate_distance_matrix(total_cities):
    distance_matrix = []
    min_distance = 5
    max_distance = 50

    for i in range(total_cities):
        row = []
        for j in range(total_cities):
            if i == j:
                row.append(float('inf'))
            else:
                distance = random.randrange(min_distance, max_distance)
                row.append(distance)

        distance_matrix.append(row)
    return distance_matrix


def generate_pheromone_matrix(total_cities):
    initial_pheromone = 1.0
    pheromone_matrix = []

    for i in range(total_cities):
        row = []
        for j in range(total_cities):
            if i == j:
                row.append(0)
```

```
            else:
                row.append(initial_pheromone)
        pheromone_matrix.append(row)
    return pheromone_matrix


def update_pheromone_matrix(pheromone_matrix, travelled_ants, total_cities,
Lmin):
    for i in range(total_cities):
        for j in range(total_cities):

            if i != j:

                delta = 0
                for ant in travelled_ants[i][j]:
                    delta += Lmin / ant.total_distance
                pheromone_matrix[i][j] = (1 - P) * pheromone_matrix[i][j] +
delta
    return pheromone_matrix


def get_Lmin(pheromone_matrix, distance_matrix, travelled_ants, a):
    ant = Ant(0, TOTAL_CITIES)

    for i in range(TOTAL_CITIES):
        ant.walking(pheromone_matrix, distance_matrix, travelled_ants,
TOTAL_CITIES, a)

    return ant.total_distance


def get_Lk(pheromone_matrix, distance_matrix):
    counter = 0
    row = 0
    path_distance = 0
    path = [0]

    while counter < len(pheromone_matrix) - 1:
        max = -1
        i_max = -1

        for j in range(len(pheromone_matrix[row])):
            if j not in path and pheromone_matrix[row][j] > max:
                max = pheromone_matrix[row][j]
                i_max = j

        path_distance += distance_matrix[row][i_max]
        row = i_max
        path.append(i_max)
        counter += 1
    return path_distance


def print_distance(distance_matrix, pheromone_matrix):
    counter = 0
    row = 0
    distance = 0
    path = [0]

    while counter < len(pheromone_matrix) - 1:
        max_value = -1
        max_value_index = -1

        for j in range(len(pheromone_matrix[row])):
```

```
            if pheromone_matrix[row][j] > max_value and j not in path:
                max_value = pheromone_matrix[row][j]
                max_value_index = j

        distance += distance_matrix[row][max_value_index]
        row = max_value_index
        path.append(max_value_index)
        counter += 1

    print(f"shorter path = {distance}\n")


def main():
    distance_matrix = generate_distance_matrix(TOTAL_CITIES)
    pheromone_matrix = generate_pheromone_matrix(TOTAL_CITIES)

    travelled_ants = [[[] for i in range(TOTAL_CITIES)]
                      for j in range(TOTAL_CITIES)]

    Lmin = get_Lmin(pheromone_matrix, distance_matrix, travelled_ants, 0)
    shortest_path = Lmin
    count = 0

    while count != ITERATIONS:
        pheromone_matrix = generate_pheromone_matrix(TOTAL_CITIES)
        ants = born_ants(M, TOTAL_CITIES)

        for i in range(TOTAL_CITIES):
            for ant_counter in range(len(ants)):
                ants[ant_counter].walking(pheromone_matrix, distance_matrix,
travelled_ants, TOTAL_CITIES, ALPHA)
            pheromone_matrix = update_pheromone_matrix(pheromone_matrix,
travelled_ants, TOTAL_CITIES, Lmin)

        Lk = get_Lk(pheromone_matrix, distance_matrix)

        print(f"{count+1})\nshortest path = {shortest_path}\nLmin = {Lmin}")

        if Lk < shortest_path:
            print_distance(distance_matrix, pheromone_matrix)
            shortest_path = Lk

        count += 1

    print(f"final shortest path = {shortest_path}")


if __name__ == "__main__":
    main()
```

Ant.py

```
import random
BETA = 4


class Ant:
    def __init__(self, current_city, total_cities):
        self.start_city = current_city
        self.current_city = current_city
        self.visited_cities_count = 1
        self.total_distance = 0
        self.path = [self.current_city]

        self.unvisited cities = {}
        for i in range(total_cities):
            self.unvisited_cities[i] = True
        self.unvisited_cities[self.current_city] = False

    def walking(self, pheromone_matrix, distance_matrix, travelled_ants,
total_cities, a):
        if self.visited_cities_count < total_cities:
            denominator = 0

            for i in range(len(self.unvisited_cities)):
                if self.unvisited_cities[i]:
                    denominator += pow(pheromone_matrix[self.current_city][i],
a) * pow((1 / distance_matrix[self.current_city][i]), BETA)
            chances = {}

            for i in range(len(self.unvisited_cities)):
                if self.unvisited_cities[i]:
                    chance = (pow(pheromone_matrix[self.current_city][i], a) *
pow((1 / distance_matrix[self.current_city][i]), BETA)) / denominator
                    chances[i] = chance
                else:
                    chances[i] = 0
            discrete_quantity = {}
            discrete_quantity_sum = 0

            for i in range(len(chances)):
                discrete_quantity_sum += chances[i]
                discrete_quantity[i] = discrete_quantity_sum
            random_number = random.uniform(0,
discrete_quantity[len(discrete_quantity) - 1])
            previous_city = self.current_city
            next_city = 0

            for i in range(1, len(discrete_quantity)):
                if discrete_quantity[i - 1] < random_number <=
discrete_quantity[i]:
                    next_city = i
```

```
                break
            self.current_city = next_city
            if a > 0:
                travelled_ants[previous_city][next_city].append(self)
            self.total_distance += distance_matrix[previous_city][next_city]
            self.visited_cities_count += 1
            self.unvisited_cities[next_city] = False
            self.path.append(next_city)
        else:
            previous_city = self.current_city
            next_city = self.start_city
            self.current_city = self.start_city
            if a > 0:
                travelled_ants[previous_city][next_city].append(self)
            self.total_distance += distance_matrix[previous_city][next_city]
            self.path.append(next_city)
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
2)
shortest path = 912
Lmin = 912
shorter path = 824

3)
shortest path = 824
Lmin = 912
shorter path = 735

4)
shortest path = 735
Lmin = 912
5)
shortest path = 735
Lmin = 912
shorter path = 713

6)
shortest path = 713
Lmin = 912
7)
shortest path = 713
Lmin = 912
```

Рисунок 3.1 – Приклад роботи програми

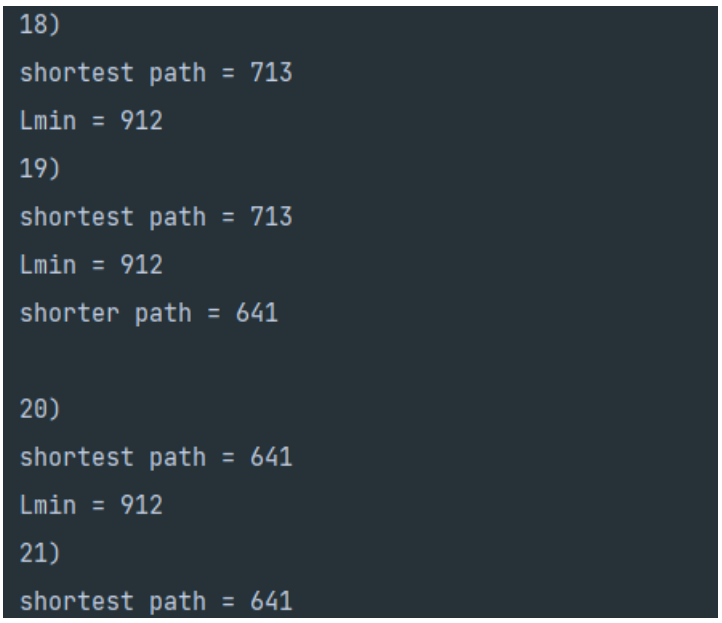

Рисунок 3.2 – Приклад роботи програми

Тестування алгоритму

3.1.3 Значення цільової функції зі збільшенням кількості ітерацій

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

Кількість ітерацій	Значення цільової функції	Кількість ітерацій	Значення цільової функції
1	866	560	611
20	678	580	611
40	643	600	611
60	643	620	611
80	643	640	611
100	643	660	611
120	643	680	611
140	643	700	611
160	643	720	611
180	643	740	611
200	631	760	603
220	631	780	603
240	631	800	603
260	631	820	603
280	611	840	603
300	611	860	603
320	611	880	603
340	611	900	603
360	611	920	603
380	611	940	598
400	611	960	598
420	611	980	598

440	611	1000	598
460	611		
480	611		
500	611		
520	611		
540	611		

3.1.4 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

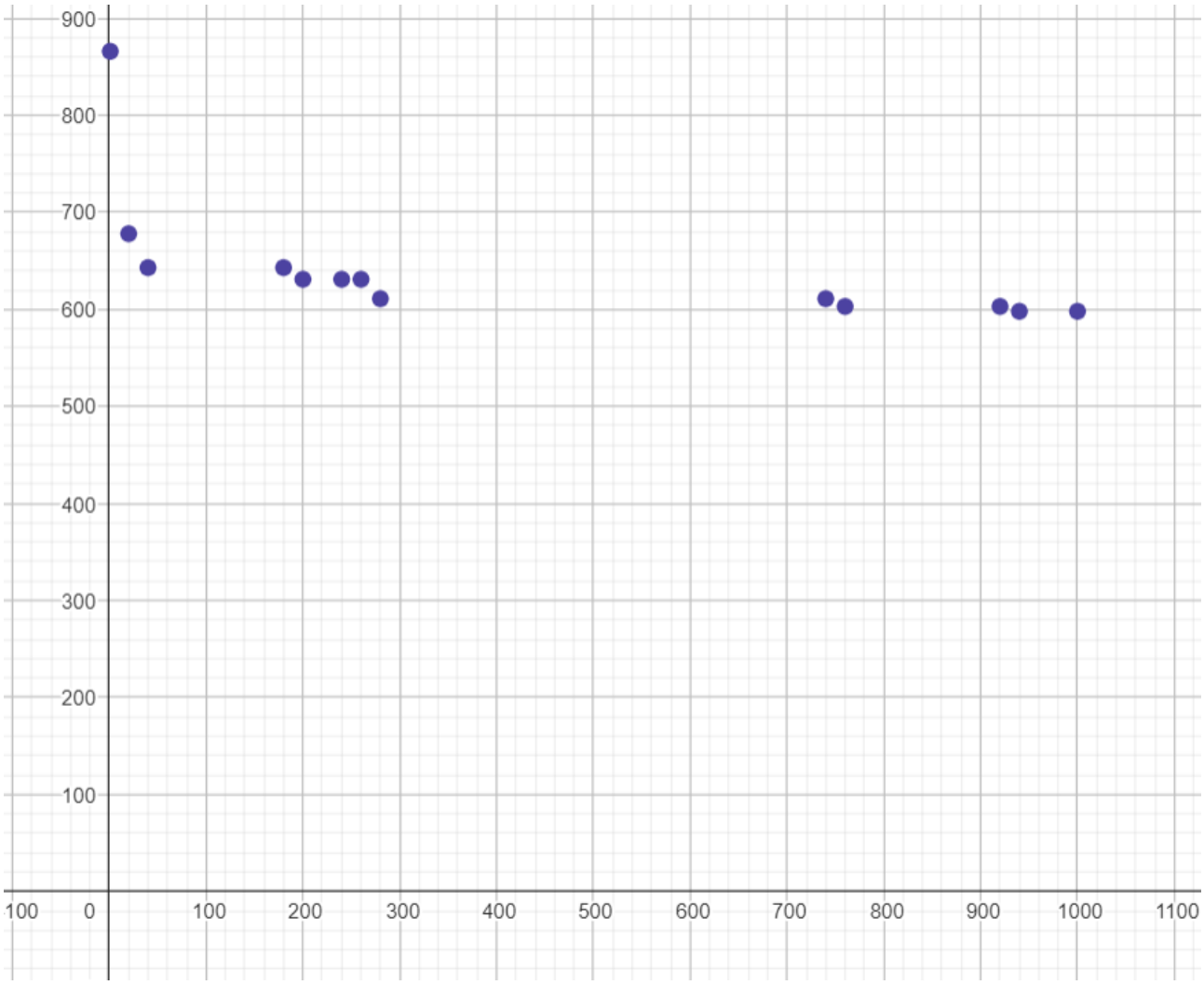

Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи вивчив основні підходи формалізації метаевристичних алгоритмів. Детально розглянув мурашиний алгоритм, реалізував на пайтоні вирішення типової задачі з його допомогою та проаналізував результати.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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