

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)

ІІІ-14 Медвідь Олександр
(шифр, прізвище, ім'я, по батькові)

Перевірив

Ахаладзе І.Е.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	10
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	10
3.1.1	<i>Вихідний код</i>	<i>10</i>
3.1.2	<i>Приклади роботи</i>	<i>18</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ	19
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій..</i>	<i>19</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій</i>	<i>21</i>
	ВИСНОВОК	22
	КРИТЕРІЇ ОЦІНЮВАННЯ	23

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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,3$, L_{\min} 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 ($\alpha = 2$, $\beta = 4$, ρ

	$= 0,6$, L_{\min} 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 ($\alpha = 4$, $\beta = 2$, $\rho = 0,3$, L_{\min} 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,7$, L_{\min} 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, L_{\min} 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, L_{\min} 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, L_{\min} 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, L_{\min} знайти жадібним алгоритмом, кількість мурах $M = 35$, починають маршрут в різних випадкових вершинах).

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

Файл Main.cpp

```
#include <iostream>
#include <string>
#include "Anthill.h"
#include "MatrixGraph.h"

int main()
{
    /*Користувач вводить назву файлу, він створюється.
    Далі генеруються значення елементів матриці випадковим чином*/
    string FileName;
    cout << "Enter the name of the file ";
    getline(cin, FileName);

    ofstream File(FileName);
    int size = 250;

    File << size;
    File << "\n";
    random_device Device;
    uniform_real_distribution<double> Dist(0, 50);
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            if (i == j)
            {
                File << 0.0;
                File << "\t";
            }
            else
            {
                File << Dist(Device);
                File << "\t";
            }
        }
        File << "\n";
    }
    File.close();

    MatrixGraph Graph(FileName);
    Anthill anthill(Graph);

    anthill.FindRoute();
}
```

Файл Ant.h

```
#pragma once
#include <random>
#include <memory>
#include "MatrixGraph.h"
#include "Route.h"
using namespace std;
```

```

//Клас представлення мурахи
class Ant
{
private:
    default_random_engine& Engine; //Генератор випадкових чисел
    int CurrentLocation; //Поточне місцерозташування мурахи
    int StartingLocation; //Початкове місцерозтішування
    unique_ptr<Route> route; //Шлях, пройдений мурахою
    bool IsAbleToMove; //Можливість мурахи продовжити рух
public:
    Ant(default_random_engine& engine, int Location, bool Exact); //Конструктор класу
    //Функція вибору подальшого кроку мурахи
    int Choice(const MatrixGraph& Pheromone, const MatrixGraph& Distance, double Alpha,
double Beta);
    int GetCurrentLocation() const; //Гетер поточного місцерозташування
    unique_ptr<Route> GetRoute(); //Гетер шляху, пройденого мурахою
    bool AbleToContinue() const; //Гетер можливості мурахи продовжувати рух
    bool AtTheStart() const; //Перевірка, чи мураха у стартовій локації
};

```

Файл Ant.cpp

```

#include "Ant.h"
#include <cmath>

/*Конструктор класу, генератор випадкових чисел передається параметром, можливість рухатись
задається як true
Якщо не важливо з якого саме міста починати, то початкова дислокація вибирається випадковим
чином
Стартова локація задається параметром поточної локації, ініціалізується шлях*/
Ant::Ant(default_random_engine& engine, int Location, bool Exact) : Engine(engine),
IsAbleToMove(true)
{
    if (Exact)
    {
        CurrentLocation = Location;
    }
    else
    {
        uniform_int_distribution<int> Distribution(0, Location - 1);
        CurrentLocation = Distribution(Engine);
    }
    StartingLocation = CurrentLocation;
    route = unique_ptr<Route>(new Route(CurrentLocation));
}

/*Функція вибору мурахою наступного кроку*/
int Ant::Choice(const MatrixGraph& Pheromone, const MatrixGraph& Distance, double Alpha,
double Beta)
{
    int From = CurrentLocation;

    if (route->GetSteps() == Distance.GetSize()) //Перевірка чи пройшла мураха усі міста
    {
        IsAbleToMove = false;
        CurrentLocation = StartingLocation;
        route->AddStep(StartingLocation, Distance.GetElement(From, StartingLocation));
        return StartingLocation;
    }

    vector<double> NeighbourProb;
    double MaxProb = 0.0;

```

```

double CurrentProb;
for (int i = 0, Size = Distance.GetSize(); i < Size; i++)
{
    if (i == From) //Не можна йти в те саме місто
    {
        NeighbourProb.push_back(0);
    }
    else if (route->Contains(i)) //Не можна двічі відвідувати місто
    {
        NeighbourProb.push_back(0);
    }
    else
    {
        CurrentProb = pow(Pheromone.GetElement(From, i), Alpha) *
pow(Distance.GetElement(From, i), Beta);
        MaxProb += CurrentProb;
        NeighbourProb.push_back(MaxProb);
    }
}

if (MaxProb == 0) //Нема куди йти
{
    IsAbleToMove = false;
    return SIZE_MAX;
}

uniform_real_distribution<double> Distribution(0.0, MaxProb);
double choice = Distribution(Engine);

for (int i = 0; i < NeighbourProb.size(); i++)
{
    if (choice < NeighbourProb[i])
    {
        CurrentLocation = i;
        route->AddStep(i, Distance.GetElement(From, i));
        return i;
    }
}

}

//Гетер поточного місцеположення
int Ant::GetCurrentLocation() const
{
    return CurrentLocation;
}

//Гетер шляху, пройденого мурахою
unique_ptr<Route> Ant::GetRoute()
{
    return move(route);
}

//Можливість мурахи йти далі
bool Ant::AbleToContinue() const
{
    return IsAbleToMove;
}

//Перевірка чи мураха у стартовій локації
bool Ant::AtTheStart() const
{
    if (CurrentLocation == StartingLocation)
    {
        return true;
    }
}

```

```

else
{
    return false;
}
}

```

Файл Anthill.h

```

#pragma once
#include <random>
#include <memory>
#include <vector>
#include "MatrixGraph.h"
#include "Ant.h"
using namespace std;

//Клас представлення мурашника
class Anthill
{
private:
    const double Alpha = 2; //Довіра феромону
    const double Beta = 4; //Довіра видимих міст
    const double Pheromone0 = 1.0; //Початковий феромон
    const double PheromoneEvaporation = 0.6; //Інтенсивність випаровування
    const double AntPheromone = 5; //Скільки феромону у мурахи
    const int Runs = 15; //Обхід графа без покращення результату

    MatrixGraph Distance; //Граф з вершинами задачі комівояжера
    MatrixGraph Pheromone; //Граф феромону

    random_device Device; //Прилад для генерації випадкових чисел
    default_random_engine Engine; //Генератор випадкових чисел

    vector <unique_ptr<Ant>> Ahill; //Масив мурах
    void CreateAnts(); //Функція створення мурах
public:
    Anthill(MatrixGraph Graph); //Конструктор класу
    void FindRoute(); //Функція пошуку найкращого маршруту для задачі комівояжера
};

```

Файл Anthill.cpp

```

#include "Anthill.h"
#include "pheromone.h"

/*Конструктор класу, атрибут Distance задається значенням графа, який використовується у
задачі комівояжера
Граф феромонів задається лямбда виразом: якщо індекси рівні, то задається як 0, бо не можна
перейти з міста в себе*/
Anthill::Anthill(MatrixGraph Graph)
    : Distance(Graph), Pheromone(Graph.GetSize(),
        [this](int Index1, int Index2) -> double {return Index1 != Index2 ? this->
Pheromone0 : 0; })),
    Device(), Engine(Device())
{
}

/*Функція створення мурах*/
void Anthill::CreateAnts()
{
    int M = 45;
    Ahill.resize(M);

    for (int i = 0; i < M; i++)
    {

```

```

        Ahill[i] = unique_ptr<Ant>(new Ant(Engine, Distance.GetSize(), false));
    }
}

/*Алгоритм мурах, головна функція для вирішення задачі.
Створюються мурахи, потім алгоритм проходиться по кожній мурасі.
Поки мураха може йти далі вона робить вибір куди йти та прокладає шлях
Якщо шлях, по якому пройшла мураха менший за попередній найкращий шлях, то
даний шлях записується у змінну BestLength. Після цього мураха прокладає феромон, той
випаровується*/
void Anthill::FindRoute()
{
    double BestLength = INFINITY;
    auto Cont = Runs;

    while (Cont > 0)
    {
        CreateAnts();

        for (unique_ptr<Ant>& ant : Ahill)
        {
            while (ant->AbleToContinue())
            {
                ant->Choice(Pheromone, Distance, Alpha, Beta);
            }

            auto CurrentRoute = ant->GetRoute();
            if (BestLength > CurrentRoute->GetDistance())
            {
                BestLength = CurrentRoute->GetDistance();
                Cont = Runs;
                cout << "Better route is found. It's length is " << BestLength <<
endl;
            }
            pheromone::Lay(Pheromone, *CurrentRoute, AntPheromone);
        }
        pheromone::Evaporate(Pheromone, PheromoneEvaporation);
        --Cont;
    }
}

```

MatrixGraph.h

```

#pragma once
#include <vector>
#include <iostream>
#include <random>
#include <string>
#include <functional>
#include <fstream>
using namespace std;

//Клас представлення графу, задається як матриця
class MatrixGraph
{
private:
    int Size; //Кількість вершин графу
    vector <vector <double>> Matrix; //Представлення матриці
public:
    MatrixGraph(string FileName); //Конструктор
    MatrixGraph(int size, function<double(int, int)> Generation); //Конструктор для графу
феромонів
    //Видалення конструктора без параметрів для ліквідації можливих створень об'єктів
класу без заданих параметрів
    MatrixGraph() = delete;
}

```

```

int GetSize() const; //Гетер кількості вершин графу (розмірності матриці)
double GetElement(int Index1, int Index2) const; //Гетер елементу матриці
void alter(int From, int To, function<void(double)> alterator); //Функція зміни
параметру
};

```

MatrixGraph.cpp

```

#include "MatrixGraph.h"

/*Конструктор класу, функція приймає рядок з назвою файла.
З даного файлу зчитується розмір матриці та її елементи*/
MatrixGraph::MatrixGraph(string FileName)
{
    ifstream File(FileName);
    File >> Size;
    Matrix.resize(Size);

    for (int i = 0; i < Size; i++)
    {
        Matrix[i].resize(Size);
        for (int j = 0; j < Size; j++)
        {
            File >> Matrix[i][j];
        }
    }

    File.close();
}

/*Конструктор для графу феромонів, у нього передається лямбда вираз
Після цього значення елементів генерується за переданою функцією*/
MatrixGraph::MatrixGraph(int size, function<double(int, int)> Generation)
{
    Size = size;
    Matrix.resize(Size);
    for (int i = 0; i < Size; i++)
    {
        Matrix[i].resize(Size);
        for (int j = 0; j < Size; j++)
        {
            Matrix[i][j] = Generation(i, j);
        }
    }
}

//Гетер кількості вершин графу (розмірність матриці)
int MatrixGraph::GetSize() const
{
    return Size;
}

//Гетер елементу матриці
double MatrixGraph::GetElement(int Index1, int Index2) const
{
    return Matrix[Index1][Index2];
}

//Функція зміни параметру
void MatrixGraph::alter(int From, int To, function<void(double)> alterator)
{
    alterator(Matrix[From][To]);
}

```

```
}
```

Route.h

```
#pragma once
#include <vector>
#include <iostream>
#include "MatrixGraph.h"
using namespace std;

//Клас представлення шляху
class Route
{
private:
    vector<int> Path; //Представлення шляху
    double Distance; //Дистанція, яку займає шлях
public:
    Route(int Initial); //Конструктор класу
    double GetDistance() const; //Гетер дистанції
    int GetSteps() const; //Отримання кількості, пройдених кроків
    int operator[] (int element) const; //Перевантаження оператора [] для ітерування
елементами класу
    void AddStep(int Vertex, double distance); //Додавання кроку до шляху
    bool Contains(int Location); //Перевірка чи була пройдена локація
};
```

Route.cpp

```
#include "Route.h"

//Конструктор класу
Route::Route(int Initial)
{
    Distance = 0;
    this->AddStep(Initial, 0);
}

//Гетер пройденної дистанції
double Route::GetDistance() const
{
    return Distance;
}

//Гетер кількості пройдених кроків
int Route::GetSteps() const
{
    return Path.size();
}

//Перевантаження оператора [] для ітерування елементами класу
int Route::operator[](int element) const
{
    return Path[element];
}

//Додавання кроку до шляху
void Route::AddStep(int Vertex, double distance)
{
    Path.push_back(Vertex);
    Distance += distance;
}

//Перевірка чи була пройдена локація
```



```

bool Route::Contains(int Location)
{
    return find(Path.begin(), Path.end(), Location) != Path.end();
}

pheromone.h

#pragma once
#include "MatrixGraph.h"
#include "Route.h"

namespace pheromone
{
    void Lay(MatrixGraph& Phero, const Route& route, double AntPheromone); //Функція
    прокладування феромону
    void Evaporate(MatrixGraph& Phero, double Evaporation); //Функція випаровування
    феромону
}

pheromone.cpp

#include "pheromone.h"

//Функція прокладування феромону
void pheromone::Lay(MatrixGraph& Phero, const Route& route, double AntPheromone)
{
    double PheromoneToLay = AntPheromone / route.GetSteps();
    for (int i = 1, n = route.GetSteps(); i < n; i++)
    {
        Phero.alter(route[i - 1], route[i], [PheromoneToLay](double& val) {val +=
PheromoneToLay; });
    }
}

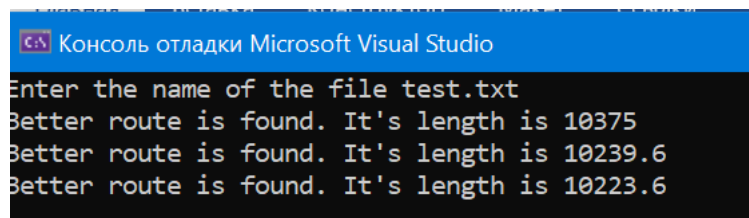
//Функція випаровування феромону
void pheromone::Evaporate(MatrixGraph& Phero, double Evaporation)
{
    for (int i = 0, n = Phero.GetSize(); i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            Phero.alter(i, j, [Evaporation](double& val) {val *= 1 - Evaporation;
});
        }
    }
}

```

3.1.2 Приклади роботи

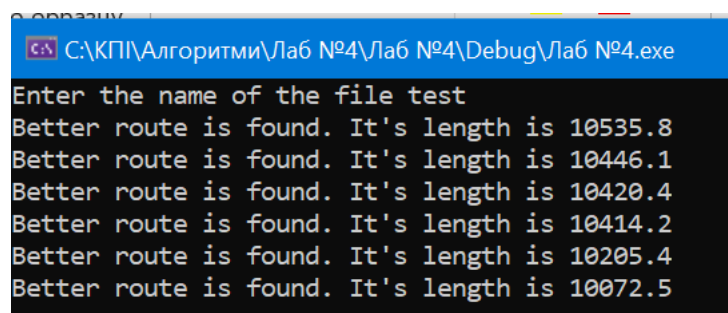
На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –



```
Консоль отладки Microsoft Visual Studio
Enter the name of the file test.txt
Better route is found. It's length is 10375
Better route is found. It's length is 10239.6
Better route is found. It's length is 10223.6
```

Рисунок 3.2 –



```
C:\КП\Алгоритми\Лаб №4\Лаб №4\Debug\Лаб №4.exe
Enter the name of the file test
Better route is found. It's length is 10535.8
Better route is found. It's length is 10446.1
Better route is found. It's length is 10420.4
Better route is found. It's length is 10414.2
Better route is found. It's length is 10205.4
Better route is found. It's length is 10072.5
```

3.2 Тестування алгоритму

3.2.1 Значення цільової функції зі збільшенням кількості ітерацій

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

Ітерація	Значення цільової функції
1	10327.6
2	10225.8
3	10198.9
4	10184
5	10389.2
6	9985.52
7	10429.8
8	10235.4
9	10281.3
10	10233.2
11	10180.7
12	10397.4
13	10212.8
14	10234.2
15	10211.4
16	10324
17	10087.3
18	10150.4
19	10200.12
20	10231.3
21	10432.9
22	10321.6
23	10212
24	10111

25	10123.4
26	10032.2
27	10372.3
28	10123.2
29	10345
30	10534.17
31	10324.4
32	10234.2
33	10324.5
34	10432.4
35	10234.6
36	10367.6
37	10213.2
38	10534
39	10321.23
40	10103
41	10213.4
42	10234.6
43	10237.8
44	10020.12
45	10213.3

Таблиця 3.1 – Значення цільової функції за кількості ітерацій

3.2.2 Графіки залежності розв'язку від числа ітерацій

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

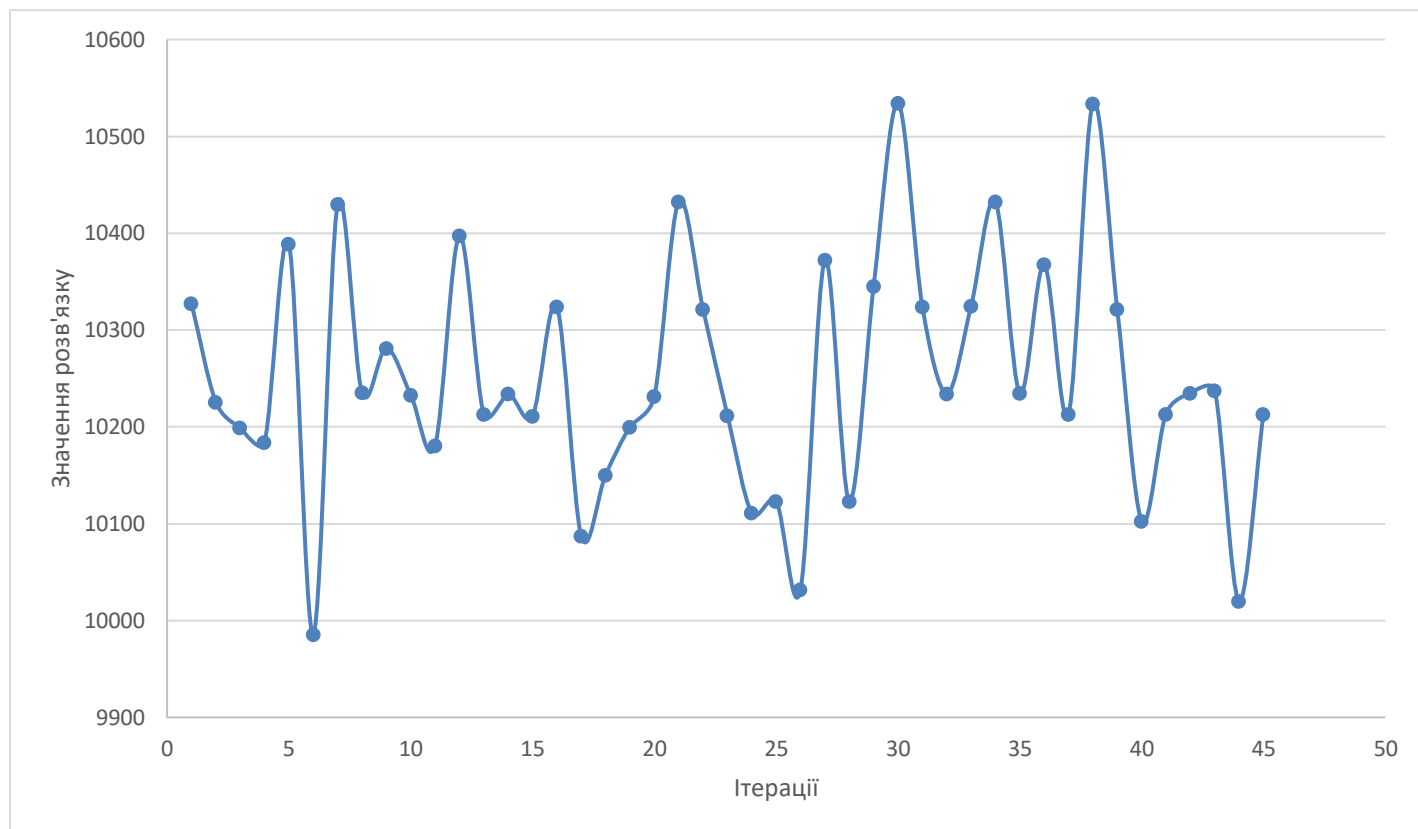


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи був розглянутий мурашиний алгоритм на прикладі задачі комівояжера. Були набуті навички їх використання у програмних специфікаціях. Для вирішення задачі була створена елементарна програма. Результати програми виявилися правильними, що стверджує на її дієвість. Завдання було виконано на мові програмування C++. На рахунок алгоритму мурах, можна сказати, що він не є оптимальним. Через те, що алгоритм працює постійно з випадковими числами, знаходження найоптимальнішого результату не є гарантованим.

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%.