

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 № 5 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла)

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

Перевірив

Головченко М.М.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	11
3.1	ПОКРОКОВИЙ АЛГОРИТМ	11
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	11
3.2.1	<i>Вихідний код.....</i>	<i>11</i>
3.2.2	<i>Приклади роботи.....</i>	<i>20</i>
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	21
	ВИСНОВОК	24
	КРИТЕРІЇ ОЦІНЮВАННЯ	24

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

2 ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім ступенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

- обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
- зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
- після цього параметр фіксується і змінюються інші параметри;
- далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
- зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

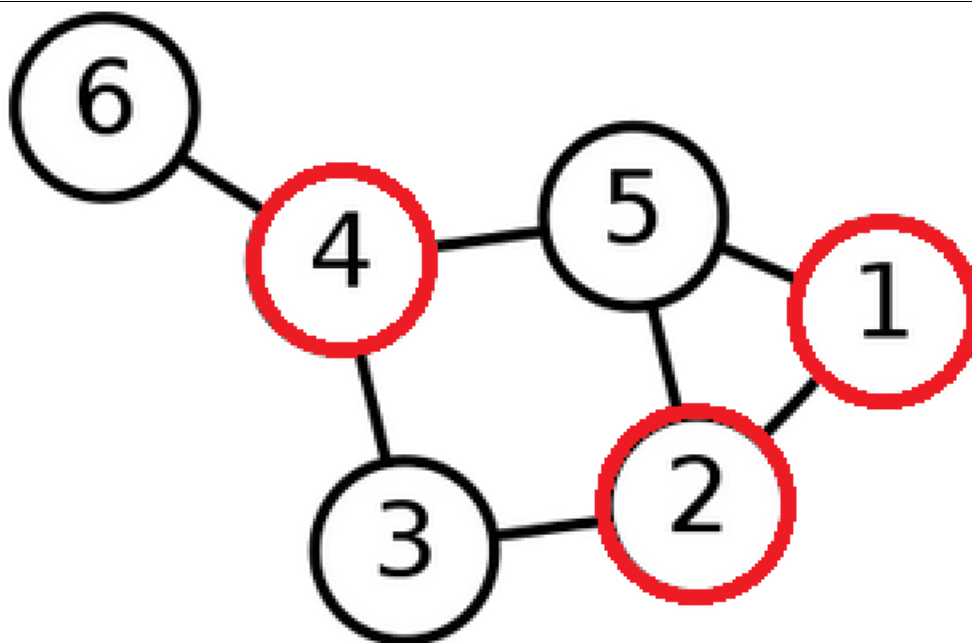
Зробити узагальнений висновок в якому обов'язково описати залежність якості розв'язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	Задача
1	Задача про рюкзак (місткість $P=500$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб

	<p>сумарна вага не перевищувала задану, а сумарна цінність була максимальною.</p> <p>Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.</p>
2	<p>Задача комівояжера (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.</p> <p>Розглядається симетричний, асиметричний та змішаний варіанти.</p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p> <p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> — доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); — доставка води;

	<ul style="list-style-type: none"> – моніторинг об'єктів; – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.
3	<p>Розфарбовування графа (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – розкладу для комунальних служб;
4	<p>Задача вершинного покриття (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа $G = (V, E)$ - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.</p> <p>Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).</p> <p>На вході: Граф $G = (V, E)$.</p> <p>Результат: множина $C \subseteq V$ - найменше вершинне покриття графа G.</p>



Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;
- проектування інтегральних схем і конвеєрних ліній.

5 **Задача про кліку** (300 вершин, степінь вершини не більше 30, але не менше 2). Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.

Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k , тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).

Застосування:

- біоінформатика;
- електротехніка;

6 **Задача про найкоротший шлях** (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але

	<p>не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.</p> <p>Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.</p>
--	--

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
1	<p>Генетичний алгоритм:</p> <ul style="list-style-type: none"> - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).
2	<p>Мурашиний алгоритм:</p> <ul style="list-style-type: none"> – α; – β; – ρ; – L_{min}; – кількість мурах M і їх типи (елітні, тощо...); – маршрути з однієї чи різних вершин.
3	<p>Бджолиний алгоритм:</p> <ul style="list-style-type: none"> – кількість ділянок; – кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм

28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3.1 Покроковий алгоритм

while Cycle < CycleLimit:

 WaggleRoute = hive.Waggle(500, ceil(180 * 0.2), BestRoute)

 if WaggleRoute.GetDistance() < BestDistance:

 BestDistance = WaggleRoute.GetDistance()

 BestRoute.ChangePath(WaggleRoute.GetPath(), BestDistance)

 RecruitRoute = hive.Recruit(BestRoute)

 if RecruitRoute.GetDistance() < BestDistance:

 BestDistance = RecruitRoute.GetDistance()

 BestRoute.ChangePath(RecruitRoute.GetPath(), BestDistance)

 Cycle += 1

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Файл Main.cpp

```
#include <iostream>
#include <string>
#include "MatrixGraph.h"
#include "Hive.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 = 300;

    File << size;
    File << "\n";
    random_device Device;
    uniform_real_distribution<double> Dist(5, 150);
    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);
int Cycle = 1;
int CycleLimit = 150;
double BestDistance = INFINITY;
Hive hive(Graph);
hive.CreateBees();
Route BestRoute;
Route WaggleRoute(0);
Route RecruitRoute(0);

while (Cycle < CycleLimit)
{
    WaggleRoute = hive.Waggle(500, ceil(180 * 0.2), BestRoute);
    if (WaggleRoute.GetDistance() < BestDistance)
    {
        BestDistance = WaggleRoute.GetDistance();
        BestRoute.ChangePath(WaggleRoute.GetPath(), BestDistance);
        cout << "Better route is found. It's length is " << BestDistance <<
endl;
    }
    RecruitRoute = hive.Recruit(BestRoute);
    if (RecruitRoute.GetDistance() < BestDistance)
    {
        BestDistance = RecruitRoute.GetDistance();
        BestRoute.ChangePath(RecruitRoute.GetPath(), BestDistance);
        cout << "Better route is found. It's length is " << BestDistance <<
endl;
    }

    Cycle += 1;
}
}

```

Файл Bee.h

```

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

//Клас представлення бджоли
class Bee
{
private:
    Route route; //Шлях, пройдений бджолою
    //Onlooker - 1, Forager - 2, Scout - 3
    int Role; //Роль бджоли
    int Cycle; //Кількість ітерацій під час вирішення задачі
public:
    Bee(int role, MatrixGraph Graph); //Конструктор класу
    void ShuffleBeeRoute(MatrixGraph Graph); //Функція перемішування шляху
    void Forage(MatrixGraph Graph, int Limit); //Функціонал бджол-працівників
    void SetRoute(vector<int> OtherRoute, double distance); //Функція заміни шляху бджоли
    void Scout(MatrixGraph Graph); //Функціонал бджоли-розвідника
    Route GetRoute(); //Гетер шляху, пройденного мурахою

```

```

void SetRole(int Value); //Сетер ролі
int GetRole(); //Гетер ролі бджоли
int GetCycle(); //Гетер ітерацій
};

```

Файл Bee.cpp

```

#include "Bee.h"

//Конструктор класу
Bee::Bee(int role, MatrixGraph Graph)
{
    Role = role;
    route.RandomRoute(Graph);
}

//Функція перемішування шляху
void Bee::ShuffleBeeRoute(MatrixGraph Graph)
{
    route.ShuffleRoute();
    route.UpdateDistance(Graph);
}

//Функціонал бджол-працівників
void Bee::Forage(MatrixGraph Graph, int Limit)
{
    Route NewPath = route.MutatePath(Graph);

    if (NewPath.GetDistance() < route.GetDistance())
    {
        route.ChangePath(NewPath.GetPath(), NewPath.GetDistance());
        Cycle = 0; //Скинути к-ть ітерацій щоб бджола могла далі працювати
    }
    else
    {
        Cycle += 1;
    }

    if (Cycle >= Limit) //Якщо бджола не робить прогресу
    {
        Role = 3;
    }
}

//Функція заміни шляху бджоли
void Bee::SetRoute(vector<int> OtherRoute, double distance)
{
    route.ChangePath(OtherRoute, distance);
}

//Функціонал бджоли-розвідника
void Bee::Scout(MatrixGraph Graph)
{
    route.ShuffleRoute();
    route.UpdateDistance(Graph);
    Role = 2;
    Cycle = 0;
}

//Гетер шляху, пройденого мурахою
Route Bee::GetRoute()
{
    return route;
}

```

```

//Сетер ролі
void Bee::SetRole(int Value)
{
    Role = Value;
}

//Гетер ролі бджоли
int Bee::GetRole()
{
    return Role;
}

//Гетер ітерацій
int Bee::GetCycle()
{
    return Cycle;
}

```

Файл Hive.h

```

#pragma once
#include <random>
#include <vector>
#include "MatrixGraph.h"
#include "Bee.h"

//Клас представлення вулику
class Hive
{
private:
    int Population; //Кількість бджол
    MatrixGraph Distance; //Граф з вершинами задачі комівояжера
    random_device Device; //Прилад для генерації випадкових чисел
    default_random_engine Engine; //Генератор випадкових чисел
    vector <unique_ptr<Bee>> BeeHive; //Масив бджол
public:
    Hive(MatrixGraph Graph); //Конструктор класу
    void CreateBees(); //Функція створення бджол
    Route Waggle(int ForagerLimit, double ScoutCount, Route BestRoute); //Збір інформації
    Route Recruit(Route BestRoute); //Перегляд найкращих результатів
};

```

Файл Hive.cpp

```

#include "Hive.h"

//Передається граф з містами задачі комівояжера
Hive::Hive(MatrixGraph Graph) : Distance (Graph)
{
    Population = 180;
}

//Функція створення бджол
void Hive::CreateBees()
{
    int Onlooker_Count, Forager_Count;

    Onlooker_Count = floor(Population * 0.5);
    Forager_Count = Onlooker_Count;
    BeeHive.resize(Population);
    for (int i = 0; i < Onlooker_Count; i++)
    {
        BeeHive[i] = unique_ptr<Bee>(new Bee(1, Distance));
    }
}

```

```

for (int i = Onlooker_Count; i < (Onlooker_Count + Forager_Count); i++)
{
    BeeHive[i] = unique_ptr<Bee>(new Bee(2, Distance));
    BeeHive[i]->ShuffleBeeRoute(Distance);
}
}

/*Збирається інформація на рахунок праці бджол - працівників
Вибирається новий випадковий шлях для розвідників.*/
Route Hive::Waggle(int ForagerLimit, double ScoutCount, Route BestRoute)
{
    int Iterator = 0;
    vector<int> Indexes;
    vector<double> ForagerDistances;
    vector<double> ForagerDistancesSorted;
    for (unique_ptr<Bee>& bee : BeeHive)
    {
        if (bee->GetRole() == 2)
        {
            bee->Forage(Distance, ForagerLimit);
            if (bee->GetRoute().GetDistance() < BestRoute.GetDistance())
            {
                BestRoute.ChangePath(bee->GetRoute().GetPath(), bee-
>GetRoute().GetDistance());
            }
            Indexes.push_back(Iterator);
            ForagerDistances.push_back(BestRoute.GetDistance());
            ForagerDistancesSorted.push_back(BestRoute.GetDistance());
        }
        else if (bee->GetRole() == 3)
        {
            bee->Scout(Distance);
        }
        Iterator++;
    }

    vector<double>::iterator Ind;
    sort(ForagerDistancesSorted.begin(), ForagerDistancesSorted.end(),
greater<double>{});
    for (int i = 0; i < ScoutCount && i < ForagerDistances.size(); i++)
    {
        Ind = find(ForagerDistances.begin(), ForagerDistances.end(),
ForagerDistancesSorted[i]);
        BeeHive[Indexes[Ind - ForagerDistances.begin()]]->SetRole(3);
    }

    return BestRoute;
}

/*Переглядається найкращі результати, які були до цього моменту отримані*/
Route Hive::Recruit(Route BestRoute)
{
    for (unique_ptr<Bee>& bee : BeeHive)
    {
        if (bee->GetRole() == 1)
        {
            bee->SetRoute(BestRoute.GetPath(), BestRoute.GetDistance());
            Route MutantRoute(0);
            MutantRoute = bee->GetRoute().MutatePath(Distance);
            bee->SetRoute(MutantRoute.GetPath(), MutantRoute.GetDistance());
            if (bee->GetRoute().GetDistance() < BestRoute.GetDistance())
            {
                BestRoute.ChangePath(bee->GetRoute().GetPath(), bee-
>GetRoute().GetDistance());
            }
        }
    }
}

```

```

    }
}

return BestRoute;
}

```

Файл Route.h

```

#pragma once
#include <vector>
#include <iostream>
#include <algorithm>
#include <random>
#include "MatrixGraph.h"

//Клас представлення шляху
class Route
{
private:
    vector<int> Path; //Представлення шляху
    double Distance; //Дистанція, яку займає шлях
public:
    Route(); //Конструктор класу
    Route(int Initial); //Конструктор класу
    Route(vector<int> path, MatrixGraph Graph); //Конструктор класу
    double GetDistance() const; //Гетер дистанції
    void RandomRoute(MatrixGraph Graph); //Рандомізувати шлях
    void ShuffleRoute(); //Перемішування шляху
    void ChangePath(vector<int> OtherRoute, double distance); //Функція заміни шляху
    Route MutatePath(MatrixGraph Graph); //Функція зміни шляху
    void UpdateDistance(MatrixGraph Graph); //Функція оновлення пройденної дистанції
    int GetSteps() const; //Отримання кількості, пройдених кроків
    vector<int> GetPath(); //Гетер шляху
    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& operator = (Route route); //Перевантаження оператора присвоєння
};

```

Файл Route.cpp

```

#include "Route.h"

//Конструктор класу
Route::Route()
{
    Distance = INFINITY;
}

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

//Конструктор класу
Route::Route(vector<int> path, MatrixGraph Graph)
{
    Path = {path.begin(), path.end()};
    this->UpdateDistance(Graph);
}

```



```

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

//Рандомізація шляху
void Route::RandomRoute(MatrixGraph Graph)
{
    Path.resize(Graph.GetSize());
    for (int i = 0; i < Graph.GetSize(); i++)
    {
        Path[i] = i;
    }

    this->ShuffleRoute();
}

//Перемішування шляху
void Route::ShuffleRoute()
{
    random_shuffle(begin(Path), end(Path));
}

//Функція заміни шляху
void Route::ChangePath(vector<int> OtherRoute, double distance)
{
    Path.resize(OtherRoute.size());
    Path = { OtherRoute.begin(), OtherRoute.end() };
    Distance = distance;
}

//Функція зміни шляху
Route Route::MutatePath(MatrixGraph Graph)
{
    int Temp;
    random_device Device;
    uniform_int_distribution<int> Dist(0, (Path.size() - 2));

    int RandIndex = Dist(Device);
    vector<int> path = {Path.begin(), Path.end()};
    Temp = path[RandIndex];
    path[RandIndex] = path[RandIndex + 1];
    path[RandIndex + 1] = Temp;

    Route route(path, Graph);
    return route;
}

//Функція оновлення пройденної дистанції
void Route::UpdateDistance(MatrixGraph Graph)
{
    Distance = 0;

    for (int i = 0; i < Path.size() - 1; i++)
    {
        Distance += Graph.GetElement(Path[i], Path[i+1]);
    }
}

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

```

```

vector<int> Route::GetPath()
{
    return Path;
}

//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();
}

//Перевантаження оператора присвоєння
Route& Route::operator=(Route route)
{
    Path = { route.Path.begin(), route.Path.end() };
    Distance = route.Distance;

    return* this;
}

```

Файл 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
    //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
};

```

Файл 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();
}

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

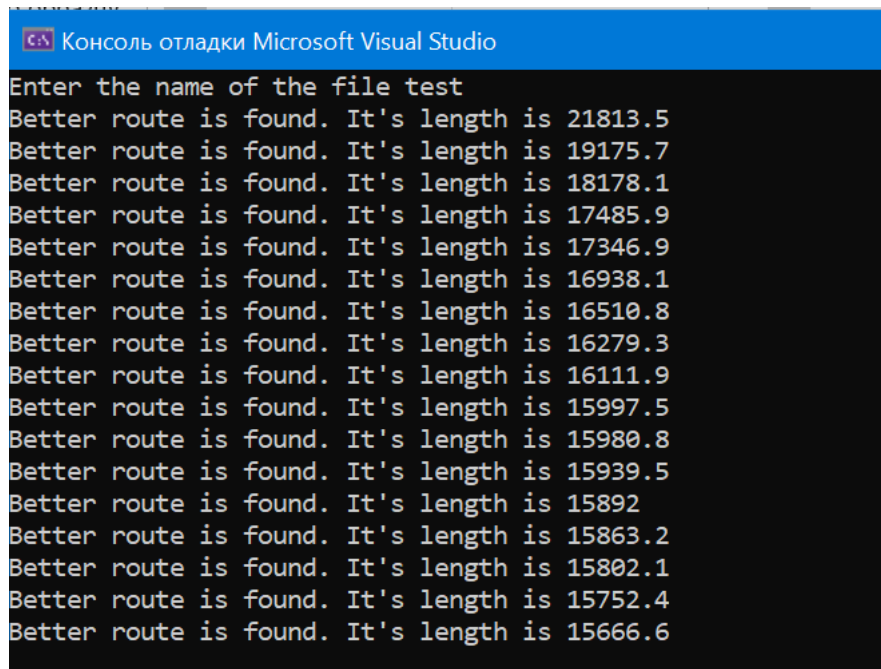
//Гетер елементу матриці
double MatrixGraph::GetElement(int Index1, int Index2) const
{
    return Matrix[Index1][Index2];
}

```

3.2.2 Приклади роботи

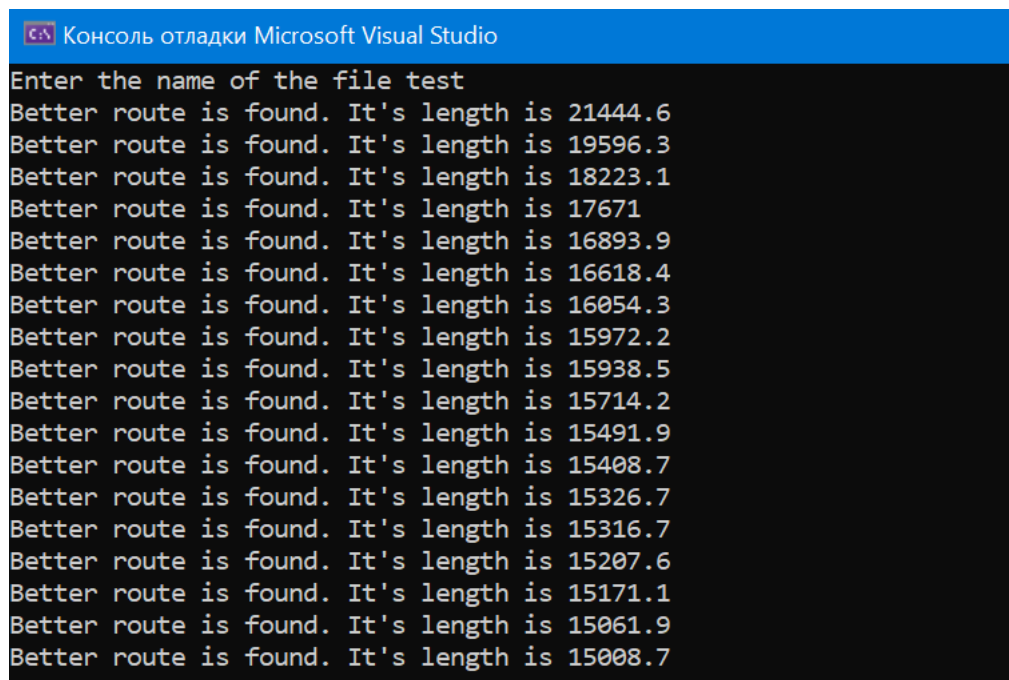
На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –



```
Консоль отладки Microsoft Visual Studio
Enter the name of the file test
Better route is found. It's length is 21813.5
Better route is found. It's length is 19175.7
Better route is found. It's length is 18178.1
Better route is found. It's length is 17485.9
Better route is found. It's length is 17346.9
Better route is found. It's length is 16938.1
Better route is found. It's length is 16510.8
Better route is found. It's length is 16279.3
Better route is found. It's length is 16111.9
Better route is found. It's length is 15997.5
Better route is found. It's length is 15980.8
Better route is found. It's length is 15939.5
Better route is found. It's length is 15892
Better route is found. It's length is 15863.2
Better route is found. It's length is 15802.1
Better route is found. It's length is 15752.4
Better route is found. It's length is 15666.6
```

Рисунок 3.2 –



```
Консоль отладки Microsoft Visual Studio
Enter the name of the file test
Better route is found. It's length is 21444.6
Better route is found. It's length is 19596.3
Better route is found. It's length is 18223.1
Better route is found. It's length is 17671
Better route is found. It's length is 16893.9
Better route is found. It's length is 16618.4
Better route is found. It's length is 16054.3
Better route is found. It's length is 15972.2
Better route is found. It's length is 15938.5
Better route is found. It's length is 15714.2
Better route is found. It's length is 15491.9
Better route is found. It's length is 15408.7
Better route is found. It's length is 15326.7
Better route is found. It's length is 15316.7
Better route is found. It's length is 15207.6
Better route is found. It's length is 15171.1
Better route is found. It's length is 15061.9
Better route is found. It's length is 15008.7
```

3.3 Тестування алгоритму

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

Ітерація	Значення цільової функції
1	21023.12
2	20078.2
3	19812.9
4	19921.2
5	18389.2
6	19985.52
7	19429.8
8	19235.4
9	19281.3
10	19233.2
11	19180.7
12	18397.4
13	19212.8
14	19234.2
15	19211.4
16	19324
17	19087.3
18	19150.4
19	19200.12
20	18231.3
21	17432.9
22	16321.6
23	17212
24	18111
25	17123.4

26	17032.2
27	17372.3
28	16123.2
29	17345
30	17534.17
31	16324.4
32	16234.2
33	16433.4
34	16893.9
35	16043.23
36	16032.12
37	15908.54
38	15943
39	15234.2
40	15512.13

Таблиця 3.1 – Значення цільової функції за кількості ітерацій

3.3.1 Графіки залежності розв'язку від числа ітерацій

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

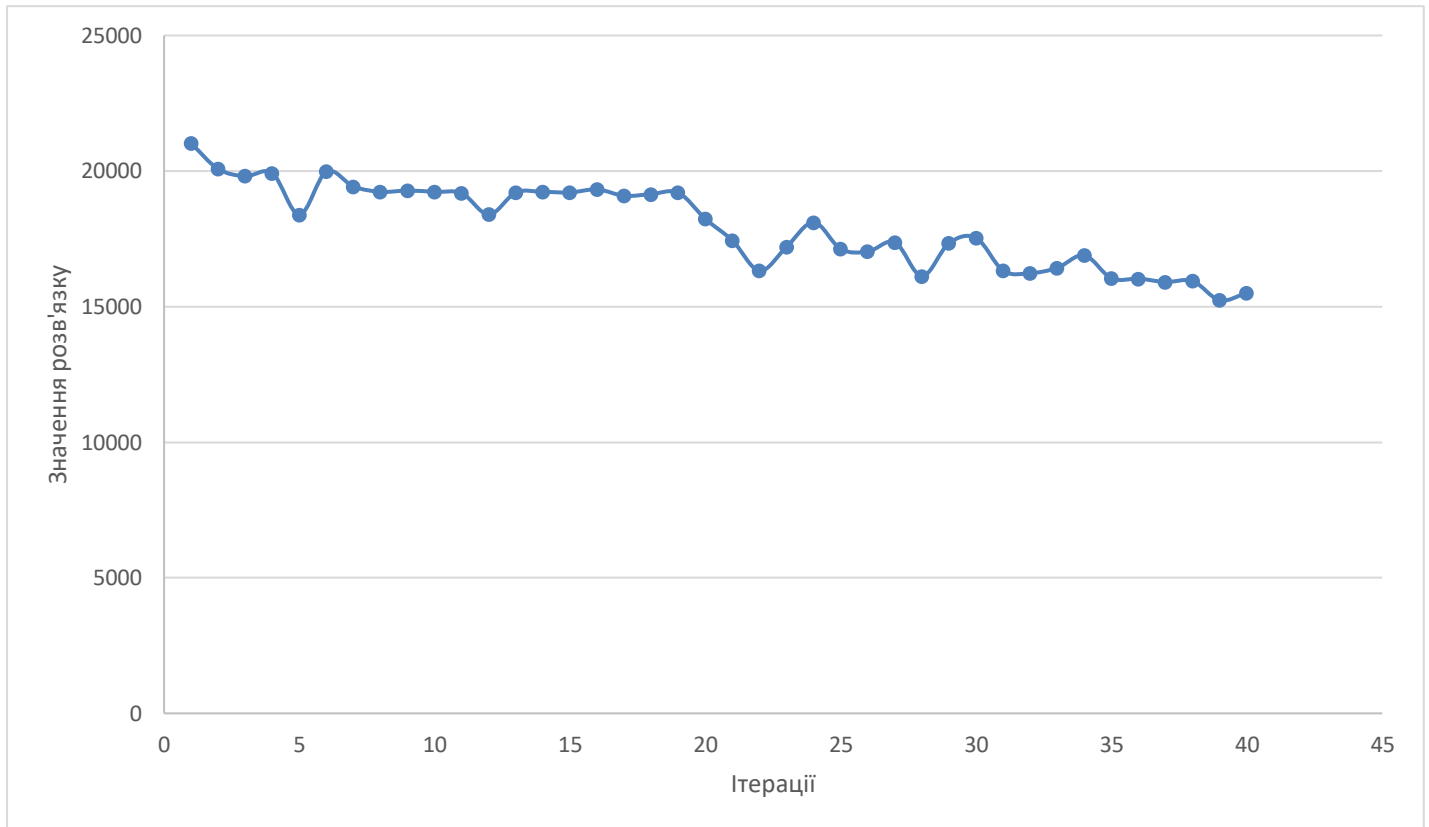


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

В рамках тестування алгоритму, змінювались 2 параметри: кількість ітерацій для зупинки алгоритму та кількість бджіл. Зі зменшенням кількості ітерацій, алгоритм працював швидше, але, звичайно, не мав достатньо часу щоб знайти найоптимальніший результат. Зі збільшенням кількості бджіл, алгоритм починає довше працювати, але встигає перебрати більше комбінацій за дану кількість ітерацій, тому і вірогідність знайти кращий результат більша.

ВИСНОВОК

В рамках даної лабораторної роботи був розглянутий бджолиний 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 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму – 30%;
- висновок – 5%.