**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Гуськов Кирило Михайлович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[**1**](#_heading=h.30j0zll) **Мета лабораторної роботи 3**

[**2**](#_heading=h.1fob9te) **Завдання 4**

[**3**](#_heading=h.3znysh7) **Виконання 10**

[3.1](#_heading=h.2et92p0) Програмна реалізація алгоритму 10

[*3.1.1*](#_heading=h.tyjcwt) *Вихідний код 10*

[*3.1.2*](#_heading=h.3dy6vkm) *Приклади роботи 10*

[3.2](#_heading=h.1t3h5sf) Тестування алгоритму 11

[*3.2.1*](#_heading=h.4d34og8) *Значення цільової функції зі збільшенням кількості ітерацій 11*

[*3.2.2*](#_heading=h.2s8eyo1) *Графіки залежності розв'язку від числа ітерацій 11*

[**Висновок 12**](#_heading=h.17dp8vu)

[**Критерії оцінювання 13**](#_heading=h.3rdcrjn)

# 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 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 (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

PA’Lab4.cpp

#include<iostream>

#include<ctime>

#include"Header.h"

usingnamespacestd;

intmain()

{

srand(time(NULL));

intgraph[200][200];

doublepheromoneGraph[200][200];

doublepheromoneSumGraph[200][200];

BuildGraph(graph);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

TSPAlgorithm(graph,pheromoneGraph,pheromoneSumGraph);

}

voidBuildGraph(intgraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

if(i==j)

{

graph[i][j]=0;

}

else

{

graph[i][j]=rand()%51;

}

}

}

}

voidBuildPheromoneGraph(doublepheromoneGraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

if(i==j)

{

pheromoneGraph[i][j]=0;

}

else

{

pheromoneGraph[i][j]=rand()%9+1;

pheromoneGraph[i][j]=pheromoneGraph[i][j]/10;

}

}

}

}

voidBuildPheromoneSumGraph(doublepheromoneSumGraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

pheromoneSumGraph[i][j]=0;

}

}

}

voidBuildVisibilityGraph(doublevisibilityGraph[200][200],intgraph[200][200])

{

for(inti=0;i<200;i++)

{

for(intj=0;j<200;j++)

{

visibilityGraph[i][j]=1/graph[i][j];

}

}

}

Header.H

#pragmaonce

#include<iostream>

#include<vector>

usingnamespacestd;

structProbability

{

doublechance;

intvertix;

};

voidBuildGraph(intgraph[200][200]);

voidBuildPheromoneGraph(doublepheromoneGraph[200][200]);

voidBuildPheromoneSumGraph(doublepheromoneSumGraph[200][200]);

voidBuildVisibilityGraph(doublevisibilityGraph[200][200],intgraph[200][200]);

voidTSPAlgorithm(intgraph[200][200],doublepheromoneGraph[200][200],doublepheromoneSumGraph[200][200]);

intGenerateStartingPoint(intUsedStartingPoints[45],intM);

boolCheckForStartingPoint(intUsedStartingPoints[45],intStartingPoint,intM);

doubleMoveProbability(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200],doubleStepTwo);

doubleMoveProbabilityStepOne(intcurVertix,intdestination,doublepheromone,intalpha,intbeta,intgraph[200][200]);

doubleGetLmin(intgraph[200][200]);

intGetMin(intgraph[200][200],intcurrVertix);

boolCheckVisited(intvisited[200]);

voidprintArray(intarr[200]);

Ant.H

#pragmaonce

#include"Header.h"

classAnt

{

public:

intStartingPoint;

vector<int>Path;

vector<int>UnvisitedVertices;

Ant()

{

this->Path.push\_back(1);

this->UnvisitedVertices.push\_back(1);

this->Path.clear();

this->UnvisitedVertices.clear();

}

Ant(intStartingPoint);

voidReset();

voidMoveToVertix(intdestination,intdestinationIdx);

voidChooseDestination(vector<Probability>Probabilities);

doublegetL(intgraph[200][200]);

};

Ant.cpp

#include"Ant.h"

Ant::Ant(intStartingPoint)

{

this->StartingPoint=StartingPoint;

this->Path.push\_back(StartingPoint);

for(inti=0;i<200;i++)

{

if(i!=StartingPoint)

{

this->UnvisitedVertices.push\_back(i);

}

}

}

voidAnt::Reset()

{

this->Path.clear();

this->Path.push\_back(this->StartingPoint);

this->UnvisitedVertices.clear();

for(inti=0;i<200;i++)

{

if(i!=this->Path.at(0))

{

this->UnvisitedVertices.push\_back(i);

}

}

}

voidAnt::MoveToVertix(intdestination,intdestinationIdx)

{

this->Path.push\_back(destination);

this->UnvisitedVertices.erase(this->UnvisitedVertices.begin()+destinationIdx);

}

voidAnt::ChooseDestination(vector<Probability>Probabilities)

{

doublelowerBound=0;

doublerandomValue=((double)rand()/(RAND\_MAX));

intdestination=0;

intdestinationIdx=0;

if(randomValue==1)

{

destination=UnvisitedVertices.at(UnvisitedVertices.size()-1);

destinationIdx=UnvisitedVertices.size()-1;

}

elseif(randomValue==0)

{

destination=UnvisitedVertices.at(0);

destinationIdx=0;

}

elseif(UnvisitedVertices.size()==1)

{

destination=UnvisitedVertices.at(0);

destinationIdx=0;

}

else

{

for(intj=0;j<Probabilities.size();j++)

{

if((Probabilities.at(j).chance>-1)&&(Probabilities.at(j).vertix==this->UnvisitedVertices.at(j)))

{

if((lowerBound<randomValue)&&(randomValue<=lowerBound+Probabilities.at(j).chance))

{

destination=UnvisitedVertices.at(j);

destinationIdx=j;

break;

}

else

{

lowerBound+=Probabilities.at(j).chance;

}

}

}

}

MoveToVertix(destination,destinationIdx);

}

doubleAnt::getL(intgraph[200][200])

{

intL=0;

for(inti=0;i<this->Path.size()-1;i++)

{

L+=graph[this->Path.at(i)][this->Path.at(i+1)];

}

returnL;

}

TSP.cpp

#include "Header.h"

#include "Ant.h"

void printList(Ant Ant);

bool CheckOrig(vector<int> Path)

{

for (int i = 0; i < Path.size() - 1; i++)

{

for (int j = i + 1; j < Path.size() - 1; j++)

{

if (Path.at(i) == Path.at(j))

{

return false;

}

}

}

return true;

}

void TSPAlgorithm(int graph[200][200], double pheromoneGraph[200][200], double pheromoneSumGraph[200][200])

{

int Alpha = 3;

int Beta = 2;

double Ro = 0.3;

double Lmin = GetLmin(graph);

vector<Ant> Ants;

vector<Probability> Probabilities;

int UsedStartingPoints[45];

for (int i = 0; i < 45; i++)

{

UsedStartingPoints[i] = -1;

}

for (int i = 0; i < 45; i++)

{

int StartingPoint = GenerateStartingPoint(UsedStartingPoints, i);

Ants.push\_back(Ant(StartingPoint));

}

int iteration = 0;

int count = 0;

while (iteration < 1000)

{

for (int i = 0; i < Ants.size(); i++)

{

Ants.at(i).Reset();

}

for (int i = 0; i < Ants.size(); i++)

{

while (Ants.at(i).UnvisitedVertices.size() != 0)

{

Probabilities.clear();

double StepTwo = 0;

for (int p = 0; p < Ants.at(i).UnvisitedVertices.size(); p++)

{

StepTwo += MoveProbabilityStepOne(Ants.at(i).Path.at(Ants.at(i).Path.size() - 1), Ants.at(i).UnvisitedVertices.at(p), pheromoneGraph[Ants.at(i).Path.at(Ants.at(i).Path.size() - 1)][Ants.at(i).UnvisitedVertices.at(p)], Alpha, Beta, graph);

}

for (int j = 0; j < Ants.at(i).UnvisitedVertices.size(); j++)

{

if (graph[Ants.at(i).Path.at(Ants.at(i).Path.size() - 1)][Ants.at(i).UnvisitedVertices.at(j)] != 0)

{

Probability prob;

prob.chance = MoveProbability(Ants.at(i).Path.at(Ants.at(i).Path.size() - 1), Ants.at(i).UnvisitedVertices.at(j), pheromoneGraph[Ants.at(i).Path.at(Ants.at(i).Path.size() - 1)][Ants.at(i).UnvisitedVertices.at(j)], Alpha, Beta, graph, StepTwo);

prob.vertix = Ants.at(i).UnvisitedVertices.at(j);

Probabilities.push\_back(prob);

}

else

{

Probability prob;

prob.chance = -1;

prob.vertix = Ants.at(i).UnvisitedVertices.at(j);

Probabilities.push\_back(prob);

}

}

Ants.at(i).ChooseDestination(Probabilities);

count++;

}

Ants.at(i).Path.push\_back(Ants.at(i).StartingPoint);

for (int m = 0; m < Ants.at(i).Path.size() - 1; m++)

{

pheromoneSumGraph[Ants.at(i).Path.at(m)][Ants.at(i).Path.at(m + 1)] += Lmin / Ants.at(i).getL(graph);

}

}

double temp;

for (int m = 0; m < 200; m++)

{

for (int n = 0; n < 200; n++)

{

if (m != n)

{

temp = pheromoneGraph[m][n];

pheromoneGraph[m][n] = (1 - Ro) \* temp + pheromoneSumGraph[m][n];

}

}

}

BuildPheromoneSumGraph(pheromoneSumGraph);

iteration++;

if ((iteration + 1) % 20 == 0)

{

int minIdx = 0;

int min = Ants.at(0).getL(graph);

int maxIdx = 0;

int max = Ants.at(0).getL(graph);

double SumL = 0;

for (int i = 1; i < Ants.size(); i++)

{

if (Ants.at(i).getL(graph) < min)

{

min = Ants.at(i).getL(graph);

minIdx = i;

}

if (Ants.at(i).getL(graph) > max)

{

max = Ants.at(i).getL(graph);

maxIdx = i;

}

SumL += Ants.at(i).getL(graph);

}

cout << Ants.at(minIdx).getL(graph) << " " << Ants.at(maxIdx).getL(graph) << " " << (SumL / Ants.size()) << endl;

}

}

}

int GenerateStartingPoint(int UsedStartingPoints[45], int M)

{

int StartingPoint = rand() % 200;

if (!CheckForStartingPoint(UsedStartingPoints, StartingPoint, M))

{

StartingPoint = GenerateStartingPoint(UsedStartingPoints, M);

}

return StartingPoint;

}

bool CheckForStartingPoint(int UsedStartingPoints[45], int StartingPoint, int M)

{

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

{

if (StartingPoint == UsedStartingPoints[i])

{

return false;

}

}

return true;

}

double MoveProbability(int curVertix, int destination, double pheromone, int alpha, int beta, int graph[200][200], double StepTwo)

{

double StepOne = MoveProbabilityStepOne(curVertix, destination, pheromone, alpha, beta, graph);

double probability = StepOne / StepTwo;

return probability;

}

double MoveProbabilityStepOne(int curVertix, int destination, double pheromone, int alpha, int beta, int graph[200][200])

{

if (graph[curVertix][destination] != 0)

{

double edge = graph[curVertix][destination];

double visibility = 1 / edge;

double StepOne = (pow(pheromone, alpha) \* pow(visibility, beta));

return StepOne;

}

else

{

return 0;

}

}

double GetLmin(int graph[200][200])

{

vector<int> Path;

int UnvisitedVertices[200];

for (int i = 0; i < 200; i++)

{

UnvisitedVertices[i] = -1;

}

double Lmin = 0;

int min = graph[0][1];

int newVertix = 1;

int oldVertix = 0;

UnvisitedVertices[oldVertix] = 1;

UnvisitedVertices[newVertix] = 1;

for (int i = 0; i < 200; i++)

{

for (int j = 0; j < 200; j++)

{

if ((graph[i][j] < min) && (graph[i][j] != 0))

{

min = graph[i][j];

UnvisitedVertices[oldVertix] = -1;

UnvisitedVertices[newVertix] = -1;

oldVertix = i;

newVertix = j;

UnvisitedVertices[oldVertix] = 1;

UnvisitedVertices[newVertix] = 1;

}

}

}

int first = oldVertix;

Path.push\_back(oldVertix);

Path.push\_back(newVertix);

while (Path.size() != 200)

{

int next = GetMin(graph, UnvisitedVertices, Path.back());

UnvisitedVertices[next] = 1;

Path.push\_back(next);

}

Path.push\_back(first);

bool check = CheckOrig(Path);

for (int i = 0; i < Path.size() - 1; i++)

{

Lmin += graph[Path.at(i)][Path.at(i + 1)];

}

return Lmin;

}

int GetMin(int graph[200][200], int UnvisitedVertices[200], int currVertix)

{

int min = 0;

int nextVertix = 0;

for (int i = 0; i < 200; i++)

{

if ((graph[currVertix][i] != 0) && (UnvisitedVertices[i] == -1))

{

min = graph[currVertix][i];

nextVertix = i;

UnvisitedVertices[i] = 1;

break;

}

}

for (int i = 0; i < 200; i++)

{

if ((graph[currVertix][i] != 0) && (graph[currVertix][i] < min) && (UnvisitedVertices[i] == -1))

{

min = graph[currVertix][i];

UnvisitedVertices[nextVertix] = -1;

nextVertix = i;

UnvisitedVertices[i] == 1;

}

}

return nextVertix;

}

void printList(Ant Ant)

{

cout << Ant.Path.size() << endl;

for (int i = 0; i < 200; i++)

{

cout << Ant.Path.at(i) << "->";

}

cout << Ant.Path.at(0);

cout << endl;

cout << endl;

}

void printArray(int arr[200])

{

for (int i = 0; i < 200; i++)

{

cout << arr[i] << "->";

}

cout << endl;

cout << endl;

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

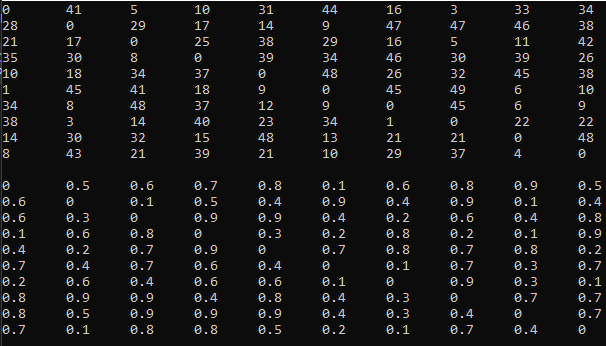
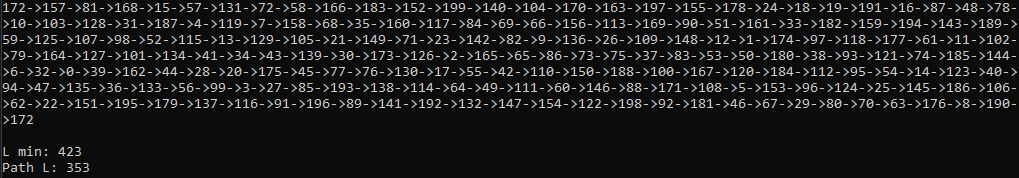


Рисунок 3.1 – Частина початкових параметрів: відстані між вершинами та феромон на ребрах них

Рисунок 3.2 – Кінцевий шлях після 20 ітерацій, отримане Lmin та довжина шляху L

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| Кількість ітерацій | Значення цільової функції |
| 20 | 378,800 |
| 40 | 351,400 |
| 60 | 337,733 |
| 80 | 338,756 |
| 100 | 342,622 |
| 120 | 342,333 |
| 140 | 340,511 |
| 160 | 338,044 |
| 180 | 337,089 |
| 200 | 340,689 |
| 220 | 337,644 |
| 240 | 338,756 |
| 260 | 339,178 |
| 280 | 337,667 |
| 300 | 337,089 |
| 320 | 338,222 |
| 340 | 340,044 |
| 360 | 339,378 |
| 380 | 337,667 |
| 400 | 337,756 |
| 420 | 338,267 |
| 440 | 341,289 |
| 460 | 338,022 |
| 480 | 341,533 |
| 500 | 337,600 |
| 520 | 340,444 |
| 540 | 342,778 |
| 560 | 337,533 |
| 580 | 338,067 |
| 600 | 338,000 |
| 620 | 338,778 |
| 640 | 339,267 |
| 660 | 339,178 |
| 680 | 338,933 |
| 700 | 341,267 |
| 720 | 338,089 |
| 740 | 338,089 |
| 760 | 338,089 |
| 780 | 337,889 |
| 800 | 339,867 |
| 820 | 347,044 |
| 840 | 343,311 |
| 860 | 339,378 |
| 880 | 339,333 |
| 900 | 337,600 |
| 920 | 342,733 |
| 940 | 339,000 |
| 960 | 337,600 |
| 980 | 338,089 |
| 1000 | 338,089 |

### Графіки залежності розв'язку від числа ітерацій

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

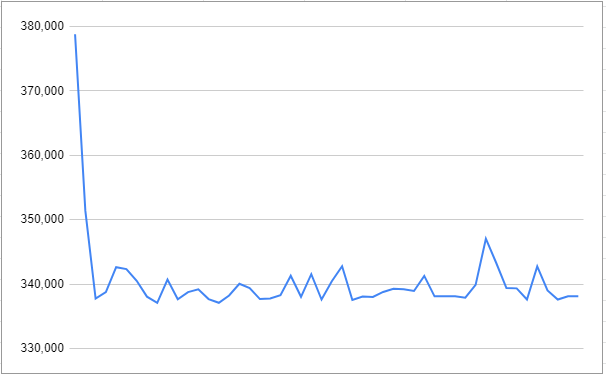


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи було розглянуто мурашиний алгоритм на прикладі рішення задачі комівояжера. Наведена програмна реалізація завдання на мові С++ та проведено аналіз отриманих результатів. Надана таблиця залежності результату від кількості ітерацій та відповідний графік.

Як видно з таблиці та графіку, результат стає кращим з більшою кількістю ітерацій, скорочуючи необхідний для обходу графа шлях, але скорочення не є необмеженим, бо з певного моменту довжина шляху або зменшується дуже повільно або взагалі досягає майже сталого стану.

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%.