**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 № 3 з дисципліни

«Проектування алгоритмів»

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

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-02 Василенко Павло Олександрович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Вєчєрковська А.С.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 9](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 9](#_Toc51260920)

[3.1.1 Вихідний код 9](#_Toc51260921)

[3.1.2 Приклади роботи 9](#_Toc51260922)

[3.2 Тестування алгоритму 10](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 10](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 10](#_Toc51260925)

[Висновок 11](#_Toc51260926)

[Критерії оцінювання 12](#_Toc51260927)

# Мета лабораторної роботи

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

# Завдання

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

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого 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** |
| 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). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

Laba3.cpp

#include <iostream>

#include <vector>

#include <string>

#include "Komivoyazher.h"

#include "AntColony.h"

using namespace std;

int main()

{

srand(time(0));

Komivoyazher komivoyazher(100);

AntColony antColony(komivoyazher.distanceMatrix, 100);

antColony.solve();

}

Komivoyazher.h

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Komivoyazher

{

public:

float\*\* distanceMatrix;

int numberOfCities;

Komivoyazher(int numOfCities);

void displayDistanceMatrix();

};

Komivoyazher.cpp

#include "Komivoyazher.h"

Komivoyazher::Komivoyazher(int numOfCities)

{

this->numberOfCities = numOfCities;

this->distanceMatrix = new float\* [numOfCities];

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

{

this->distanceMatrix[i] = new float[numOfCities];

}

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

{

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

{

distanceMatrix[i][j] = 0;

}

}

//for (int i = 0; i < numOfCities; i++)

//{

// for (int j = 0; j < numOfCities; j++)

// {

// if (i == j) distanceMatrix[i][j] = 0;

// else if (distanceMatrix[i][j] != 0) continue;

// else {

// int num = rand()%40 + 5;

// distanceMatrix[i][j] = distanceMatrix[j][i] = num;

// }

// }

//}

//READ FROM FILE

ifstream file;

file.open("data.txt");

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

{

string curDataLine;

getline(file, curDataLine);

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

{

int value = stoi(curDataLine.substr(0, curDataLine.find(' ')));

curDataLine.erase(0, curDataLine.find(' ') + 1);

distanceMatrix[i][j] = value;

}

}

file.close();

}

void Komivoyazher::displayDistanceMatrix()

{

for (int i = 0; i < this->numberOfCities; i++)

{

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

{

cout << distanceMatrix[i][j] << '\t';

}

cout << endl;

}

}

Ant.h

#pragma once

class Ant

{

public:

int trailSize;

int\* trail;

bool\* visited;

Ant(int tourLength);

void visitCity(int currentIndex, int city);

bool isVisited(int i);

void forget();

float trailLength(float\*\* graph);

};

Ant.cpp

#include "Ant.h"

Ant::Ant(int tourLength)

{

trailSize = tourLength;

trail = new int[tourLength];

visited = new bool[tourLength];

}

void Ant::visitCity(int currentIndex, int city)

{

trail[currentIndex + 1] = city;

visited[city] = true;

}

bool Ant::isVisited(int i)

{

return visited[i];

}

void Ant::forget() {

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

visited[i] = false;

}

}

float Ant::trailLength(float\*\* graph)

{

float length = graph[trail[trailSize - 1]][trail[0]];

for (int i = 0; i < trailSize - 1; i++)

{

length += graph[trail[i]][trail[i + 1]];

}

return length;

}

AntColony.h

#pragma once

#include <vector>

#include <cmath>

#include "Ant.h"

#include <iostream>

using namespace std;

class AntColony

{

int alpha = 2;

int beta = 4;

float evaporation = 0.4;

int numberOfAnts = 30;

float randomFactor = 0.01;

int maxIterations = 1000;

int numberOfCities;

double Lmin = 0;

float\*\* graph;

float\*\* sightMatrix;

float\*\* trails;

vector<Ant> ants;

double\* probabilities;

int currentIndex;

int\* bestTourOrder;

float bestTourLength;

void getLMin();

bool CheckInChecked(int a, vector<int> checked);

void setAnts();

void setStartTrails();

void setSightMatrix();

void moveAnts();

void calculateProbabilities(Ant ant);

int selectCity(Ant ant);

void updateTrails();

void updateBestSolution();

public:

AntColony(float\*\* generatedMatrix, int size);

int\* solve();

};

AntColony.cpp

#include "AntColony.h"

AntColony::AntColony(float\*\* generatedMatrix, int size)

{

graph = new float\* [size];

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

{

graph[i] = new float[size];

}

graph = generatedMatrix;

trails = new float\* [size];

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

{

trails[i] = new float[size];

}

sightMatrix = new float\* [size];

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

{

sightMatrix[i] = new float[size];

}

probabilities = new double [size];

numberOfCities = size;

bestTourOrder = new int[size];

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

{

bestTourOrder[i] = 0;

}

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

{

ants.push\_back(Ant(numberOfCities));

}

}

void AntColony::setAnts()

{

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

{

ants[i].forget();

int cityNum = rand() % numberOfCities;

ants[i].visitCity(-1, cityNum);

}

currentIndex = 0;

}

int\* AntColony::solve()

{

getLMin();

setStartTrails();

setSightMatrix();

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

{

setAnts();

moveAnts();

updateTrails();

updateBestSolution();

if (i % 20 == 0) {

cout << "Iteration: " << i << endl;

cout << "Length of the best path founded: " << bestTourLength << endl;

cout << "Order of best solution: ";

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

{

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

}

cout << endl;

}

}

cout <<"LMin" << Lmin << endl;

return bestTourOrder;

}

void AntColony::setStartTrails()

{

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

{

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

{

if (i == j) trails[i][j] = 0;

else {

trails[i][j] = 0.2; //start number of feromon

}

}

}

}

void AntColony::setSightMatrix()

{

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

{

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

{

if (i != j) {

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

}

else {

sightMatrix[i][j] = 0;

}

}

}

}

void AntColony::moveAnts()

{

for (int i=currentIndex; i < numberOfCities-1; i++, currentIndex++)

{

for(auto ant: ants)

{

ant.visitCity(i, selectCity(ant));

}

}

}

int AntColony::selectCity(Ant ant) {

float random = static\_cast <float>(rand()%1000)/static\_cast<float>(1000);

if (random < randomFactor) {

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

{

if (!ant.isVisited(i)) {

return i;

}

}

}

calculateProbabilities(ant);

float counter = 0;

int index = 0;

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

{

counter += probabilities[i];

if (counter >= random) {

index= i;

break;

}

}

return index;

}

void AntColony::calculateProbabilities(Ant ant)

{

int i = ant.trail[currentIndex];

double feromone = 0.0;

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

{

if (!ant.isVisited(j)) {

feromone += pow(trails[i][j], alpha) \* pow(sightMatrix[i][j], beta);

}

}

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

{

if (ant.isVisited(j)) {

probabilities[j] = 0;

}

else {

double current = pow(trails[i][j], alpha) \* pow(sightMatrix[i][j], beta);

probabilities[j] = current / feromone;

}

}

}

void AntColony::updateTrails()

{

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

{

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

{

trails[i][j] \*= (1 - evaporation); //feromon disappear

}

}

for (auto ant : ants) {

float contribution = (Lmin / ant.trailLength(graph));

for (int i = 0; i < numberOfCities - 1; i++) {

trails[ant.trail[i]][ant.trail[i + 1]] += contribution;

}

trails[ant.trail[numberOfCities - 1]][ant.trail[0]] += contribution;

}

}

void AntColony::updateBestSolution()

{

if (bestTourOrder[0]==0 && bestTourOrder[1]==0) {

bestTourOrder = ants[0].trail;

bestTourLength = ants[0].trailLength(graph);

}

for (auto ant : ants) {

if (ant.trailLength(graph) < bestTourLength) {

bestTourLength = ant.trailLength(graph);

bestTourOrder = ant.trail;

}

}

}

void AntColony::getLMin()

{

int Lmin = 0;

vector<int> checked;

int i = 0;

while (checked.size() < numberOfCities)

{

if (checked.size() + 1 == numberOfCities)

{

Lmin += graph[i][0];

break;

}

int min = 51;

int count = 0;

checked.push\_back(i);

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

{

if (min > graph[i][j] && CheckInChecked(j, checked))

{

min = graph[i][j];

count = j;

}

}

i = count;

Lmin += min;

}

this->Lmin = Lmin;

}

bool AntColony::CheckInChecked(int a, vector<int> checked)

{

for (int n = 0; n < checked.size(); n++)

{

if (a == checked[n])

{

return false;

}

}

return true;

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

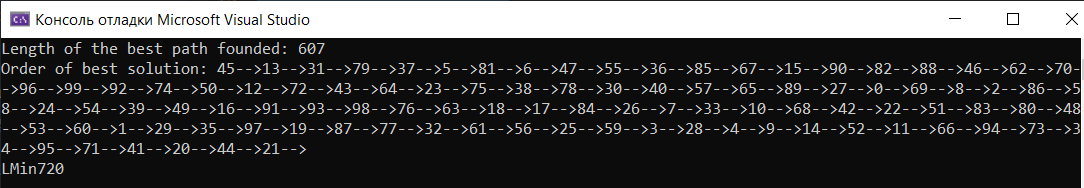


Рисунок 3.1 –

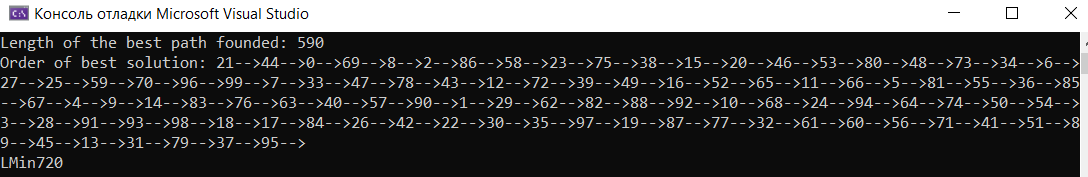


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| Ітерація | Довжина кращого шляху |
| 20 | 758 |
| 40 | 600 |
| 60 | 600 |
| 80 | 584 |
| 100 | 580 |
| 120 | 580 |
| 140 | 580 |
| 160 | 580 |
| 180 | 580 |
| 200 | 580 |
| 220 | 580 |
| 240 | 580 |
| 260 | 580 |
| 280 | 580 |
| 300 | 580 |
| 320 | 580 |
| 340 | 580 |
| 360 | 580 |
| 380 | 580 |
| 400 | 575 |
| 420 | 575 |
| 440 | 575 |
| 460 | 575 |
| 480 | 575 |
| 500 | 575 |
| 520 | 575 |
| 540 | 575 |
| 560 | 575 |
| 580 | 575 |
| 600 | 575 |
| 620 | 575 |
| 640 | 575 |
| 660 | 575 |
| 680 | 575 |
| 700 | 575 |
| 720 | 575 |
| 740 | 575 |
| 760 | 575 |
| 780 | 575 |
| 800 | 575 |
| 820 | 575 |
| 840 | 575 |
| 860 | 575 |
| 880 | 575 |
| 900 | 575 |
| 920 | 575 |
| 940 | 575 |
| 960 | 575 |
| 980 | 575 |
| 1000 | 575 |

### Графіки залежності розв'язку від числа ітерацій

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

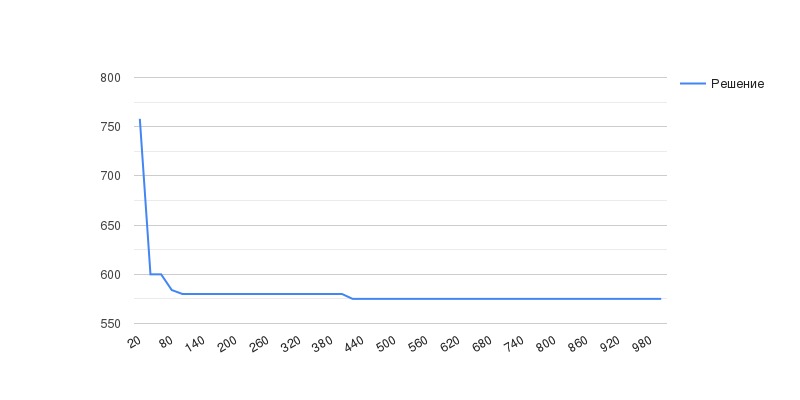


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи я вивчив роботу мурашиного алгоритму та виконав його програмну реалізацію. В мою задачу входило зокрема розв’язати задачу комівояжера мурашиним алгоритмом. Провів також дослідження алгоритму за кількістю ітерацій та побудував графіки.

Критерії оцінювання

При здачі лабораторної роботи до 5.11.2021 включно максимальний бал дорівнює – 5. Після 5.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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