**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Гуськов Кирило Михайлович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 10](#_Toc52291750)

[3.1 Покроковий алгоритм 10](#_Toc52291751)

[3.2 Програмна реалізація алгоритму 10](#_Toc52291752)

[3.2.1 Вихідний код 10](#_Toc52291753)

[3.2.2 Приклади роботи 10](#_Toc52291754)

[3.3 Тестування алгоритму 11](#_Toc52291755)

[Висновок 12](#_Toc52291756)

[Критерії оцінювання 13](#_Toc52291757)

# Мета лабораторної роботи

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

# Завдання

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем 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 сумарна вага не перевищувала задану, а сумарна цінність була максимальною.  Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика. |
| 2 | **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.  **Розглядається симетричний, асиметричний та змішаний варіанти.**  В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.  У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.  У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху.  Застосування:   * доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); * доставка води; * моніторинг об'єктів; * поповнення банкоматів готівкою; * збір співробітників для доставки вахтовим методом. |
| 3 | **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.  Застосування:   * розкладу для освітніх установ; * розкладу в спорті; * планування зустрічей, зборів, інтерв'ю; * розклади транспорту, в тому числі - авіатранспорту; * розкладу для комунальних служб; |
| 4 | **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.  Задача вершинного покриттяполягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).  На вході: Граф G = (V, E).  Результат: множина C ⊆ V - найменше вершинне покриття графа G.    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 не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.  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. |

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

|  |  |
| --- | --- |
| **№** | **Алгоритми і досліджувані параметри** |
| 1 | **Генетичний алгоритм:**   * оператор схрещування (мінімум 3); * мутація (мінімум 2); * оператор локального покращення (мінімум 2). |
| 2 | **Мурашиний алгоритм**:   * α; * β; * ρ; * Lmin; * кількість мурах М і їх типи (елітні, тощо…); * маршрути з однієї чи різних вершин. |
| 3 | **Бджолиний алгоритм:**   * 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 |

# Виконання

## Покроковий алгоритм

Ініцалізація графу, де з випадковим шансом 50/50 ребро i→j буде або дорівнювати ребру j→i, або ні.

Ініціалізація початкового графу феромонів, де кожне ребро має 0.1 феромон.

ПОКИ 20 ітерація або менша

ДЛЯ i ВІД 0 ДО Кількості мурах М З КРИКОМ 1

ПОКИ кількість непройдених вершин != 0

ДЛЯ j ВІД 0 ДО Кількості непройдених вершин З КРИКОМ 1

Рахуємо імовірність переходу до вершини за формулою

після чого обираємо вершину за випадково згенерованим числом в діапазоні від 0 до 1

КІНЕЦЬ ДЛЯ

КІНЕЦЬ ПОКИ

КІНЕЦЬ ДЛЯ

Оновлюємо феромони для кожного ребра за формулою

КІНЕЦЬ ПОКИ

КІНЕЦЬ

## Програмна реалізація алгоритму

### Вихідний код

PA’Lab5.cpp

#include<iostream>

#include<ctime>

#include"Header.h"

usingnamespacestd;

intmain()

{

srand(time(NULL));

intgraph[300][300];

autopheromoneGraph=newdouble[300][300];

autopheromoneSumGraph=newdouble[300][300];

BuildGraph(graph);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

/\*

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

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

}

cout << endl;

}

cout << endl;

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 10; j++)

{

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

}

cout << endl;

}\*/

TSPFindOptimal(graph,pheromoneGraph,pheromoneSumGraph);

}

voidBuildGraph(intgraph[300][300])

{

for(inti=0;i<300;i++)

{

for(intj=0;j<300;j++)

{

if(i==j)

{

graph[i][j]=0;

}

else

{

graph[i][j]=(rand()%146)+5;

}

}

}

for(inti=0;i<300;i++)

{

for(intj=i;j<300;j++)

{

if(i!=j)

{

intrandNum=rand()%2;

if(randNum==1)

{

graph[j][i]=graph[i][j];

}

}

}

}

}

voidBuildPheromoneGraph(doublepheromoneGraph[300][300])

{

for(inti=0;i<300;i++)

{

for(intj=0;j<300;j++)

{

if(i==j)

{

pheromoneGraph[i][j]=0;

}

else

{

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

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

}

}

}

}

voidBuildPheromoneSumGraph(doublepheromoneSumGraph[300][300])

{

for(inti=0;i<300;i++)

{

for(intj=0;j<300;j++)

{

pheromoneSumGraph[i][j]=0;

}

}

}

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 TSPFindOptimal(int graph[300][300], double pheromoneGraph[300][300], double pheromoneSumGraph[300][300])

{

int Alpha = 2;

int Beta = 2;

double Ro = 0.3;

double Lmin = GetLmin(graph);

int M = 45;

int FirstSP = 152;

int eliteGen = 0;

int count = 0;

/\*

cout << "Enter Alpha: ";

cin >> Alpha;

cout << "Enter Beta: ";

cin >> Beta;

cout << "Enter Ro: ";

cin >> Ro;

cout << "Enter Lmin(Suggested " << Lmin << "): ";

cin >> Lmin;

cout << "Enter number of Ants: ";

cin >> M;

//cout << "Enter starting point for the first Ant: ";

//cin >> FirstSP;

//cout << "Will first Ant be elite? ";

//cin >> eliteGen;

cout << TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, Beta, Ro, Lmin, M, FirstSP, eliteGen) << endl;\*/

auto arrAlpha = new double[10];

for (int i = 0; i < 10; i++)

{

arrAlpha[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, (i + 1), Beta, Ro, Lmin, M, FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

}

double min = arrAlpha[0];

for (int i = 0; i < 10; i++)

{

if (arrAlpha[i] < min)

{

min = arrAlpha[i];

Alpha = i;

}

}

auto arrBeta = new double[10];

for (int i = 0; i < 10; i++)

{

arrBeta[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, (i + 1), Ro, Lmin, M, FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

}

min = arrBeta[0];

for (int i = 0; i < 10; i++)

{

if (arrBeta[i] < min)

{

min = arrBeta[i];

Beta = i;

}

}

auto arrRo = new double[10];

double tempRo = 0;

for (int i = 0; i < 10; i++)

{

arrRo[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, Beta, tempRo, Lmin, M, FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

tempRo += 0.1;

}

min = arrRo[0];

for (int i = 0; i < 10; i++)

{

if (arrRo[i] < min)

{

min = arrRo[i];

Ro = (i / 10);

}

}

auto arrLmin = new double[10];

for (int i = 0; i < 10; i++)

{

if (i == 5)

{

arrLmin[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, Beta, Ro, Lmin, M, FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

}

else

{

arrLmin[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, Beta, Ro, (400 \* (i + 1)), M, FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

}

}

min = arrLmin[0];

for (int i = 0; i < 10; i++)

{

if (arrLmin[i] < min)

{

min = arrLmin[i];

if (i == 5)

{

break;

}

else

{

Lmin = (400 \* (i + 1));

}

}

}

auto arrM = new double[10];

for (int i = 0; i < 10; i++)

{

arrM[i] = TSPAlgorithm(graph, pheromoneGraph, pheromoneSumGraph, Alpha, Beta, tempRo, Lmin, (9 \* (i + 1)), FirstSP, eliteGen);

BuildPheromoneGraph(pheromoneGraph);

BuildPheromoneSumGraph(pheromoneSumGraph);

}

min = arrM[0];

for (int i = 0; i < 10; i++)

{

if (arrM[i] < min)

{

min = arrM[i];

M = (9 \* (i + 1));

}

}

cout << "Alpha: " << Alpha << endl;

cout << "Beta: " << Beta << endl;

cout << "Ro: " << Ro << endl;

cout << "Lmin: " << Lmin << endl;

cout << "M: " << M << endl;

cout << min;

}

double TSPAlgorithm(int graph[300][300], double pheromoneGraph[300][300], double pheromoneSumGraph[300][300], int alpha, int beta, double ro, double LMin, int m, int FirstSp, int EliteGen)

{

int Alpha = alpha;

int Beta = beta;

double Ro = ro;

double Lmin = LMin;

int M = m;

int FirstSP = FirstSp;

int eliteGen = EliteGen;

vector<Ant> Ants;

vector<Probability> Probabilities;

int\* UsedStartingPoints = new int[M];

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

{

UsedStartingPoints[i] = -1;

}

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

{

if (i == 0)

{

Ants.push\_back(Ant(FirstSP, eliteGen));

}

else

{

int StartingPoint = GenerateStartingPoint(UsedStartingPoints, i);

int eliteGenerator = rand() % 2;

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

}

}

int iteration = 0;

int count = 0;

while (iteration < 20)

{

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);

if (Ants.at(i).elite)

{

pheromoneSumGraph[Ants.at(i).Path.at(Ants.at(i).Path.size() - 2)][Ants.at(i).Path.at(Ants.at(i).Path.size() - 1)] += 2 \* (Lmin / Ants.at(i).getL(graph));

count++;

}

else

{

pheromoneSumGraph[Ants.at(i).Path.at(Ants.at(i).Path.size() - 2)][Ants.at(i).Path.at(Ants.at(i).Path.size() - 1)] += Lmin / Ants.at(i).getL(graph);

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);

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

{

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

{

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

minIdx = i;

}

}

cout << endl;

printList(Ants.at(minIdx));

cout << "L min: " << Lmin << endl;

cout << "Path L: " << Ants.at(minIdx).getL(graph) << endl;

}\*/

/\*if ((iteration + 1) % 20 == 0)

{

int maxIdx = 0;

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

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

{

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

{

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

maxIdx = i;

}

}

cout << endl;

printList(Ants.at(minIdx));

cout << "L min: " << Lmin << endl;

cout << "Path L: " << Ants.at(maxIdx).getL(graph) << endl;

}\*/

/\*if (iteration % 20 == 0)

{

double SumL = 0;

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

{

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

}

cout << endl;

printList(Ants.at(minIdx));

cout << "L min: " << Lmin << endl;

cout << SumL/Ants.size() << endl;

}\*/

/\*if (iteration % 20 == 0)

{

cout << endl;

printList(Ants.at(0));

cout << "L min: " << Lmin << endl;

cout << "Path L: " << Ants.at(0).getL(graph) << endl;

}\*/

}

double SumL = 0;

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

{

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

}

return SumL / Ants.size();

}

int GenerateStartingPoint(int UsedStartingPoints[], int M)

{

int StartingPoint = rand() % 300;

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

{

StartingPoint = GenerateStartingPoint(UsedStartingPoints, M);

}

return StartingPoint;

}

bool CheckForStartingPoint(int UsedStartingPoints[], 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[300][300], 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[300][300])

{

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[300][300])

{

vector<int> Path;

int UnvisitedVertices[300];

for (int i = 0; i < 300; 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 < 300; i++)

{

for (int j = 0; j < 300; 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() != 300)

{

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[300][300], int UnvisitedVertices[300], int currVertix)

{

int min = 0;

int nextVertix = 0;

for (int i = 0; i < 300; i++)

{

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

{

min = graph[currVertix][i];

nextVertix = i;

UnvisitedVertices[i] = 1;

break;

}

}

for (int i = 0; i < 300; 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 < 300; i++)

{

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

}

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

cout << endl;

cout << endl;

}

void printArray(int arr[300])

{

for (int i = 0; i < 300; i++)

{

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

}

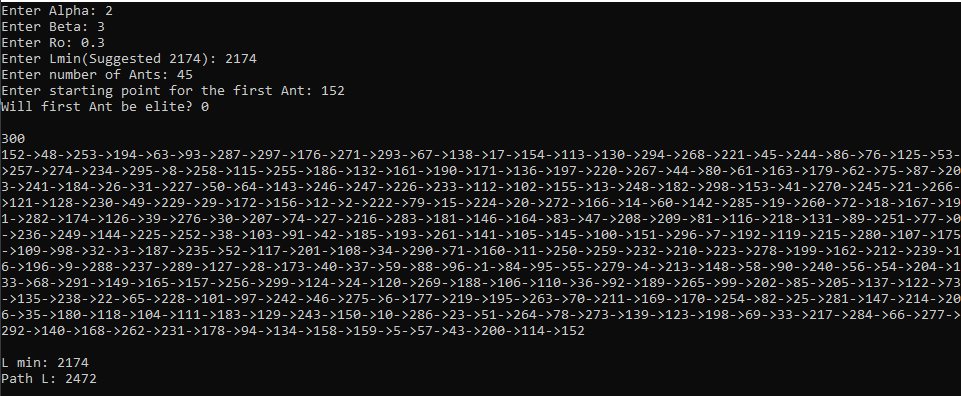
cout << endl;

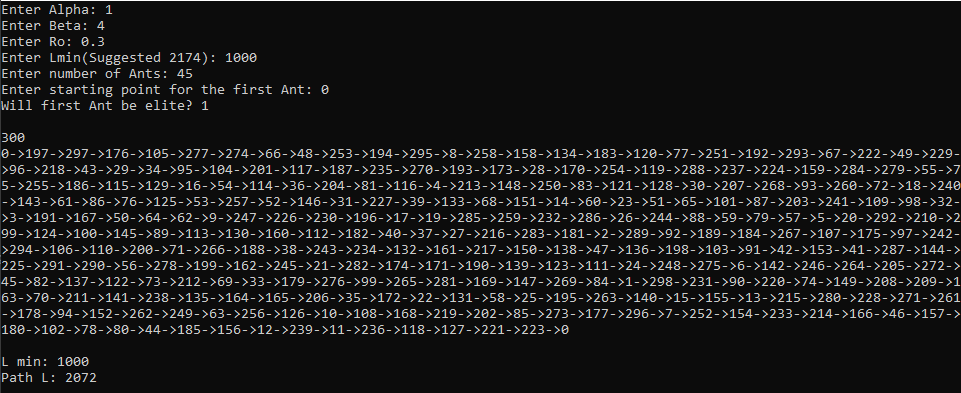
cout << endl;

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

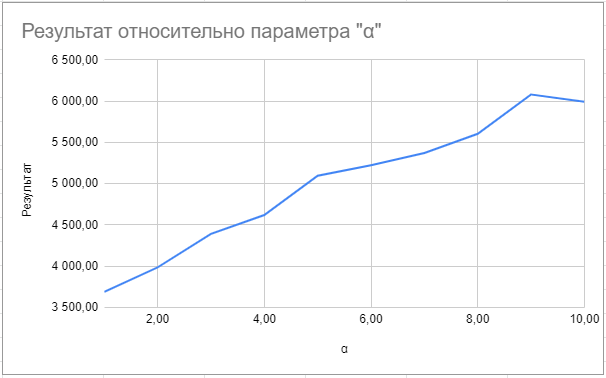
Рисунок 3.1 – Результат програми для перщої мурахи при стандартних параметрах

Рисунок 3.2 – Результат програми для перщої мурахи при оптимальних параметрах

## Тестування алгоритму

Як початкові значення використовуємо дані в лабораторній 4, а сама α = 3, β = 2, ρ = 0.3, M = 45. Lmin вираховуємо жадібним алгоритмом.

|  |  |
| --- | --- |
| α | Результат |
| 1,00 | 3 689,71 |
| 2,00 | 3 984,44 |
| 3,00 | 4392,6 |
| 4,00 | 4 619,69 |
| 5,00 | 5 096,71 |
| 6,00 | 5 224,56 |
| 7,00 | 5 372,44 |
| 8,00 | 5 604,71 |
| 9,00 | 6081,09 |
| 10,00 | 5 994,31 |

Рисунок 3.3.1 – Графік залежності середнього значення α при α від 1 до 10

|  |  |
| --- | --- |
| β | Результат |
| 1,00 | 7 602,47 |
| 2,00 | 3 968,67 |
| 3,00 | 2 986,62 |
| 4,00 | 2 646,24 |
| 5,00 | 2497,04 |
| 6,00 | 2 385,36 |
| 7,00 | 2 377,93 |
| 8,00 | 2 381,89 |
| 9,00 | 2 308,00 |
| 10,00 | 2 287,42 |

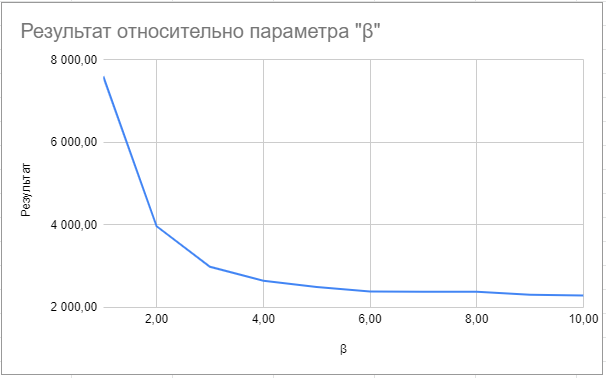
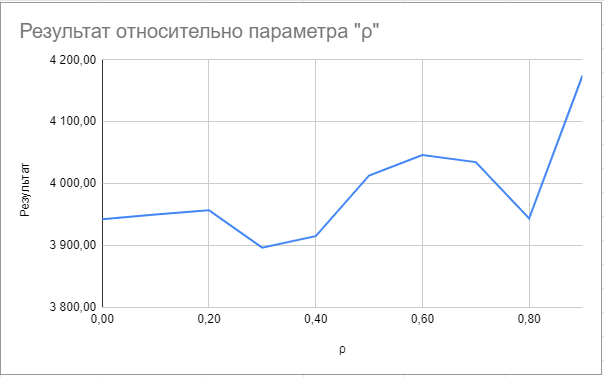


Рисунок 3.3.2 – Графік залежності середнього значення L при β від 1 до 10

|  |  |
| --- | --- |
| ρ | Результат |
| 0,00 | 3 942,42 |
| 0,10 | 3950,09 |
| 0,20 | 3 956,91 |
| 0,30 | 3 896,44 |
| 0,40 | 3 915,24 |
| 0,50 | 4 012,93 |
| 0,60 | 4 046,24 |
| 0,70 | 4 034,84 |
| 0,80 | 3 943,91 |
| 0,90 | 4 174,56 |

Рисунок 3.3.3 – Графік залежності середнього значення L при ρ від 0 до 0.9

|  |  |
| --- | --- |
| Lmin | Результат |
| 400,00 | 4 003,69 |
| 800,00 | 3 902,78 |
| 1 200,00 | 4 008,36 |
| 1 600,00 | 3 948,96 |
| 2 000,00 | 3 972,47 |
| 2 270,00 | 3 978,78 |
| 2 800,00 | 3946,4 |
| 3 200,00 | 4 080,29 |
| 3 600,00 | 3979,4 |
| 4 000,00 | 3904,02 |

Рисунок 3.3.4 – Графік залежності середнього значення L при Lmin від 400 до 4000 з кроком 400 за винятком вирахуваного Lmin.

|  |  |
| --- | --- |
| M | Результат |
| 9,00 | 3 953,33 |
| 18,00 | 3972,11 |
| 27,00 | 3 942,33 |
| 36,00 | 3 921,25 |
| 45,00 | 3 878,47 |
| 54,00 | 3981,7 |
| 63,00 | 3 996,25 |
| 72,00 | 3 987,75 |
| 81,00 | 3 957,44 |
| 90,00 | 3 896,79 |

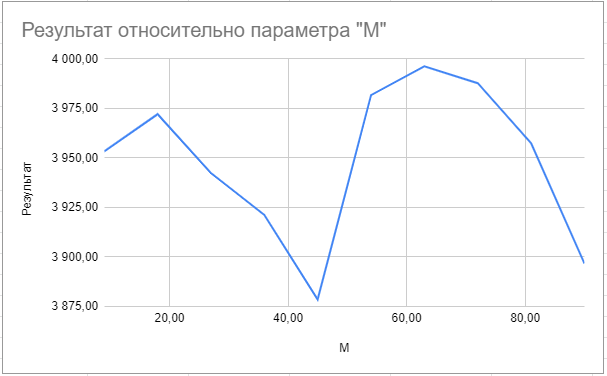


Рисунок 3.3.5 – Графік залежності середнього значення M при Lmin від 9 до 90 з кроком 9.

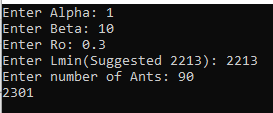


Рисунок 3.3.6 – Результат виконання програми при оптимальних значеннях з таблиць.

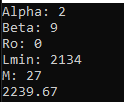


Рисунок 3.3.6 – Результат виконання програми при знаходженні оптимальних значень.

Також проведено аналіз результатів виключно для першого мурахи, де спочатку він був звичайним та починав з випадкової точки, а потім елітним та починав за вершини 0.

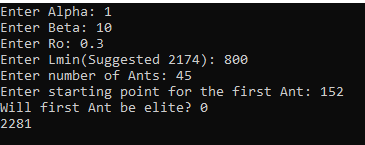


Рисунок 3.3.7 – Результат виконання програми при першому наборі параметрів.

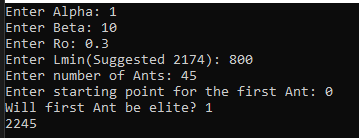


Рисунок 3.3.8 – Результат виконання програми при другому наборі параметрів.

Висновок

В рамках даної лабораторної роботи було розглянуто мурашиний алгоритм на прикладі рішення задачі комівояжера зі змішаним графом. Наведені покроковий алгоритм та програмна реалізація завдання на мові С++. Проведо аналіз результатів при різних вхідних даних та наведено відповідні таблиці та графіки.

Результат є найкращим при невеликих значеннях змінної α, яка відповідає за внесок кількості феромону на ребрі при визначенні шансу переходу. Це можна пов’язати з надто великим результатом після кількох ітерацій, що може погіршити результат для деяких мурах.

Результат є найкращим при великих значеннях змінної β, яка відповідає за внесок видимості вершини при визначенні шансу переходу. Це можна пов’язати з тим, що при великих значеннях β внесок малих значень видимості стане мінімальним, адже видимість лежить в діапазоні від 0 до 1, а рахується завдяки діленню 1 на довжину ребра до вершини. Отже чим ближчою до 1 буде видимість, тим більшою буде її внесок та отриманий результат шансу переходу.

Результат є найкращим при середніх значеннях змінної ρ, яка відповідає за випаровування феромону. Це можна пов’язати з тим, що таким чином залишені неоптимальні феромони не стануть подовгу впливати на розрахунок, адже будуть випаровуватися, а оптимальні не будуть випаровуватися надто швидко.

Результат є найкращим при більш низьких значеннях змінної Lmin, яка відповідає за залишений феромон, адже в такому випадку критерій буде вище, а отже неоптимальні шляхи будуть залишати невелику кількість феромону.

Результат є найкращим при більш великих значеннях змінної M, яка відповідає за кількість мурах, адже таким чином з більшою кількістю ітерацій на оптимальних шляхаха буде залишатися все більше феромону, а отже покращуватися шанс обрати їх при переході.

Також бачимо, що при наданні елітного статусу мурасі його результат покращується, бо він залишає більше феромону на оптимальному для себе шляху, а при створенні на оптимальній початковій точці він одразу має як мінімум оптимальний початок.

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%.