**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)

*Гавриленко Я.С*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О*

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** |
| 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). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

#region prepare

using System;

using System.Collections.Generic;

using System.Linq;

int iterCount = 100,

antCount = 30,

citiesCount = 100,

Lmin,

antPosition = 0,

newAntPosition;

int[] Lk = new int[antCount];

var positionRandomizer = new Random();

var visited = new HashSet<int>();

var distances = getDistances(citiesCount);

var pheromone = new double[citiesCount, citiesCount];

for (int i = 0; i < pheromone.GetLength(0); i++)

for (int j = 0; j < pheromone.GetLength(1); j++)

pheromone[i, j] = 1;

Lmin = greedyLength(0, distances);

var memory = new bool[antCount, citiesCount, citiesCount];

int[] curShortestPheromonePath;

int Lpr;

#endregion

for (int i = 1; i <= iterCount; i++)

{

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

{

antPosition = positionRandomizer.Next(0, citiesCount); // generate an initial position for an ant

visited.Add(antPosition); // add initial state as visited

while (visited.Count != citiesCount) // while all cities haven't been visited

{

newAntPosition = move(distances, antPosition); // calculate a position, where the ant will move

Lk[j] += distances[antPosition, newAntPosition]; // add the move distance to the ant's already traveled distance

memory[j, antPosition, newAntPosition] = true; // remember, that this ant j has visited the pass from [antPosition] to [newAntPosition]

antPosition = newAntPosition; // move the ant

visited.Add(antPosition); // set the new position as already seen

}

visited.Clear(); // clear the set for the new ant

}

update(pheromone); // update the level of pheromones on the map

curShortestPheromonePath = getShortestPheromonePath(pheromone);

Lpr = getLpr(curShortestPheromonePath);

if (Lpr < Lmin)

Lmin = Lpr;

visited.Clear();

}

Console.WriteLine($"Lmin = {Lmin}");

int getLpr(int[] citiesMap)

{

int num = 0;

for (int i = 0; i < (citiesMap.Length - 1); i++)

{

num += distances[citiesMap[i], citiesMap[i + 1]];

}

return num;

}

int[] getShortestPheromonePath(double[,] pheromone)

{

var resultPath = new Dictionary<int, double>(); // index / pheromone

int resultLength = 0,

curNode = 0;

resultPath.Add(0, int.MaxValue);

while (resultPath.Count < pheromone.GetLength(0))

{

resultPath = resultPath

.Concat(

Enumerable.Range(0, pheromone.GetUpperBound(1) + 1)

.Select(i => KeyValuePair.Create(i, pheromone[curNode, i]))

.OrderByDescending(n => resultPath.ContainsKey(n.Key) ? .0 : n.Value))

.Take(++resultLength)

.ToDictionary(n => n.Key, n => n.Value);

curNode = resultPath.Last().Key;

}

return resultPath.Keys.ToArray();

}

void update(double[,] pheromone)

{

double p = 0.4;

for (int i = 0; i < pheromone.GetLength(0); i++)

{

for (int j = 0; j < pheromone.GetLength(1); j++)

{

pheromone[i, j] \*= p;

pheromone[i, j] += calculateDeltaPheromone(i, j);

}

pheromone[i, i] = 1;

}

}

double calculateDeltaPheromone(int from, int to)

{

double num = 0.0;

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

{

bool flag = memory[i, from, to];

num = flag ? (num + (((double)Lmin) / ((double)Lk[i]))) : (num + 0.0);

}

return num;

}

int move(byte[,] distances, int prevPos)

{

int alpha = 2,

beta = 4;

var movingRandomizer = new Random();

byte[] availablePasses =

Enumerable.Range(0, distances.GetUpperBound(1) + 1)

.Select(i => distances[prevPos, i])

.ToArray();

double fullProbability = 0;

double pherToDistRelation;

double sum = 0;

for (int i = 0; i < availablePasses.Length; i++)

{

if (!visited.Contains(i))

{

pherToDistRelation = Math.Pow(pheromone[prevPos, i], alpha) \* Math.Pow((double)1 / availablePasses[i], beta);

fullProbability += pherToDistRelation;

}

}

double[] availableProbabilities = availablePasses.Select((n, index) =>

{

if (visited.Contains(index))

return 0;

pherToDistRelation = Math.Pow(pheromone[prevPos, index], alpha) \* Math.Pow((double)1 / n, beta);

double prob = pherToDistRelation / fullProbability;

sum += prob;

return prob;

}).ToArray();

double sum1 = 0;

double randomNum = movingRandomizer.NextDouble();

for (int i = 0; i < availableProbabilities.Length; i++)

{

sum1 += availableProbabilities[i];

if (sum1 >= randomNum) return i;

}

throw new Exception("wut");

}

int greedyLength(int startingIndex, byte[,] graph)

{

int length = 0;

int minLengthIndex = 0;

HashSet<int> visited = new HashSet<int>() { startingIndex };

while (visited.Count != graph.GetLength(0))

{

for (int i = 0; i < graph.GetLength(1); i++)

{

if (graph[startingIndex, i] < graph[startingIndex, minLengthIndex] && !visited.Contains(i))

minLengthIndex = i;

}

visited.Add(minLengthIndex);

length += graph[startingIndex, minLengthIndex];

startingIndex = minLengthIndex;

}

return length;

}

byte[,] getDistances(int length)

{

var r = new Random();

var distances = new byte[length, length];

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

{

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

{

distances[i, j] = (byte)r.Next(5, 51);

distances[j, i] = distances[i, j];

}

distances[i, i] = byte.MaxValue;

}

return distances;

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

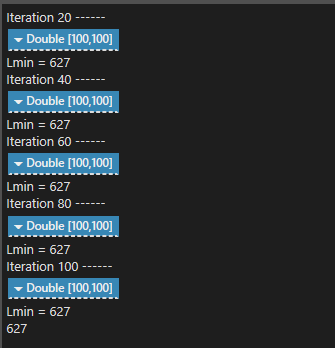


Рисунок 3.1 –

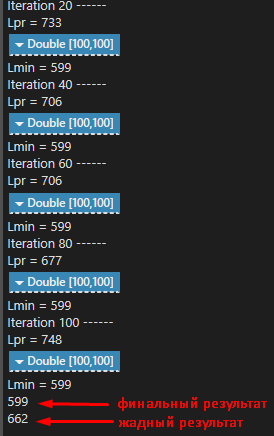


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| **Номер ітерації** | **Значення функції** |
| 20 | 742 |
| 60 | 731 |
| 80 | 724 |
| 100 | 724 |
| 120 | 724 |
| 140 | 724 |
| 160 | 724 |
| 180 | 724 |
| 200 | 724 |
| 220 | 724 |
| 240 | 724 |
| 260 | 724 |
| 280 | 724 |
| 300 | 724 |
| 320 | 724 |
| 340 | 724 |
| 360 | 724 |
| 380 | 724 |
| 400 | 724 |
| 420 | 724 |
| 440 | 724 |
| 460 | 724 |
| 480 | 724 |
| 500 | 724 |
| 520 | 724 |
| 540 | 724 |
| 560 | 724 |
| 580 | 724 |
| 600 | 724 |
| 620 | 724 |
| 640 | 724 |
| 660 | 724 |
| 680 | 724 |
| 700 | 724 |
| 720 | 724 |
| 740 | 724 |
| 760 | 724 |
| 780 | 724 |
| 800 | 724 |
| 820 | 724 |
| 840 | 724 |
| 860 | 724 |
| 880 | 724 |
| 900 | 724 |
| 920 | 724 |
| 940 | 724 |
| 960 | 724 |
| 980 | 724 |
| 1000 | 724 |

### 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 лабораторної роботи ми вивчили особливості роботи алгоритму мурав’їної колонії (Ant colony algorithm), вирішення задачі комівояжера за допомогою даного алгортиму, реалізували цю систему програмно та протестували дієздатність на прикладі випадково згенерованого графу

Github repository : https://github.com/yan14171/APLAB2.git

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%.