**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.2**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01, Черпак Андрій Вадимович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2020

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.

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** |
| 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 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. |

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

|  |  |
| --- | --- |
| **№** | **Алгоритми і досліджувані параметри** |
| 2 | **Мурашиний алгоритм**:   * α; * β; * ρ; * Lmin; * кількість мурах М і їх типи (елітні, тощо…); * маршрути з однієї чи різних вершин. |

Таблиця 2.3 – Варіанти задач і алгоритмів

|  |  |
| --- | --- |
| **№** | **Задачі і алгоритми** |
| 29 | Задача комівояжера (симетрична мережа) + Мурашиний алгоритм |

# Виконання

## Покроковий алгоритм

Спочатку зчитаємо з файлу вхідні дані і сформуємо матриці відстаней та феромонів. Далі в циклі на 1000 ітерацій створюємо 20 мурах та «запускаємо» їх шукати маршрут з різних точок. Кожна мураха має знайти гамільтонів цикл якнайменшої довжини. Критерієм вибору наступної вершини слугують відстань та досвід інших мурах, який виражається феромоном. Подальша поведінка залежить від типу мурахи: дикі ходять рандомно, елітні – по найкращому шляху, звичайні ж обирають випадкову вершину, але ймовірність вибору залежить від відстані та феромону. Після проходження шляху всіма мурахами оновлюємо матрицю феромонів, враховуючи випаровування та додавання нового феромону. Кількість феромону, залишеного мурахою, залежить від якості знайденого шляху. Елітні мурахи залишають вдвічі більше сліду. Після цього можна оновити параметр Lmin, а також найкращий маршрут, якщо такий зберігається окремо.

Після виконання 1000 ітерацій знову аналізуємо матрицю феромонів і визначаємо найкращий маршрут, базуючись на «досвіді» мурах, якщо не зберігали мінімальний шлях під час роботи програми. Виводимо маршрут і його довжину.

## Програмна реалізація алгоритму

### Вихідний код

public void ACO(int numberOfIterations)  
{  
 for (int ictr = 0; ictr < numberOfIterations; ictr++)  
 {  
 List<Ant> ants = new();  
 while (ants.Count<M)  
 {  
 int p = rand.Next(D.Length);  
 ants.Add(new Ant(D, T, p));  
 }  
 updateT(ants);  
if ((ictr + 1) % 20 == 0)  
 {  
Console.WriteLine("\r"+ictr+" : "+bestL);  
}  
 }

getResult();

}

public void getResult()  
{  
 string path = $"[{bestPath[0]}";  
 for(int i = 1; i<bestPath.Count; i++)  
 {  
 path += $" ==> {bestPath[i]}";  
 }  
 path += $"], {bestL}";  
 Console.WriteLine("\r"+path);  
}

private void updateT(List<Ant> ants)  
{  
 for (int i = 0; i<T.Length; i++) for (int j = 0; j < T[i].Length; j++) T[i][j] \*= 1 - p;  
 foreach (Ant ant in ants)  
 {  
 if (ant.Lk < bestL)  
 {  
 bestL = ant.Lk;  
 bestPath = ant.path;  
 }  
 for (int i = 1; i < ant.path.Count; i++)  
 {  
 T[ant.path[i - 1]][ant.path[i]] += (double)Lmin / ant.Lk;  
 T[ant.path[i]][ant.path[i-1]] += (double)Lmin / ant.Lk;  
 }  
 }  
}

public Ant(int[][] D, double[][] T, int startPoint)  
{  
 Lk = 0;  
 int current = startPoint;  
 path = new();  
 path.Add(current);  
 visited = new int[D.Length];  
 visited[current] = 1;  
 while (path.Count < D.Length)  
 {  
 int next = chooseNext(D[current], T[current], visited);  
 path.Add(next);  
 Lk += D[current][next];  
 visited[next] = 1;  
 current = next;  
 }  
 Lk += D[current][startPoint];  
 path.Add(startPoint);  
}

private static int chooseNext(int[] D, double[] T, int[] visited)  
{  
 List<KeyValuePair<int, double>> \_possible = new List<KeyValuePair<int, double>>();  
 double denominator = 0;  
 for (int i = 0; i < D.Length; i++)  
 {  
 if (visited[i] == 0)  
 {  
 double numerator = Math.Pow(T[i], a) \* Math.Pow(1.0 / D[i], b);  
 if (numerator < Math.Pow(10, -15)) numerator = Math.Pow(10, -15);  
 \_possible.Add(new KeyValuePair<int, double>(i, numerator));  
 denominator += numerator;  
 }  
 }  
 double test = 0;  
 List<KeyValuePair<int, double>> possible = new List<KeyValuePair<int, double>>();  
 for (int i = 0; i < \_possible.Count; i++)  
 {  
 test += \_possible[i].Value / denominator;  
 possible.Add(new KeyValuePair<int, double>(\_possible[i].Key, \_possible[i].Value/denominator));  
 }  
 if (Math.Round(test, 8)!=1) Console.WriteLine($"{test}!=1");  
 double choise = rand.NextDouble();  
 foreach (var vertice in possible)  
 {  
 if (choise < vertice.Value) return vertice.Key;  
 choise -= vertice.Value;  
 }  
 return -1;  
}

public static void Main()  
{  
MatrixFactory mf = new MatrixFactory(@"file2.csv");  
 int[][] D = mf.getD();  
 double[][] T = mf.getT();  
 ACO aco = new(D, T);  
 Console.WriteLine($"Lmin = {aco.Lmin}");  
 aco.Go(1000);  
}

public int[][] getD()  
{  
 using (StreamReader sr = new StreamReader(filename, System.Text.Encoding.Default))  
 {  
 graphSize = Int32.Parse(sr.ReadLine());  
 int[][] D = new int[graphSize][];  
 for (int i = 0; i < graphSize; i++)  
 {  
 D[i] = new int[graphSize];  
 String[] strDistances = sr.ReadLine().Split(new[] {','}, StringSplitOptions.**RemoveEmptyEntries**);  
 for (int j = 0; j < graphSize; j++) D[i][j] = Int32.Parse(strDistances[j]);  
 }  
 return D;  
 }  
}  
  
public double[][] getT()  
{  
 double[][] T = new double[graphSize][];  
 Random rand = new();  
 for (int i = 0; i < graphSize; i++)  
 {  
 T[i] = new double[graphSize];  
 for (int j = 0; j < graphSize; j++)  
 {  
 if (i == j) T[i][j] = 0;  
 else T[i][j] = 3;  
 }  
 }  
  
 return T;  
}

### Приклади роботи

На рисунку 3.1 показаний приклад роботи програми.

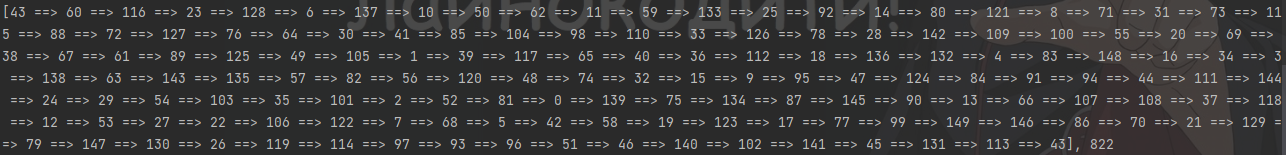


Рисунок 3.1 – результат виконання програми

## Тестування алгоритму

Проаналізуємо залежність якості рішення від параметрів а, b, p, M, MType, StartPoint

Графік 3.1 – Залежність якості рішення від параметру а

Графік 3.2 – Залежність якості рішення від параметру b

Графік 3.3 – Залежність якості рішення від параметру p

Графік 3.4 – Залежність якості рішення від параметру M

Графік 3.5 – Залежність якості рішення від параметру MType

Графік 3.6 – Залежність якості рішення від параметру StartPoint

Як бачимо з графіків, найвищу ефективність алгоритм проявив при таких значеннях параметрів: a = 2, b = 7, p = 0.55, M = 20, MType (відсоток диких чи елітних мурах) = 0%, StartPoint – починати з однієї й тої ж точки.

Графік 3.7 – Залежність якості рішення від різних параметрів

Як бачимо, найбільший вплив на якість рішення має параметр а. Другим по значимості йде параметр b, третє місце розділяють p та кількість і тип мурах.

Висновок

В рамках даної лабораторної роботи ми розробили мурашиний алгоритм для вирішення задачі комівояжера та проаналізували його роботу при різних параметрах і з’ясували, що оптимальними являються такі параметри: a = 2, b = 7, p = 0.55, M = 20, MType (відсоток диких чи елітних мурах) = 0%. Хоч це й суперечить здоровому глузду, було неоднократно доведено, що запускати мурах варто з одної точки.

Критерії оцінювання

При здачі лабораторної роботи до 26.11.2021 включно максимальний бал дорівнює – 5. Після 26.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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