

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)

ІП-13 Дем'янчук Олександр
(шифр, прізвище, ім'я, по батькові)

Перевірив

Сопов О. О.
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ.....	7
3.1	ПОКРОКОВИЙ АЛГОРИТМ	7
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	8
3.2.1	<i>Вихідний код.....</i>	8
3.2.2	<i>Приклади роботи</i>	15
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	17
	ВИСНОВОК	36
	КРИТЕРІЇ ОЦІНЮВАННЯ	37

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 ЗАВДАННЯ

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім ступенем 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
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

	<p>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.</p> <p>Розглядається симетричний, асиметричний та змішаний варіанти.</p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p> <p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – 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.
--	---

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
2	<p>Мурашиний алгоритм:</p> <ul style="list-style-type: none"> – α; – β;

	<ul style="list-style-type: none"> – ρ; – L_{min}; – кількість мурах M і їх типи (елітні, тощо...); – маршрути з однієї чи різних вершин.
--	---

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

1. Основний алгоритм

2. ПОЧАТОК

2.1. Ініціалізувати пустий масив AllTimeBestPath

2.2. Ініціалізувати число AllTimeBestDistance = Int32.MaxValue

2.3. ПОКИ не виконану кількість ітерацій – вхідний параметр iterations

2.3.1. Ініціалізація колонії мурах

2.3.2. Розташування мурах по графу в залежності від булевого параметра _differentPlacement

2.3.3. Запуск обходу кожної мурахи послідовно (Пункт 4)

2.3.4. ЦИКЛ ДЛЯ мурах у колонії

2.3.4.1. ЯКЩО шлях мурахи є циклом ТА ant._pathDistance < AllTimeBestDistance ТО

2.3.4.1.1. Зберегти шлях цієї мурахи у bestPath, дистанцію її шляху – у bestDistance

2.3.4.2. КІНЕЦЬ ЯКЩО

2.3.5. ЯКЩО bestDistance < AllTimeBestDistance ТО

2.3.5.1. AllTimeBestPath = bestPath

2.3.5.2. AllTimeBestDistance = bestDistance

2.3.6. КІНЕЦЬ ЯКЩО

2.3.7. Оновлення матриці феромонів

2.4. КІНЕЦЬ ПОКИ

3. КІНЕЦЬ

4. Метод обходу мурахи

4.1. ПОЧАТОК

4.1.1. ПОКИ мураха має доступні для переходу вершини

4.1.1.1. Отримати список доступних вершин для переходу

4.1.1.2. Отримати набір ймовірностей переходу у кожному з

вершин за формулою $\frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{j \in J_i} \tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}$

4.1.1.3. ЯКЩО мураха елітна ТО

4.1.1.3.1. Перейти у вершину з найбільшою ймовірністю

4.1.1.4. КІНЕЦЬ ЯКЩО

4.1.1.5. ЯКЩО мураха звичайна ТО

4.1.1.5.1. Отримати випадкове значення у межах [0.0, 1.0)

4.1.1.5.2. Обрахувати циклічно вершину для переходу залежно від випадкового значення

4.1.1.6. КІНЕЦЬ ЯКЩО

4.1.1.7. ЯКЩО мураха дика ТО

4.1.1.7.1. Обрати випадкову вершину без урахування ймовірностей

4.1.1.8. КІНЕЦЬ ЯКЩО

4.1.2. КІНЕЦЬ ПОКИ

4.2. КІНЕЦЬ

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Program.cs

```
using System.Diagnostics;

namespace Lab5;

class Program
{
    public static void Main(string[] args)
    {
        Graph gr = new Graph();
        AntColony antColony = new AntColony(g: gr, a: 1d, b: 1d, r: 0.5, lm:
500, true, elite: 0, reg: 10, wild: 0);
        Stopwatch stopwatch = Stopwatch.StartNew();
        antColony.Start(200);
        Console.WriteLine($"Done in {stopwatch.Elapsed}");
        Console.WriteLine("Best path is: ");
        antColony.WritePath();
        antColony.SaveResults();
        Console.WriteLine($"All the other info is written into
{AntColony.filepath}");
    }
}
```



```
}  
}
```

AntColony.cs

```
using System.Reflection.Metadata;  
  
namespace Lab5;  
  
public class AntColony  
{  
    public Graph graph;  
    public double _alpha;  
    public double _beta;  
    public double _rho;  
    public int Lmin;  
    private bool _differentPlacement;  
    private int _elites;  
    private int _regs;  
    private int _wilds;  
    private Ant[] colony;  
    private double[,] _pheromonesMatrix;  
    private int[] AllTimeBestPath;  
    private int AllTimeBestDistance;  
    public static string filepath = "data.txt";  
  
    public AntColony(Graph g, double a, double b, double r, int lm, bool  
placement, int elite, int reg, int wild)  
    {  
        graph = g;  
        _alpha = a;  
        _beta = b;  
        _rho = r;  
        Lmin = lm;  
        _differentPlacement = placement;  
        _elites = elite;  
        _regs = reg;  
        _wilds = wild;  
  
        _pheromonesMatrix = new double[Graph._amtOfVertices,  
Graph._amtOfVertices];  
        for (int i = 0; i < Graph._amtOfVertices; i++)  
        {  
            for (int j = 0; j < Graph._amtOfVertices; j++)  
            {  
                if (graph.DistanceMatrix[i, j] > 0)  
                    _pheromonesMatrix[i, j] = 0.1;  
                else  
                    _pheromonesMatrix[i, i] = 0d;  
            }  
        }  
  
        AllTimeBestDistance = Int32.MaxValue;  
    }  
  
    public void Start(int iterations)  
    {  
        int itr = 0;  
        int[] bestPath = new int[Graph._amtOfVertices];  
        int bestDistance;  
        while (itr < iterations)  
        {  
            itr++;  
            CreateColony();  
        }  
    }  
}
```

```

        PlaceAnts();
        foreach (Ant ant in colony)
            ant.Traverse(this);
        bestDistance = Int32.MaxValue;
        foreach (Ant ant in colony)
        {
            if (ant.IsPathValid(graph) && bestDistance > ant._pathDistance)
            {
                bestDistance = ant._pathDistance;
                bestPath = ant._path;
            }
        }

        if (AllTimeBestDistance > bestDistance)
        {
            AllTimeBestDistance = bestDistance;
            AllTimeBestPath = bestPath;
        }
        PlacePheromones();
    }
}

private void CreateColony()
{
    colony = new Ant[_elites + _regs + _wilds];
    int offset = 0;
    if (_elites > 0)
    {
        for (int i = 0; i < _elites; i++)
            colony[i + offset] = new EliteAnt();
        offset += _elites;
    }

    if (_regs > 0)
    {
        for (int i = 0; i < _regs; i++)
            colony[i + offset] = new RegularAnt();
        offset += _regs;
    }

    for (int i = 0; i < _wilds; i++)
        colony[i + offset] = new WildAnt();
}

public double[] GetChoiceProbs(int vertice, List<int> availableVertices)
{
    double[] probs = new double[availableVertices.Count];
    double sumOfProbs = 0d;
    for (int i = 0; i < probs.Length; i++)
    {
        probs[i] = CalculateProbability(vertice, availableVertices[i]);
        sumOfProbs += probs[i];
    }

    for (int i = 0; i < probs.Length; i++)
        probs[i] /= sumOfProbs;

    return probs;
}

private double CalculateProbability(int vertice, int probable) =>
    Math.Pow(_pheromonesMatrix[vertice, probable], _alpha) *
    Math.Pow(getVisibility(vertice, probable), _beta);

```

```

double getVisibility(int i, int j) => 1d / graph.DistanceMatrix[i, j];

private void PlaceAnts()
{
    Random rng = new Random();
    if (_differentPlacement)
    {
        foreach (Ant ant in colony)
            ant.PlaceAtStart(rng.Next(0, 300));
        return;
    }

    int vertice = rng.Next(0, 300);
    foreach (Ant ant in colony)
        ant.PlaceAtStart(vertice);
}

private void PlacePheromones()
{
    for (int i = 0; i < _pheromonesMatrix.GetLength(0); i++)
    {
        for (int j = 0; j < _pheromonesMatrix.GetLength(1); j++)
        {
            _pheromonesMatrix[i, j] *= (1d - _rho);
        }
    }
    foreach (Ant ant in colony)
    {
        for (int i = 1; i < ant._currentLength; i++)
        {
            _pheromonesMatrix[ant._path[i - 1], ant._path[i]] +=
ant.pheromones[i - 1] * ant.pheromoneCoeff;
        }
    }
}

public void WritePath()
{
    for (int i = 0; i < AllTimeBestPath.Length; i++)
        Console.Write($"{AllTimeBestPath[i]} ");
    Console.WriteLine($"\\nwith a distance of {AllTimeBestDistance}");
}

public void SaveResults()
{
    StreamWriter writer = new StreamWriter(AntColony.filepath, false);
    string diffplace = this._differentPlacement ? "yes" : "no";
    string message = $"Alpha: {_alpha}\\nBeta: {_beta}\\nRho: {_rho}\\nL min:
{Lmin}\\nAnts:\\n\\tElite: {_elites}\\n\\tRegular: {_regs}\\n\\tWild:
{_wilds}\\nDifferent Placement: {diffplace}";
    writer.Write(message);
    message = "\\nPheromones matrix:\\n";
    for (int i = 0; i < _pheromonesMatrix.GetLength(0); i++)
    {
        for (int j = 0; j < _pheromonesMatrix.GetLength(1); j++)
            message += $"{_pheromonesMatrix[i, j]} ";

        message += "\\n";
    }
    writer.Write(message);
    message = "\\nThe best path found:\\n";
    for (int i = 0; i < AllTimeBestPath.Length; i++)
        message += $"{AllTimeBestPath[i]} ";
}

```

```

        message += $"\\nwith a distance of {AllTimeBestDistance}";
        writer.Write(message);

        writer.Dispose();
    }
}

```

Ant.cs

```

using System.Net.Security;

namespace Lab5;

public abstract class Ant
{
    public int[] _path { get; }
    public int _currentLength;
    public int _pathDistance;
    public abstract int pheromoneCoeff { get; }
    public double[] pheromones { get; }

    public Ant()
    {
        _path = new int[Graph._amtOfVertices];
        pheromones = new double[Graph._amtOfVertices - 1];
        _currentLength = 0;
        _pathDistance = 0;
    }

    protected void SetPheromone(AntColony antColony)
    {
        pheromones[_currentLength - 2] = (double)antColony.Lmin / _pathDistance;
    }

    public void PlaceAtStart(int vertice)
    {
        _path[0] = vertice;
        _currentLength++;
    }

    public abstract void Traverse(AntColony antColony);

    protected List<int> GetAvailableVertices(AntColony antColony)
    {
        List<int> adjacents =
antColony.graph.GetAdjacentVertices(_path[_currentLength - 1]);
        for (int i = 0; i < _currentLength; i++)
            adjacents.Remove(_path[i]);
        return adjacents;
    }

    protected void MoveToVertice(AntColony antColony, List<int> adjacents, int
chosenVertice)
    {
        _path[_currentLength++] = adjacents[chosenVertice];
        _pathDistance += antColony.graph.DistanceMatrix[_path[_currentLength -
2], _path[_currentLength - 1]];

        SetPheromone(antColony);
    }

    public bool IsPathValid(Graph graph)

```

```

    {
        List<int> verts = graph.GetAdjacentVertices(_path[_currentLength - 1]);
        return _currentLength == Graph._amtOfVertices &&
verts.Contains(_path[0]);
    }
}

```

EliteAnt.cs

```

namespace Lab5;

public class EliteAnt : Ant
{
    public override int pheromoneCoeff => 2;

    public override void Traverse(AntColony antColony)
    {
        while (true)
        {
            List<int> adjacents = GetAvailableVertices(antColony);
            if (adjacents.Count == 0) return;
            double[] probabilites =
antColony.GetChoiceProbs(_path[_currentLength - 1], adjacents);

            int chosenVertice = 0;
            double maxProbability = 0d;

            for (int i = 0; i < probabilites.Length; i++)
            {
                if (maxProbability < probabilites[i])
                {
                    maxProbability = probabilites[i];
                    chosenVertice = i;
                }
            }

            MoveToVertice(antColony, adjacents, chosenVertice);
        }
    }
}

```

RegularAnt.cs

```

namespace Lab5;

public class RegularAnt : Ant
{
    public override int pheromoneCoeff => 1;

    public override void Traverse(AntColony antColony)
    {
        while (true)
        {
            List<int> adjacents = GetAvailableVertices(antColony);
            if (adjacents.Count == 0) return;
            double[] probabilites =
antColony.GetChoiceProbs(_path[_currentLength - 1], adjacents);
            Random rng = new Random();
            double randomChoice = rng.NextDouble();
            int chosenVertice = 0;
            if (probabilites.Length > 1)

```

```

        {
            for (int i = 1; i < probabilites.Length; i++)
                probabilites[i] += probabilites[i - 1];

            while (chosenVertice < probabilites.Length && randomChoice >
                probabilites[chosenVertice])
                chosenVertice++;
        }

        MoveToVertice(antColony, adjacents, chosenVertice);
    }
}

```

WildAnt.cs

```

namespace Lab5;

public class WildAnt : Ant
{
    public override int pheromoneCoeff => 1;

    public override void Traverse(AntColony antColony)
    {
        while (true)
        {
            List<int> adjacents = GetAvailableVertices(antColony);
            if (adjacents.Count == 0) break;
            Random rng = new Random();
            int chosenVertice = rng.Next(0, adjacents.Count);

            MoveToVertice(antColony, adjacents, chosenVertice);
        }
    }
}

```

Graph.cs

```

namespace Lab5;

public class Graph
{
    private static string path = "graph.txt";
    public static int _amtOfVertices = 300;
    public int[, ] DistanceMatrix { get; }

    public Graph()
    {
        DistanceMatrix = new int[_amtOfVertices, _amtOfVertices];
        for (int k = 0; k < _amtOfVertices; k++)
        {
            for (int l = 0; l < _amtOfVertices; l++)
            {
                DistanceMatrix[k, l] = -1;
            }
            DistanceMatrix[k, k] = 0;
        }

        int weight = 0;
        StreamReader reader = new StreamReader(path);
        for (int i = 0; i < _amtOfVertices; i++)
        {
            string[] line = reader.ReadLine().Split();
            for (int j = 0; j < _amtOfVertices; j++)
            {

```

```

        DistanceMatrix[i,j] = Int32.Parse(line[j]);
    }
}
reader.Dispose();
}

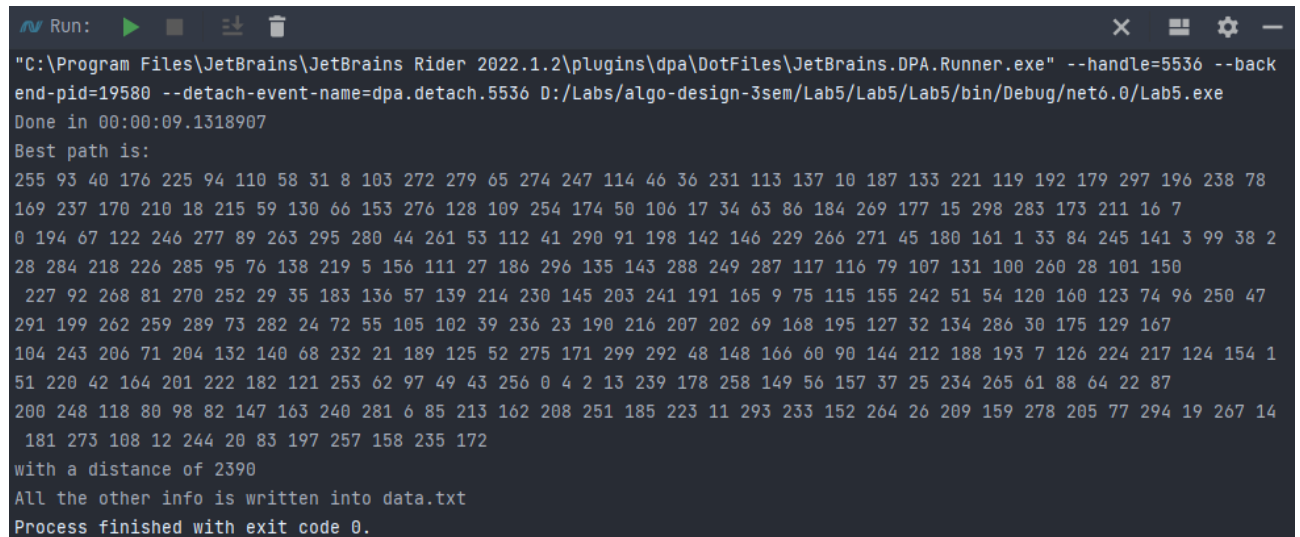
public List<int> GetAdjacentVertices(int vertice)
{
    List<int> adjacentVertices = new List<int>();
    for (int i = 0; i < _amtOfVertices; i++)
        if (vertice != i && DistanceMatrix[vertice, i] != -1)
adjacentVertices.Add(i);

    return adjacentVertices;
}
}

```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



```

Run: [Icons]
"C:\Program Files\JetBrains\JetBrains Rider 2022.1.2\plugins\dpa\DotFiles\JetBrains.DPA.Runner.exe" --handle=5536 --back
end-pid=19580 --detach-event-name=dpa.detach.5536 D:/Labs/algo-design-3sem/Lab5/Lab5/bin/Debug/net6.0/Lab5.exe
Done in 00:00:09.1318907
Best path is:
255 93 40 176 225 94 110 58 31 8 103 272 279 65 274 247 114 46 36 231 113 137 10 187 133 221 119 192 179 297 196 238 78
169 237 170 210 18 215 59 130 66 153 276 128 109 254 174 50 106 17 34 63 86 184 269 177 15 298 283 173 211 16 7
0 194 67 122 246 277 89 263 295 280 44 261 53 112 41 290 91 198 142 146 229 266 271 45 180 161 1 33 84 245 141 3 99 38 2
28 284 218 226 285 95 76 138 219 5 156 111 27 186 296 135 143 288 249 287 117 116 79 107 131 100 260 28 101 150
227 92 268 81 270 252 29 35 183 136 57 139 214 230 145 203 241 191 165 9 75 115 155 242 51 54 120 160 123 74 96 250 47
291 199 262 259 289 73 282 24 72 55 105 102 39 236 23 190 216 207 202 69 168 195 127 32 134 286 30 175 129 167
104 243 206 71 204 132 140 68 232 21 189 125 52 275 171 299 292 48 148 166 60 90 144 212 188 193 7 126 224 217 124 154 1
51 220 42 164 201 222 182 121 253 62 97 49 43 256 0 4 2 13 239 178 258 149 56 157 37 25 234 265 61 88 64 22 87
200 248 118 80 98 82 147 163 240 281 6 85 213 162 208 251 185 223 11 293 233 152 264 26 209 159 278 205 77 294 19 267 14
181 273 108 12 244 20 83 197 257 158 235 172
with a distance of 2390
All the other info is written into data.txt
Process finished with exit code 0.

```

Рисунок 3.1 – Виведення найкращого шляху

```
data.txt - Notepad
File Edit View

Alpha: 1
Beta: 1
Rho: 0,5
L min: 500
Ants:
    Elite: 0
    Regular: 10
    Wild: 0
Different Placement: yes
Pheromones matrix:
0 3,0462089036134075E-45 6,223015277861142E-62 6,223015277861142E-62 15,017263949418844 6,223015277861142E-62
142E-62 6,223015277861142E-62 0 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 0 0 0 6,2230
77861142E-62 1,8476663555622572E-61 6,223015277861142E-62 6,223015277861142E-62 0 6,223015277861142E-62 6,2230
,683006608032662E-60 0 6,223015277861142E-62 0 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-
20357875E-61 6,223015277861142E-62 6,223015277861142E-62 0 0 6,223015277861142E-62 6,223015277861142E-62 0 6,2
6,223015277861142E-62 0 0 6,223015277861142E-62 0 4,5321055427781465E-29 6,223015277861142E-62 0 2,14783406530
7861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,22301527
,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6
2 3,446545169127514E-41 7,637483742779962E-54 0 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E
1166265E-56 6,223015277861142E-62 3,326071319838699E-47 0 6,223015277861142E-62 3,244029214323219E-57 6,223015
-62 0 1,2763231019176056E-61 0 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 4,39788809901
6,223015277861142E-62 0 0 6,223015277861142E-62 1,3544648749652165E-14 1,794783357009364E-50 6,223015277861142
23015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 8,8
277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015
42E-62 6,223015277861142E-62 1,3723392640026706E-37 6,223015277861142E-62 6,223015277861142E-62 6,223015277861
6991509E-61 0 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 0,0011513894732423952 2,636344
61142E-62 6,223015277861142E-62 6,223015277861142E-62 5,5907767995315366E-61 6,223015277861142E-62 6,223015277
64192276745E-58 6,223015277861142E-62 6,223015277861142E-62 0
6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 0 6,223015277861142E-62 6,223015277861142E-6
6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 5,333660622888202E-56 6,223015277861142E-62
15277861142E-62 6,223015277861142E-62 6,223015277861142E-62 47,74041593013869 3,736923788977877E-61 6,22301527
2E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,22301527786114
2E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,22301527786114
223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 1,
61142E-62 6,672875448895635E-61 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 6,2230152778
3,989016900403007E-24 0 14,624803720106732 6,223015277861142E-62 0 6,223015277861142E-62 6,223015277861142E-62
5277861142E-62 0 6,223015277861142E-62 0 0 0 0 6,223015277861142E-62 6,223015277861142E-62 6,426773334711325
840037E-38 6,223015277861142E-62 6,223015277861142E-62 6,223015277861142E-62 0 6,223015277861142E-62 6,2230152
2,148705202199116E-08 6,223015277861142E-62 0 7,135304147822144E-61 6,223015277861142E-62 6,223015277861142E-6
142E-62 0 0 0 6,223015277861142E-62 6,223015277861142E-62 0 0 6,223015277861142E-62 0 0 2,6730429084921116E-28
```

Рисунок 3.2 – Збереження додаткової інформації у файл data.txt

3.3 Тестування алгоритму

Перед початком тестування необхідно означити: весь код працює на основі одного й того ж графа, збереженого у текстовому файлі graph.txt, що підпадає під умову Дірака: степінь кожної вершини є не меншою за загальну кількість вершин. До того ж опишемо деякі сталі значення, які будуть незмінними протягом усього процесу тестування:

- a. Кількість ітерацій - 200
- b. Кількість вершин – 300
- c. Нижня межа відстані між вершинами – 5
- d. Верхня межа відстані між вершинами – 150

Далі зафіксуємо початкові значення параметрів з таблиці 2.2, з яких почнеться наше дослідження. Дані початкові значення наведено у таблиці 3.3

Таблиця 3.3 – Початкові значення досліджуваних параметрів

Параметр	Значення
α	1.0
β	1.0
ρ	0.5
L_{\min}	500
$M_{\text{елітні}}$	0
$M_{\text{звичайні}}$	10
$M_{\text{дикі}}$	0
Маршрути з однієї чи різних вершин	3 різних вершин

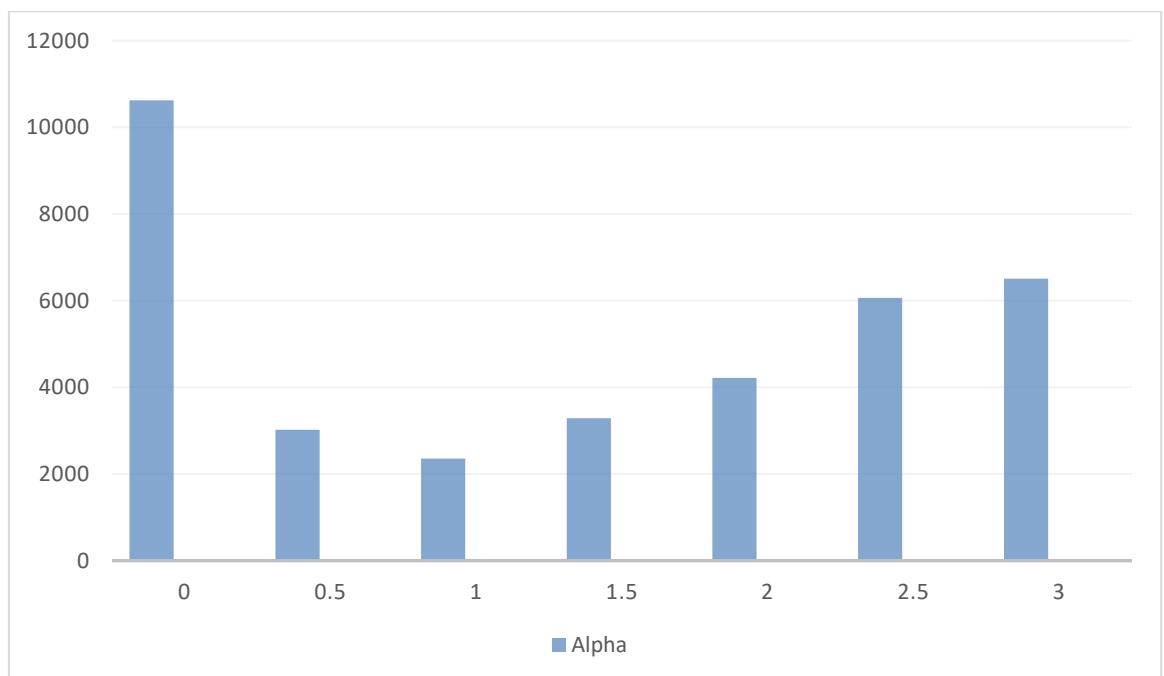
3.3.1 Дослідження параметру α

Параметр α відповідає за вагу значення феромона при обрахуванні ймовірності переходу в певну вершину. Набором значень α будуть числа від 0.0 до 3.0 з кроком 0.5. Інші параметри будуть узяті з таблиці 3.3. Для кожного значення α проведемо 5 дослідів і визначимо середній результат для поточного значення.

Таблиця 3.3 – Значення цільової функції при різних значеннях α

Значення α	Номер дослідження	Значення цільової функції
0	1	11024
	2	10367
	3	10299
	4	10706
	5	10707
	Середнє	10621
0.5	1	2993
	2	3043
	3	2988
	4	2958
	5	3120
	Середнє	3020
1	1	2308
	2	2336
	3	2309
	4	2373
	5	2449
	Середнє	2355
1.5	1	3157
	2	3290
	3	3332
	4	3133
	5	3537
	Середнє	3290
2	1	4218
	2	4558
	3	4470

	4	3957
	5	3878
	Середнє	4216
2.5	1	6061
	2	6168
	3	6012
	4	5882
	5	6203
	Середнє	6065
3	1	6416
	2	6418
	3	6640
	4	6949
	5	6105
	Середнє	6506



Діаграма 3.1 – Значення цільової функції при різних значеннях α

З діаграми розуміємо, що оптимальним значенням α є 1. При $\alpha=0$ алгоритм стає жадібним, що означає що мурахи найімовірніше перейдуть до

найближчої вершини, такий варіант не є надто точним для нашої задачі.

Занадто великі значення теж є неоптимальними.

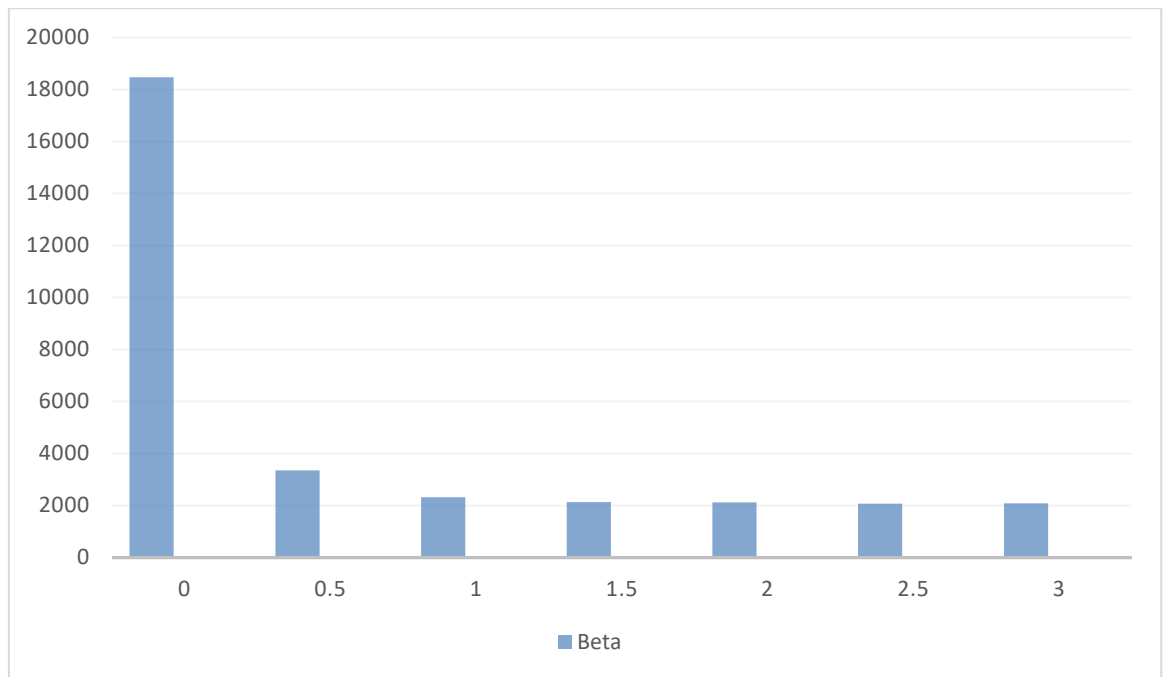
3.3.2 Дослідження параметру β

Параметр β відповідає за вагу значення видимості вершини (значення, обернене відстані до цієї вершини) при обрахуванні ймовірності переходу в певну вершину. Набором значень β будуть числа від 0.0 до 3.0 з кроком 0.5. Інші параметри будуть узяті з таблиці 3.3. Для кожного значення β проведемо 5 дослідів і визначимо середній результат для поточного значення.

Таблиця 3.4 – Значення цільової функції при різних значеннях β

Значення β	Номер дослідження	Значення цільової функції
0	1	18386
	2	18470
	3	18753
	4	18232
	5	18511
	Середнє	18470
0.5	1	3132
	2	3340
	3	3319
	4	3367
	5	3612
	Середнє	3354
1	1	2345
	2	2354
	3	2358
	4	2328
	5	2209
	Середнє	2319

1.5	1	2118
	2	2108
	3	2096
	4	2217
	5	2073
	Середнє	2142
2	1	2084
	2	2090
	3	2022
	4	2077
	5	2129
	Середнє	2080
2.5	1	2129
	2	2003
	3	2105
	4	2065
	5	2103
	Середнє	2081
3	1	2030
	2	2143
	3	2089
	4	2059
	5	2125
	Середнє	2089



Діаграма 3.2 – Значення цільової функції при різних значеннях β

Бачимо, що різке підвищення ефективності спостерігається при переході зі значень $\beta=0$ і $\beta=0.5$. При $\beta=0$ мурахи починають орієнтуватись суто на «запах» феромону, що теж виявилось вкрай неефективним, оскільки на початку всі дуги мають однакову кількість феромону. Далі зміна значення бета не грає важливої ролі. І все ж, з мінімальним розривом найоптимальнішим виявилось значення 2.5.

3.3.3 Дослідження набору параметрів α і β

В попередніх дослідях ми розглядали значення α і β по окремі, беручи початкові значення інших параметрів. Тепер поглянемо на взаємодію цих двох величин: набором значень є декартовий добуток наборів α від 0 до 3 через 0.5 і β від 0 до 3 через 0.5. Усі інші значення беруться з таблиці початкових значень.

Таблиця 3.5 – Значення цільової функції при різних значеннях α та β

Значення α	Значення β	Значення цільової функції
0	0	21142
0	0.5	16080
0	1	10737
0	1.5	6987
0	2	4663
0	2.5	3566

0	3	3041
0.5	0	20461
0.5	0.5	5221
0.5	1	3037
0.5	1.5	2573
0.5	2	2490
0.5	2.5	2253
0.5	3	2164
1	0	18498
1	0.5	3311
1	1	2364
1	1.5	2158
1	2	2167
1	2.5	2006
1	3	2004
1.5	0	21086
1.5	0.5	6796
1.5	1	3479
1.5	1.5	2582
1.5	2	2402
1.5	2.5	2178
1.5	3	2151
2	0	21250
2	0.5	9984
2	1	4216
2	1.5	3092
2	2	2533
2	2.5	2328
2	3	2216
2.5	0	21524
2.5	0.5	12760
2.5	1	5940
2.5	1.5	3163
2.5	2	2775
2.5	2.5	2493
2.5	3	2486
3	0	22360
3	0.5	12674
3	1	7664
3	1.5	3567
3	2	3099
3	2.5	2541
3	3	2443

Таким чином, взаємно оптимальними значеннями для α і β є 1 і 3 відповідно. Для наступних досліджень використовуватимемо саме ці значення. Також спостерігається, що при $\alpha=0$ або $\beta=0$ спостерігаються погані результати, в цілому вони покращуються, коли $\alpha>0$ і $\beta>0$.

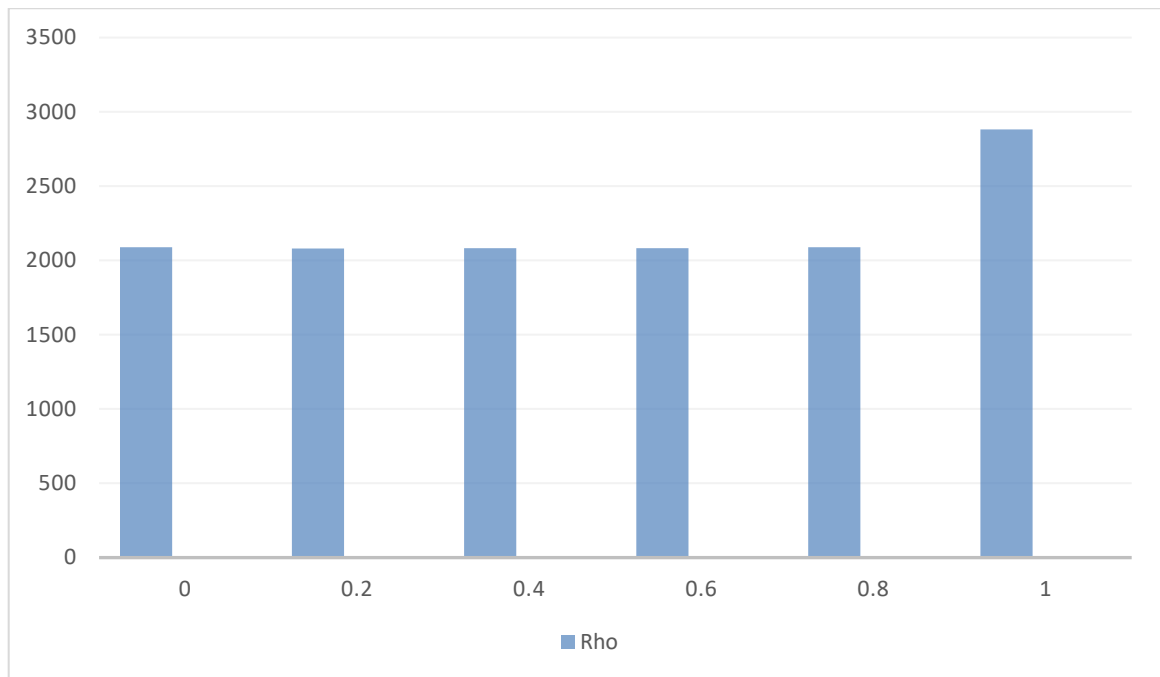
3.3.4 Дослідження параметру ρ

Наступний параметр у серії досліджень – ρ . Він визначає швидкість випаровування феромонів на усьому графі: з кожною ітерацією старі значення феромонів зменшуються у $\frac{1}{1-\rho}$ разів. Проведемо серію вимірювань при $\alpha=1$, $\beta=3$, інші значення – з таблиці 3.3, ρ – від 0 до 1 з кроком 0.2.

Таблиця 3.6 – Значення цільової функції при різних значеннях ρ

Значення ρ	Номер дослідження	Значення цільової функції
0	1	2071
	2	2084
	3	2095
	4	2107
	5	2093
	Середнє	2090
0.2	1	2095
	2	1992
	3	2144
	4	2103
	5	2069
	Середнє	2081
0.4	1	2000
	2	2079
	3	2157
	4	2072

	5	2108
	Середнє	2083
0.6	1	2143
	2	1981
	3	2100
	4	2083
	5	2101
	Середнє	2082
0.8	1	1985
	2	2132
	3	2074
	4	2147
	5	2104
	Середнє	2088
1	1	2852
	2	2809
	3	2974
	4	2962
	5	2809
	Середнє	2881



Діаграма 3.3 – Значення цільової функції при різних значеннях ρ

Отже, чітко видно, що найважливішим критерієм графіку є $\rho < 1$, але строго математично маємо найоптимальніше значення $\rho = 0.2$. При моментальному випаровуванні ($\rho = 1$) мурахи не відчують феромону, залишеного попередніми популяціями, тому результати – найгірші. Надалі використовуватимемо оптимальне значення ρ .

3.3.5 Дослідження параметру L_{\min}

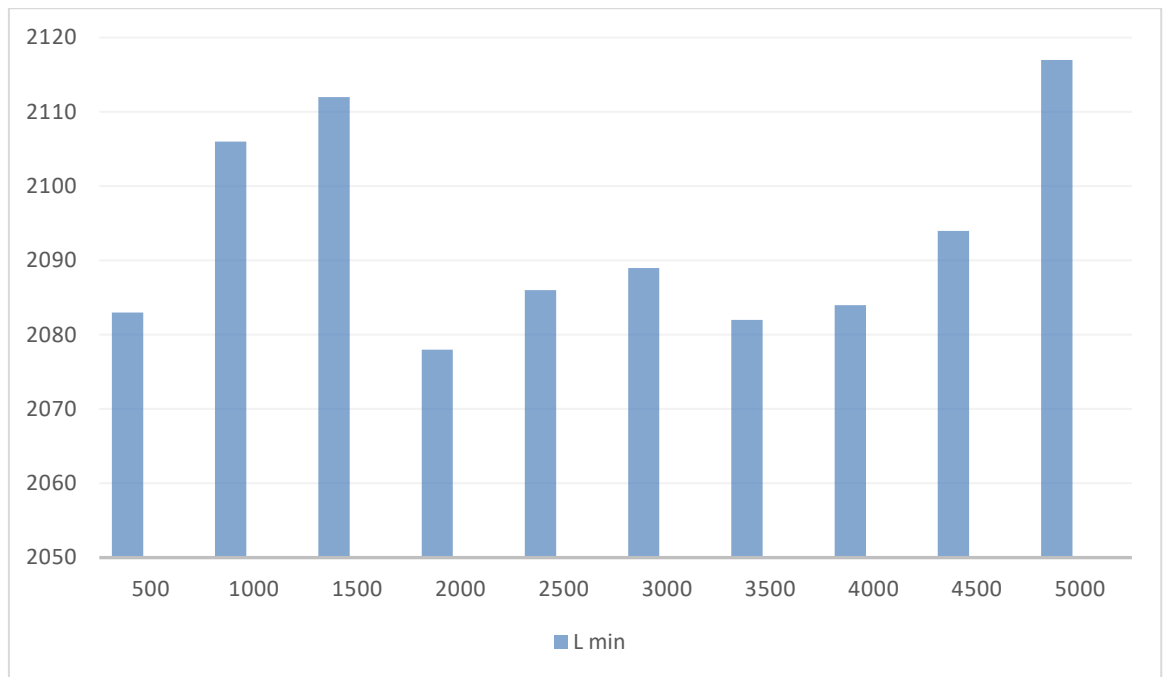
Наступним параметром у задачі є L_{\min} – ціна передбачуваного ідеального рішення проблеми. Параметр фігурує у формулі кількості феромону, що відкладає кожна мураха ($\frac{L_{\min}}{L_k}$). Тому чим більше очікуване ідеальне рішення, тим більше відкладатиме феромону мураха. Проводитимемо дослідження при знайдених раніше оптимальних рішеннях, інші – з таблиці 3.3. Діапазон L_{\min} – від 500 до 5000 з кроком 500.

Таблиця 3.7 – Значення цільової функції при різних значеннях L_{\min}

Значення L_{\min}	Номер дослідження	Значення цільової функції
500	1	2113
	2	2084

	3	2083
	4	2008
	5	2128
	Середнє	2083
1000	1	2122
	2	2103
	3	2058
	4	2127
	5	2119
	Середнє	2106
1500	1	2109
	2	2061
	3	2146
	4	2144
	5	2099
	Середнє	2112
2000	1	2115
	2	2052
	3	2029
	4	2102
	5	2092
	Середнє	2078
2500	1	2101
	2	2107
	3	2052
	4	2077
	5	2094
	Середнє	2086
3000	1	2103

	2	2063
	3	2124
	4	2069
	5	2088
	Середнє	2089
3500	1	2061
	2	2101
	3	2089
	4	2063
	5	2094
	Середнє	2082
4000	1	2043
	2	2105
	3	2117
	4	2014
	5	2142
	Середнє	2084
4500	1	2101
	2	2140
	3	2059
	4	2103
	5	2065
	Середнє	2094
5000	1	2113
	2	2086
	3	2098
	4	2137
	5	2152
	Середнє	2117



Діаграма 3.4 – Значення цільової функції при різних значеннях L_{\min}

Цей параметр не дає чітких тенденцій, видно, що параметр не сильно впливає на ефективність алгоритму, але зрозуміло, що найоптимальнішим є значення $L_{\min}=2000$. Записуємо його до інших оптимальних рішень та використовуємо далі.

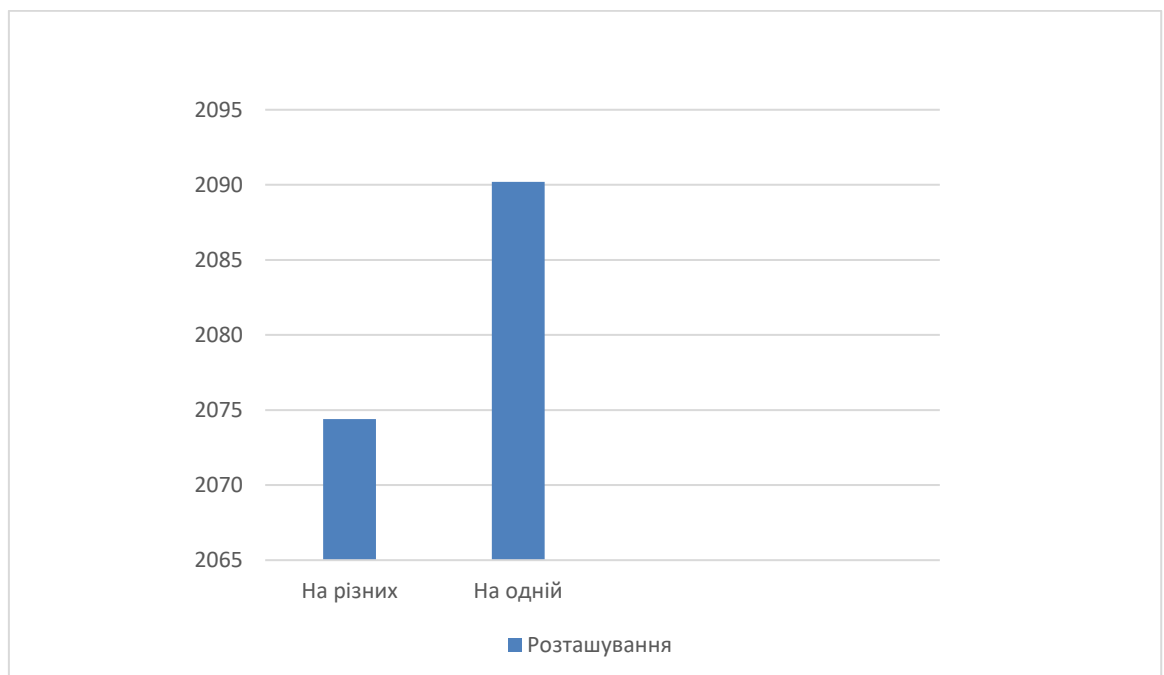
3.3.6 Дослідження методу розташування мурах

Наступний параметр не є скалярним, проте не менш важливий за попередні. Розташування мурах можливе у одній або різних вершинах. Проведемо по 10 дослідів на обох варіантах, використовуємо знайдені раніше оптимальні параметри, інші – з таблиці 3.3.

Таблиця 3.8 – Значення цільової функції при різних розміщеннях мурах

Розташування мурах	Номер дослідження	Значення цільової функції
На різних вершинах	1	2027
	2	2072
	3	2064
	4	2117
	5	2084

	6	2108
	7	2079
	8	2019
	9	2114
	10	2060
	Середнє	2074.4
На одній вершині	1	2065
	2	2093
	3	2117
	4	2073
	5	2051
	6	2029
	7	2114
	8	2105
	9	2100
	10	2155
	Середнє	2090.2



Діаграма 3.5 – Значення цільової функції при різних розміщеннях мурах

Бачимо прекрасні результати роботи при обох варіантах, проте розташування на різних вершинах все ж перемагає. Вважаємо таке розміщення оптимальним.

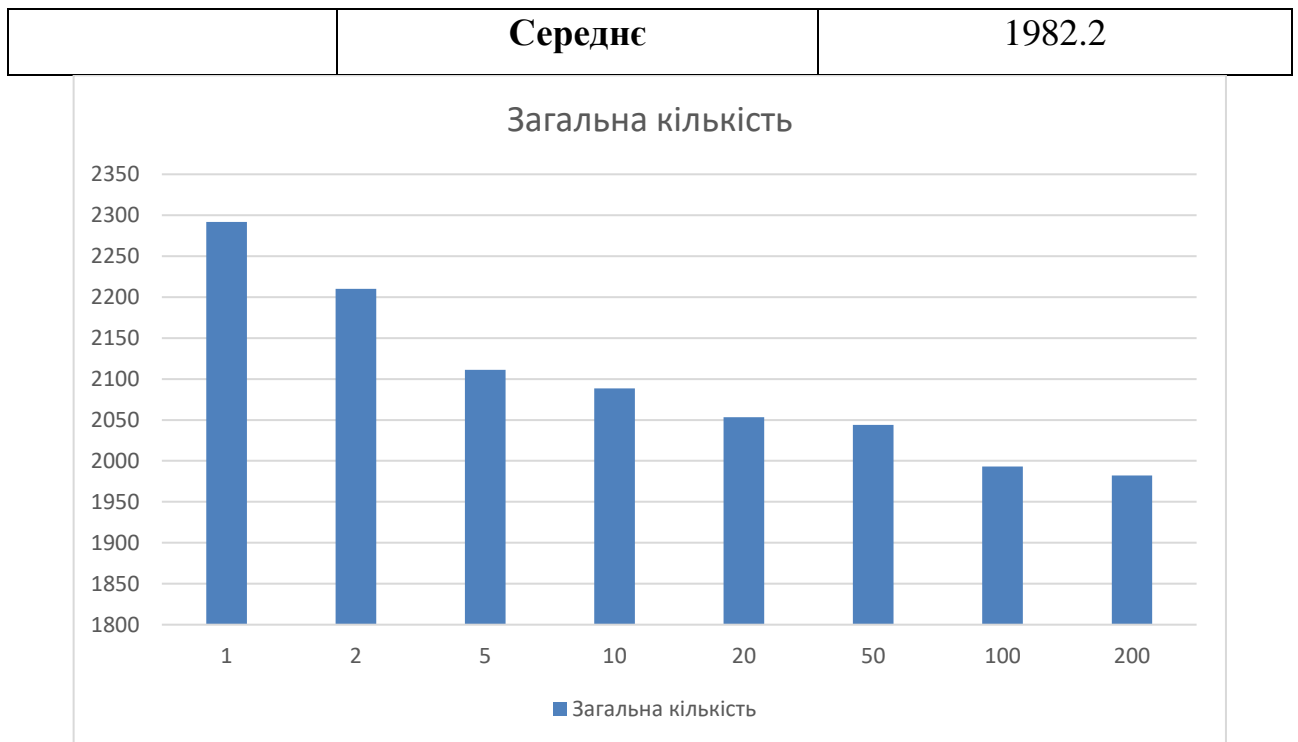
3.3.7 Дослідження параметрів M , $M_{\text{елітні}}$, $M_{\text{звичайні}}$, $M_{\text{дикі}}$

Останніми параметрами є загальна кількість мурах M , та окрема кількість кожного з видів. Спочатку дослідимо оптимальну кількість мурах в загальному, використовуючи вид звичайних мурах. Діапазон візьмемо від 1 до 200, крок експоненційний.

Таблиця 3.9 – Значення цільової функції при різній кількості мурах M

Значення M	Номер дослідження	Значення цільової функції
1	1	2379
	2	2227
	3	2244
	4	2327
	5	2283
	Середнє	2292
2	1	2182
	2	2197
	3	2291
	4	2173
	5	2208
	Середнє	2210.2
5	1	2106
	2	2124
	3	2096
	4	2133
	5	2097
	Середнє	2111.2

10	1	2067
	2	2048
	3	2111
	4	2112
	5	2105
	Середнє	2088.6
20	1	2067
	2	2050
	3	2004
	4	2068
	5	2077
	Середнє	2053.2
50	1	2073
	2	1994
	3	2052
	4	2047
	5	2053
	Середнє	2043.8
100	1	2000
	2	1971
	3	2009
	4	2000
	5	1986
	Середнє	1993.2
200	1	1990
	2	1941
	3	2004
	4	1972
	5	2004



Діаграма 3.6 – Значення цільової функції при різній кількості мурах M

Загальна тенденція цілком зрозуміла та логічна – чим більше мурах в колонії – тим краще результат, тому вносимо кількість мурах $M=200$ до оптимальних значень.

Тепер дослідимо кількість елітних мурах $M_{\text{елітні}}$. Діапазон – від 0 до 200, інші мурахи – звичайні.

Таблиця 3.10 – Значення цільової функції при різних значеннях $M_{\text{елітні}}$

Значення $M_{\text{звичайні}}$	Значення $M_{\text{елітні}}$	Значення цільової функції
200	0	1994
199	1	1984
198	2	1989
195	5	2013
190	10	2012
180	20	2056
150	50	2035
100	100	2080
0	200	2097

Отже, найкращі результати виявились при наборі зі 199 звичайних та 1 елітної мурахи. Таку кількість елітних мурах використовуватимемо і надалі.

Тепер протестуємо кількість диких мурах $M_{\text{дикі}}$. Розглядуватимемо 2 випадки, коли є 1 елітна мураха і коли їх нема зовсім. У першому випадку діапазон кількості диких мурах буде від 0 до 199, а кількість звичайних мурах варіюватиметься від 200 до 0. Другий випадок – окрема єдина ситуація, коли маємо колонію з 200 диких мурах.

Таблиця 3.11 – Значення цільової функції при різних значеннях $M_{\text{дикі}}$

$M_{\text{звичайні}}$	$M_{\text{елітні}}$	$M_{\text{дикі}}$	Значення цільової функції
199	1	0	2003
198	1	1	2015
197	1	2	2031
194	1	5	1972
189	1	10	1983
179	1	20	1968
149	1	50	2031
99	1	100	2036
0	1	199	2333
0	0	200	20090

Ось таким чином, через серію дослідів, отримали оптимальні значення усіх досліджуваних параметрів задачі комівояжера. Їх значення наведені нижче у таблиці 3.12.

Таблиця 3.12 – Оптимальні значення досліджуваних параметрів

Параметр	Значення
α	1.0
β	3.0
ρ	0.2
L_{\min}	2000
$M_{\text{елітні}}$	1
$M_{\text{звичайні}}$	179
$M_{\text{дикі}}$	20
Маршрути з однієї чи різних вершин	З різних вершин

ВИСНОВОК

В рамках даної лабораторної роботи дослідив ціль задачі комівояжера, реалізував пошук рішення за допомогою алгоритму мурашиної колонії (АСО), а саме:

- Розглянув концепцію мурашиної колонії
- Побудував граф що підходить для задачі
- Реалізував поведінку різних видів мурах
- Реалізував основний алгоритм
- Провів серію досліджень з пошуку оптимальних значень параметрів мурашиного алгоритму.

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%.