**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.1”**

**Виконав(ла)** *ІП-14 Хільчук А.В.* _____
(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Ахаладзе І.Е.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

# 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

# 2 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 M = 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 M = 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 M = 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 M = 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 M = 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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 ($\alpha = 2$, $\beta = 3$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 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). |

| | |
|---|---|
| 31 | 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. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | 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. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах). |

# 3 ВИКОНАННЯ

## 3.1 Програмна реалізація алгоритму

### 3.1.1 Вихідний код

**IAnt.cs**

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PALab3.Ants
{
    interface IAnt
    {
        public int CalculateGreedyPath(int length);
        public IEnumerable<Tuple<City,City>> SeekPath(int length);
        public void Refresh();
        public int GetLMin();
        public int GetPathLength();
        public Tuple<City, City> ReturnToStart();
    }
}
```

**RegularAnt.cs**

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
namespace PALab3.Ants
{
    class RegularAnt : IAnt
    {
        private Stack<City> _path;
        private int _lmin;
        private int _pathLength;
        private AntColony _colony;

        public RegularAnt(City startCity, AntColony colony)
        {
            _path = new Stack<City>();
            _path.Push(startCity);
            _colony = colony;
        }

        public int CalculateGreedyPath(int length)
        {
            var pathLen = 0;
            var tempPath = new Stack<City>();
            tempPath.Push(_path.Peek());

            while (tempPath.Count != length)
            {
                var minAdjacent = tempPath.Peek().GetAdjacents()
                    .Where(a => !tempPath.Contains(a.Key))
                    .OrderBy(a=>a.Value).First();

                tempPath.Push(minAdjacent.Key);
                pathLen += minAdjacent.Value;
            }
```

```
    var temp = tempPath.ToArray();

    var startCity = temp[tempPath.Count - 1];

    pathLen += tempPath.Peek().GetAdjacents()[startCity];//перевірити

    return pathLen;
}


public IEnumerable<Tuple<City, City>> SeekPath(int length)
{
    Random rand = new Random();
    _lmin = CalculateGreedyPath(length);

    while (_path.Count != length)
    {
        var chance = rand.NextDouble();
        var chances = CalculateChances().ToList();
        var adjacents = _path.Peek().GetAdjacents();

        if (chance < chances[0].Item2)
        {
            var temp = _path.Peek();

            _pathLength += adjacents[chances[0].Item1];
            _path.Push(chances[0].Item1);
            yield return Tuple.Create<City, City>(temp, chances[0].Item1);
        }
        else
        {
            for (int i = 1; i < chances.Count; i++)
            {
                if (chance > chances[i - 1].Item2 && chance < chances[i].Item2)
                {
                    _pathLength += adjacents[chances[i].Item1];

                    var temp = _path.Peek();

                    _path.Push(chances[i].Item1);

                    yield return Tuple.Create<City, City>(temp, chances[i].Item1);
                    break;
                }
            }
        }

    }
    yield return ReturnToStart();
}

public List<Tuple<City, double>> CalculateChances()
{
    var pheromones = _colony.GetPheromones(_path.Peek())
        .Where(a => !_path.Contains(a.Key))
        .OrderBy(a=>a.Value);

    var adjacentsDistances = _path.Peek().GetAdjacents()
        .Where(a => !_path.Contains(a.Key))
        .ToDictionary(a=>a.Key,a=>a.Value);

    double totalPheromonesXVisibility = 0;
    foreach(var pheromone in pheromones)
    {
        totalPheromonesXVisibility +=
```

```
                    Math.Pow(pheromone.Value, _colony.Alpha) *
                    Math.Pow(1D /(double)adjacentsDistances[pheromone.Key], _colony.Beta);
            }

            List<Tuple<City, double>> citiesAndProbabilities = new List<Tuple<City,
double>>();
            foreach (var pheromone in pheromones)
            {
                citiesAndProbabilities.Add( Tuple.Create(
                    pheromone.Key,
                    (Math.Pow(pheromone.Value, _colony.Alpha) *
                    Math.Pow(1D / (double)adjacentsDistances[pheromone.Key], _colony.Beta))/
totalPheromonesXVisibility));
            }

            return citiesAndProbabilities;
        }


        public int GetLMin()
        {
            return _lmin;
        }

        public int GetPathLength()
        {
            return _pathLength;
        }

        public Tuple<City, City> ReturnToStart()
        {
            var startCity = _path.ToArray()[_path.Count - 1];

            _pathLength += _path.Peek().GetAdjacents()[startCity];

            var currentCity = _path.Peek();

            _path.Push(startCity);

            return Tuple.Create<City, City>(currentCity, startCity);
        }

        public void Refresh()
        {
            while (_path.Count != 1)
            {
                _path.Pop();
            }

            _pathLength = 0;
            _lmin = 0;
        }
    }
}
```

## WildAnt.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace PALab3.Ants
{
    class WildAnt : IAnt
    {
```

```
private Stack<City> _path;
private int _lmin;
private int _pathLength;
private AntColony _colony;

public WildAnt(City startCity, AntColony colony)
{
    _path = new Stack<City>();
    _path.Push(startCity);
    _colony = colony;
}

public int CalculateGreedyPath(int length)
{
    var pathLen = 0;
    var tempPath = new Stack<City>();

    tempPath.Push(_path.Peek());

    while (tempPath.Count != length)
    {

        var minAdjacent = tempPath.Peek().GetAdjacents()
            .Where(a => !tempPath.Contains(a.Key))
            .OrderBy(a=>a.Value).First();

        tempPath.Push(minAdjacent.Key);
        pathLen += minAdjacent.Value;

    }
    var temp = tempPath.ToArray();

    var startCity = temp[tempPath.Count - 1];

    pathLen += tempPath.Peek().GetAdjacents()[startCity];//перевірити

    return pathLen;
}

public int GetLMin()
{
    return _lmin;
}

public int GetPathLength()
{
    return _pathLength;
}

public void Refresh()
{
    while (_path.Count != 1)
    {
        _path.Pop();
    }

    _pathLength = 0;
    _lmin = 0;
}

public Tuple<City, City> ReturnToStart()
{
    var temp = _path.ToArray();
```

```
            var startCity = temp[_path.Count - 1];

            var currentCity = _path.Peek();

            _pathLength += currentCity.GetAdjacents()[startCity];//перевірити

            _path.Push(startCity);

            return Tuple.Create<City, City>(currentCity, startCity);

        }

        public IEnumerable<Tuple<City, City>> SeekPath(int length)
        {
            Random rand = new Random();
            _lmin = CalculateGreedyPath(length);
            while (_path.Count != length)
            {
                var temp = _path.Peek();

                var adjacents = temp.GetAdjacents()
                    .Where(a => !_path.Contains(a.Key))
                    .ToDictionary(a=>a.Key,a=>a.Value);
                var decision = adjacents.OrderBy(_ => rand.Next()).First();

                _pathLength += adjacents[decision.Key];
                _path.Push(decision.Key);

                yield return Tuple.Create<City, City>(temp, decision.Key);
            }
            yield return ReturnToStart();
        }
    }
}
```

## AntColony.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using PALab3.Ants;
using System.Linq;

namespace PALab3
{
    class AntColony
    {
        private int _size;
        private List<City> _cities;
        private int _antCount = 0;
        private int _wildAntCount = 0;
        private List<IAnt> _ants;
        private Dictionary<Tuple<City, City>, Double> _edgesAndPheromoneLevels;

        public double Alpha
        {
            get;
            private set;
        }
        public double Beta
        {
            get;
            private set;
        }
        public double Ro
        {
```

```
            get;
            private set;
        }


        public AntColony(int size, int antCount, int wildAntCount, double alpha, double
beta, double ro)
        {
            _size = size;

            _cities = new List<City>();

            for (int i = 0; i < _size; i++)
            {
                _cities.Add(new City(i));
            }

            _antCount = antCount;
            _wildAntCount = wildAntCount;

            Alpha = alpha;
            Beta = beta;
            Ro = ro;
        }

        public AntColony InitCityTravelCost()
        {
            foreach(var a in _cities)
            {
                a.MapAdjacents(_cities);
            }
            return this;
        }

        public AntColony InitAnts()
        {
            _ants = new List<IAnt>();

            Random rand = new Random();
            _cities.OrderBy(_ => rand.Next());

            int i;
            for (i = 0; i < _antCount; i++)
            {
                _ants.Add(new RegularAnt(_cities[i], this));
            }

            for (int j = 0; j < _wildAntCount; j++)
            {
                _ants.Add(new WildAnt(_cities[i+j],this));
            }
            return this;
        }
        public AntColony InitPheromones()
        {
            Random rand = new Random();

            _edgesAndPheromoneLevels = new Dictionary<Tuple<City, City>, double>();

            foreach(var city in _cities)
            {
                var temp = city.GetAdjacents();

                foreach(var adjacentCity in temp)
                {
```

```
                    var tempDouble = rand.NextDouble();
                    while (tempDouble == 0) { tempDouble = rand.NextDouble(); }// щоб
переконатися, що буде ненульове, щоб потім не було проблем з виконанянм алгоритму
                    _edgesAndPheromoneLevels.Add(
                        Tuple.Create(city,adjacentCity.Key),
                        tempDouble);
                }
            }

            return this;
        }

        public Dictionary<City, double> GetPheromones(City fromWhere)
        {
            Dictionary<City, double> goal = _edgesAndPheromoneLevels
                .Where(a => a.Key.Item1 == fromWhere)
                .ToDictionary(a=> a.Key.Item2, a=>a.Value);

            return goal;
        }

        public void EvaporatePheromones(Dictionary<Tuple<City,City>,List<IAnt>>
pathwaysAndAntsThatUsedThem)
        {

            foreach(var pathway in _edgesAndPheromoneLevels.Keys)
            {
                double deltaT = 0;
                if (pathwaysAndAntsThatUsedThem.ContainsKey(pathway))
                {
                    foreach (var ant in pathwaysAndAntsThatUsedThem[pathway])
                    {
                        deltaT += (double)ant.GetLMin() / (double)ant.GetPathLength();
                    }
                }
                _edgesAndPheromoneLevels[pathway]=(1-
Ro)*_edgesAndPheromoneLevels[pathway]+deltaT;
            }
        }

        public IEnumerable<Tuple<int,double>> SolveTravellingMerchantProblem(int
colonyLifeSpan)
        {
            for (int i = 0; i < colonyLifeSpan; i++)
            {

                Dictionary<Tuple<City, City>, List<IAnt>> pathwaysAndAntsThatUsedThem = new
Dictionary<Tuple<City, City>, List<IAnt>>();
                foreach (var ant in _ants)
                {
                    foreach(var path in ant.SeekPath(_size))
                    {
                        if (pathwaysAndAntsThatUsedThem.ContainsKey(path))
                        {
                            pathwaysAndAntsThatUsedThem[path].Add(ant);
                        }
                        else
                        {
                            pathwaysAndAntsThatUsedThem.Add(path, new List<IAnt>());
                            pathwaysAndAntsThatUsedThem[path].Add(ant);
                        }
                    }
                }

                var minimum = _ants.Min(a => a.GetPathLength());
```

```
                double average = _ants.Average(a => a.GetPathLength());
                EvaporatePheromones(pathwaysAndAntsThatUsedThem);

                foreach(var ant in _ants)
                {
                    ant.Refresh();
                }

                yield return Tuple.Create(minimum, average);
            }
        }
    }
}
```

## City.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PALab3
{
    class City
    {
        private int _identificator;
        private Dictionary<City, int> _adjacentCitiesAndTravelCosts;
        public City(int identificator)
        {
            _identificator = identificator;
            _adjacentCitiesAndTravelCosts = new Dictionary<City, int>();
        }

        public void MapAdjacents(List<City> cities)
        {
            Random rand = new Random();

            foreach(var city in cities)
            {
                if (city == this) continue;
                _adjacentCitiesAndTravelCosts.Add(city, rand.Next(1, 41));
            }
        }

        public Dictionary<City, int> GetAdjacents()
        {
            return _adjacentCitiesAndTravelCosts;
        }

        private int GetIdentificator()
        {
            return _identificator;
        }
    }
}
```

## Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace PALab3
{
    class Program
    {
        static void Main(string[] args)
```

```
{
    int cityNum = -1,
        regularAnts=-1,
        wildAnts=-1;

    double alpha=-1,
           beta= -1,
           ro= -1;


    Console.WriteLine("Welcome to ant colony algorithm!\n");

    Console.WriteLine("First off, determine prameters:");

    Console.WriteLine("Would you like to choose default values? [Y/N]");
    var decision = Console.ReadLine().ToUpper();

    while (decision != "Y"&&decision != "N")
    {
        Console.WriteLine("Please, enter one of the two options");
        decision = Console.ReadLine().ToUpper();
    }

    switch (decision)
    {
        case "Y":
            {
                cityNum = 200;
                regularAnts = 30;
                wildAnts = 15;
                alpha = 2;
                beta = 4;
                ro=0.7;
              break; }
        default: {
                Console.WriteLine("\tPlease, enter the number of cities:");

                var tempValue = Console.ReadLine();

                while (!Int32.TryParse(tempValue, out cityNum) && cityNum < 0)
                {
                    Console.WriteLine("Please, enter the correct value");
                    tempValue = Console.ReadLine();
                }

                Console.WriteLine("\tEnter number of regular ants:");

                tempValue = Console.ReadLine();

                while (!Int32.TryParse(tempValue, out regularAnts) && regularAnts <
1)
                {
                    Console.WriteLine("Please, enter the correct value");
                    tempValue = Console.ReadLine();
                }

                Console.WriteLine("\tEnter number of wild ants:");

                tempValue = Console.ReadLine();

                while (!Int32.TryParse(tempValue, out wildAnts) && wildAnts < 1)
                {
                    Console.WriteLine("Please, enter the correct value");
                    tempValue = Console.ReadLine();
                }
```

```
                        Console.WriteLine("\tEnter alpha:");

                        tempValue = Console.ReadLine();

                        while (!Double.TryParse(tempValue, out alpha) && alpha < 0)
                        {
                            Console.WriteLine("Please, enter the correct value");
                            tempValue = Console.ReadLine();
                        }

                        Console.WriteLine("\tEnter Beta:");

                        tempValue = Console.ReadLine();

                        while (!Double.TryParse(tempValue, out beta) && beta < 0)
                        {
                            Console.WriteLine("Please, enter the correct value");
                            tempValue = Console.ReadLine();
                        }

                        Console.WriteLine("\tEnter ro:");

                        tempValue = Console.ReadLine();

                        while (!Double.TryParse(tempValue, out ro) && ro < 0&&ro>=1)
                        {
                            Console.WriteLine("Please, enter the correct value");
                            tempValue = Console.ReadLine();
                        }
                        break; }
                }



                Console.WriteLine("\nEverything's ready. Launching the algorithm...\n");
                Console.WriteLine("To exit, press ctrl+c");

                AntColony colony = new AntColony(cityNum, regularAnts, wildAnts, alpha, beta,
ro);

                colony.InitCityTravelCost()
                    .InitAnts()
                    .InitPheromones();
                int i = -1;
                foreach (var a in colony.SolveTravellingMerchantProblem(2001))
                {
                    i++;
                    if (i % 20 == 0)
                    {
                        Console.WriteLine("Iteration: {0}",i+1);
                        Console.WriteLine("Minimal:{0}\nAverage:{1}", a.Item1,a.Item2);
                    }
                }
            }
        }
}
```

### 3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
Welcome to ant colony algorithm!

First off, determine prameters:
Would you like to choose default values? [Y/N]
y

Everything's ready. Launching the algorithm...

To exit, press ctrl+c
Iteration: 1
Minimal:315
Average:1604,088888888889
Iteration: 21
Minimal:242
Average:1549,3777777777777
Iteration: 41
Minimal:249
Average:1555,7777777777778
Iteration: 61
Minimal:238
Average:1557,7333333333333
Iteration: 81
Minimal:238
Average:1567,4444444444443
Iteration: 101
Minimal:240
Average:1532,7777777777778
Iteration: 121
Minimal:238
Average:1551,8
Iteration: 141
Minimal:238
Average:1549,4222222222222
Iteration: 161
Minimal:238
Average:1539,6
Iteration: 181
Minimal:238
Average:1552,6444444444444
Iteration: 201
Minimal:238
Average:1544,2
Iteration: 221
Minimal:238
Average:1535,0666666666666
Iteration: 241
Minimal:238
Average:1559,4888888888888
```

Рисунок 3.1 – приклад роботи програми

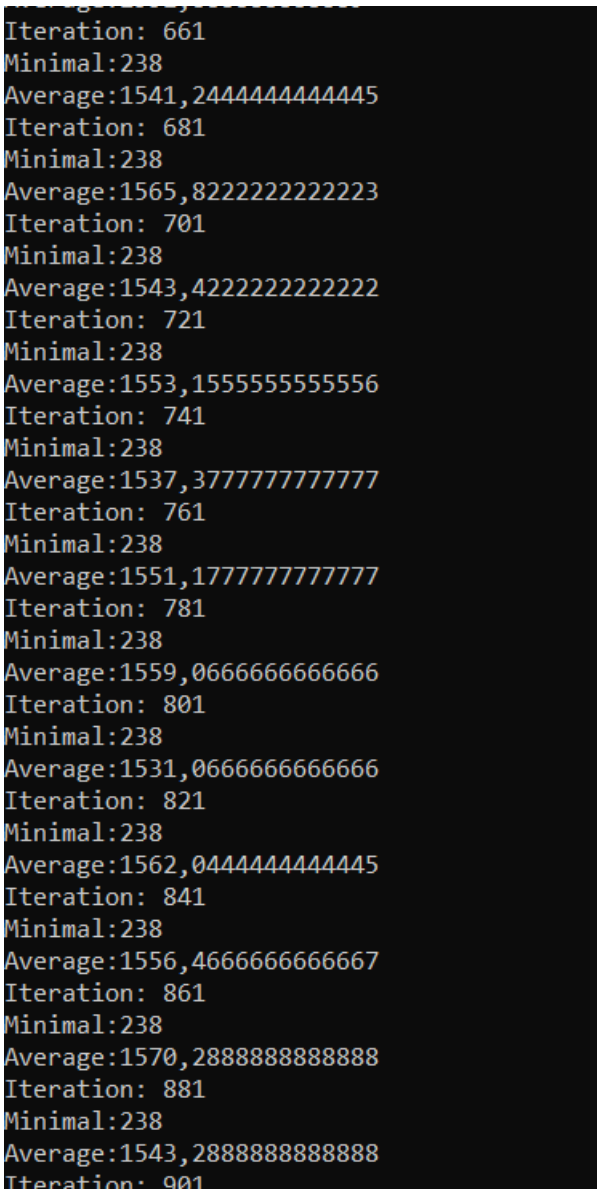


Рисунок 3.2 - приклад роботи програми

## 3.2 Тестування алгоритму

### 3.2.1 Значення цільової функції зі збільшенням кількості ітерацій

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

Табл 3.1

| Ітерація | ЦФ |
|---|---|
| 1 | 359 |
| 21 | 267 |
| 41 | 257 |
| 61 | 249 |
| 81 | 254 |
| 101 | 257 |
| 121 | 254 |
| 141 | 251 |
| 161 | 244 |
| 181 | 245 |
| 201 | 242 |
| 221 | 242 |
| 241 | 242 |
| 261 | 242 |
| 281 | 242 |
| 301 | 242 |
| 321 | 238 |
| 341 | 242 |
| 361 | 242 |
| 381 | 242 |
| 401 | 242 |
| 421 | 242 |
| 441 | 242 |
| 461 | 242 |
| 481 | 242 |
| 501 | 242 |
| 521 | 242 |
| 541 | 240 |
| 561 | 274 |
| 581 | 271 |
| 601 | 252 |
| 621 | 270 |
| 641 | 274 |
| 661 | 274 |

Подовження табл 3.1

| Ітерації | ЦФ |
|---|---|
| 681 | 245 |
| 701 | 245 |
| 721 | 245 |
| 741 | 245 |
| 761 | 245 |
| 781 | 242 |
| 801 | 245 |
| 821 | 245 |
| 841 | 271 |
| 861 | 245 |
| 881 | 245 |
| 901 | 245 |
| 921 | 269 |
| 941 | 263 |
| 961 | 245 |
| 981 | 245 |
| 1001 | 245 |

### 3.2.2 Графіки залежності розв'язку від числа ітерацій

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

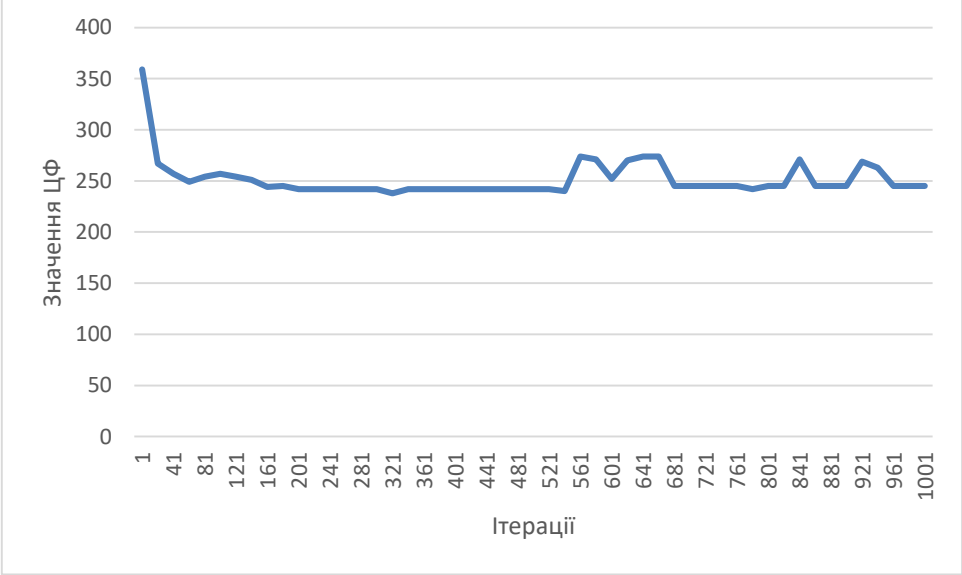


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

# ВИСНОВОК

Отож, у ході виконання лабораторної роботи було ознайомлено з варіаціями метаевристичних та вірогіднісних алгоритмів, зокрема було розглянуто мурашиний алогритм. Було виявлено його специфіку та особливості атрибутики. Під час виконання лабораторної роботи було проаналізовано даний алгоритм та спроектовано його на задачу комівояжера, реалізувавши його компільованою мовою C#. У ході виконання поставленого завдання проводилася фіксація значення цільової функції для кожної певної кількості ітерацій. Було практичним чином доведено, що час виконання алгоритму за довгострокового випробування позитивно корелюють із якістю кінцевого значення цільової функції. Набуто практичних навичок реалізації вірогіднісних алгоритмів.

## КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

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