**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**”

**Варіант 20**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-15, Плугатирьов Д.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 13](#_Toc51260925)

[Висновок 14](#_Toc51260926)

[Критерії оцінювання 15](#_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). |
| 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 (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 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 (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

using System.Text;

namespace TravelingSalesmanProblemLogic

{

    public class Ant

    {

        public bool IsElite { get; set; }

        private List<int> \_visitedVerticesNumbers = new List<int>();

        public Ant(int placementVertexNumber, bool isElite = false)

        {

            \_visitedVerticesNumbers.Add(placementVertexNumber);

            IsElite = isElite;

        }

        public void DeleteAllVerticesExceptInitial()

        {

            \_visitedVerticesNumbers.RemoveRange(1, \_visitedVerticesNumbers.Count - 1);

        }

        public int GetVisitedVerticesCount()

        {

            return \_visitedVerticesNumbers.Count;

        }

        public List<int> GetVerticesNumbers()

        {

            return ((int[])\_visitedVerticesNumbers.ToArray().Clone()).ToList();

        }

        /// <exception cref="ArgumentException"></exception>

        public void AddVisitedVerticeNumber(int verticeNumberToAdd)

        {

            if (IsVerticeVisited(verticeNumberToAdd))

            {

                throw new ArgumentException("Specified vertice is already visited");

            }

            \_visitedVerticesNumbers.Add(verticeNumberToAdd);

        }

        /// <exception cref="ArgumentException"></exception>

        public void AddVisitedVerticeNumber(AntGraphEdge visitedEdge)

        {

            if (IsVerticeVisited(visitedEdge.FirstVertex.Number)

                && IsVerticeVisited(visitedEdge.SecondVertex.Number))

            {

                throw new ArgumentException("Specified edge is already visited");

            }

            if (IsVerticeVisited(visitedEdge.FirstVertex.Number))

            {

                \_visitedVerticesNumbers.Add(visitedEdge.SecondVertex.Number);

            }

            else

            {

                \_visitedVerticesNumbers.Add(visitedEdge.FirstVertex.Number);

            }

        }

        public bool IsVerticeVisited(int verticeNumber)

        {

            return \_visitedVerticesNumbers.Contains(verticeNumber);

        }

        public override string ToString()

        {

            var builder = new StringBuilder();

            \_visitedVerticesNumbers.ForEach(v => builder.Append($" {v} "));

            return $"Is elite: {IsElite}\n"

                + $"Visited vertices' numbers: {builder.ToString()}\n\n";

        }

    }

}

using System.Collections;

namespace TravelingSalesmanProblemLogic

{

    public class AntGraphEdge : IEnumerable<Ant>

    {

        public GraphVertex FirstVertex { get; set; }

        public GraphVertex SecondVertex { get; set; }

        public int Length { get; set; }

        public double PheromonLevel { get; set; }

        private List<Ant> \_antsPassedDuringIteration = new List<Ant>();

        public AntGraphEdge(GraphVertex firstPlace, GraphVertex secondPlace)

        {

            FirstVertex = firstPlace;

            SecondVertex = secondPlace;

            Random random = new Random();

            PheromonLevel = random.NextDouble();

            if (PheromonLevel == 0)

            {

                PheromonLevel = 0.1;

            }

            int smallestLength = 1;

            int longestLength = 40;

            Length = random.Next(smallestLength, longestLength + 1);

        }

        public void RemovePassedAnts()

        {

            \_antsPassedDuringIteration.Clear();

        }

        public void AddPassedAnt(Ant passedAnt)

        {

            if (\_antsPassedDuringIteration.Contains(passedAnt))

            {

                throw new ArgumentException("Specified ant already exists in the list");

            }

            \_antsPassedDuringIteration.Add(passedAnt);

        }

        public override string ToString()

        {

            return $"{FirstVertex} <--{Length}--> {SecondVertex}";

        }

        public IEnumerator<Ant> GetEnumerator()

        {

            foreach (Ant ant in \_antsPassedDuringIteration)

            {

                yield return ant;

            }

        }

        IEnumerator IEnumerable.GetEnumerator()

        {

            return GetEnumerator();

        }

    }

}

namespace TravelingSalesmanProblemLogic

{

    public class AntMover

    {

        private EdgeSelector \_edgeSelector;

        private GraphOfSites \_graph;

        public AntMover(EdgeSelector edgeSelector, GraphOfSites graph)

        {

            \_edgeSelector = edgeSelector;

            \_graph = graph;

        }

        public void MoveAntToNextVertex(Ant antToMove)

        {

            if (antToMove.GetVisitedVerticesCount() != \_graph.GetVerticesCount())

            {

                var optimalEdge = \_edgeSelector

                    .AntSelectOptimalEdge(antToMove.IsElite, antToMove.GetVerticesNumbers().ToArray());

                antToMove.AddVisitedVerticeNumber(optimalEdge);

                optimalEdge.AddPassedAnt(antToMove);

            }

        }

    }

}

namespace TravelingSalesmanProblemLogic

{

    public class AntProblemSolver

    {

        private GraphOfSites \_graph;

        private EdgeSelector \_edgeSelector;

        private AntSpawner \_antSpawner;

        private AntMover \_antMover;

        private List<List<AntGraphEdge>> \_graphBypassesHistory = new List<List<AntGraphEdge>>();

        public int MinimalSolutionPrice { get; set; }

        public List<AntGraphEdge> MinimalSolutionPath { get; set; }

        public AntProblemSolver(GraphOfSites graph, EdgeSelector edgeSelector,

            AntSpawner antSpawner, AntMover antMover)

        {

            \_graph = graph;

            \_edgeSelector = edgeSelector;

            \_antSpawner = antSpawner;

            \_antMover = antMover;

            MinimalSolutionPath = GetMinimalSolutionPath();

            MinimalSolutionPrice = MinimalSolutionPath

                .Sum(e => e.Length);

        }

        /// <exception cref="InvalidOperationException"></exception>

        public List<AntGraphEdge> Solve(int iterationsCount = 1000)

        {

            if (\_antSpawner.GetAntsCount() == 0)

            {

                throw new InvalidOperationException("Ants are not spawned");

            }

            List<AntGraphEdge> shortestGraphBypass = null;

            FileHandler.WriteLine(DateTime.Now.ToString());

            FileHandler.WriteLine($"Default ants count: {\_antSpawner.DefaultAntsCount}, "

                + $"elite ants count: {\_antSpawner.EliteAntsCount}, iterations count: {iterationsCount}, "

                + $"vertices count: {\_graph.GetVerticesCount()}");

            for (var i = 0; i < iterationsCount; i++)

            {

                MoveAntsThroughWholeGraph();

                shortestGraphBypass = GetShortestGraphBypass();

                \_graphBypassesHistory.Add(shortestGraphBypass);

                foreach (AntGraphEdge edge in \_graph)

                {

                    \_graph.UpdatePheromonLevel(MinimalSolutionPrice, edge);

                }

                foreach (AntGraphEdge edge in \_graph)

                {

                    edge.RemovePassedAnts();

                }

                foreach (Ant ant in \_antSpawner)

                {

                    ant.DeleteAllVerticesExceptInitial();

                }

                if ((i + 1) % 20 == 0)

                {

                    FileHandler.WriteLine

                        ($"Iteration {i + 1}, "

                        + $"minimal length {shortestGraphBypass.Sum(e => e.Length)}");

                }

            }

            FileHandler.WriteLine("");

            return \_graphBypassesHistory

                .Find(bypass => bypass.Sum(e => e.Length) == \_graphBypassesHistory.Min(b => b.Sum(e => e.Length)));

        }

        private List<AntGraphEdge> GetShortestGraphBypass()

        {

            List<AntGraphEdge> result = null;

            int minimalPathLength = int.MaxValue;

            var antsVisitedVerticesNumbers = \_antSpawner.GetAntsVisitedVerticesNumbers();

            foreach (List<int> visitedVerticesNumbers in antsVisitedVerticesNumbers)

            {

                visitedVerticesNumbers.Add(visitedVerticesNumbers.First());

                var path = \_graph.GetEdgesWhichConnectVertices(visitedVerticesNumbers.ToArray());

                var pathLength = path.Sum(e => e.Length);

                if (pathLength < minimalPathLength)

                {

                    minimalPathLength = pathLength;

                    result = path;

                }

            }

            return result;

        }

        private void MoveAntsThroughWholeGraph()

        {

            foreach (Ant ant in \_antSpawner)

            {

                for (var j = 0; j < \_graph.GetVerticesCount() - 1; j++)

                {

                    \_antMover.MoveAntToNextVertex(ant);

                }

            }

        }

        public List<AntGraphEdge> GetMinimalSolutionPath()

        {

            int initialVertexNumber = 1;

            var visitedVerticesNumbers = new List<int>() { initialVertexNumber };

            var currentVertexNumber = initialVertexNumber;

            var result = new List<AntGraphEdge>();

            do

            {

                var previousVertexNumber = currentVertexNumber;

                var optimalTransition = \_edgeSelector

                    .GreedySelectOptimalEdge(visitedVerticesNumbers.ToArray());

                if (optimalTransition.FirstVertex.Number == currentVertexNumber)

                {

                    currentVertexNumber = optimalTransition.SecondVertex.Number;

                }

                else

                {

                    currentVertexNumber = optimalTransition.FirstVertex.Number;

                }

                visitedVerticesNumbers.Add(currentVertexNumber);

                result.Add(\_graph

                    .GetEdgeByEndsNumbers(previousVertexNumber, currentVertexNumber));

            }

            while (visitedVerticesNumbers.Count != \_graph.GetVerticesCount());

            var edgeOfStartAndEndVertices = \_graph.GetEdgeByEndsNumbers

                (visitedVerticesNumbers.Last(), initialVertexNumber);

            result.Add(edgeOfStartAndEndVertices);

            return result;

        }

    }

}

using System.Collections;

namespace TravelingSalesmanProblemLogic

{

    public class AntSpawner : IEnumerable<Ant>

    {

        public int DefaultAntsCount { get; }

        public int EliteAntsCount { get; }

        private GraphOfSites \_graph;

        private List<Ant> \_ants = new List<Ant>();

        public AntSpawner(GraphOfSites graph, int defaultAntsCount = 35, int eliteAntsCount = 10)

        {

            \_graph = graph;

            DefaultAntsCount = defaultAntsCount;

            EliteAntsCount = eliteAntsCount;

        }

        public List<List<int>> GetAntsVisitedVerticesNumbers()

        {

            var result = new List<List<int>>();

            foreach (Ant ant in \_ants)

            {

                result.Add(ant.GetVerticesNumbers());

            }

            return result;

        }

        public int GetAntsCount()

        {

            return \_ants.Count;

        }

        /// <exception cref="InvalidOperationException"></exception>

        public void SpawnAnts()

        {

            SpawnAnts(\_graph.NextVertexNumber, false);

            SpawnAnts(\_graph.NextVertexNumber, true);

        }

        /// <exception cref="InvalidOperationException"></exception>

        private void SpawnAnts(int maximalVertexNumber, bool isElite)

        {

            if (\_graph.GetVerticesCount() < DefaultAntsCount + EliteAntsCount)

            {

                throw new InvalidOperationException

                    ("The count of ants is bigger than count of graph vertices");

            }

            Random random = new Random();

            int antsCount;

            if (isElite)

            {

                antsCount = EliteAntsCount;

            }

            else

            {

                antsCount = DefaultAntsCount;

            }

            for (var i = 0; i < antsCount; i++)

            {

                var antIsSpawned = false;

                do

                {

                    var currentVertexNumber = random.Next(1, \_graph.GetVerticesCount() + 1);

                    if (\_graph.VertexExists(currentVertexNumber))

                    {

                        if (!\_ants.Exists(a => a.IsVerticeVisited(currentVertexNumber)))

                        {

                            var antToAdd = new Ant(currentVertexNumber, isElite);

                            \_ants.Add(antToAdd);

                            antIsSpawned = true;

                        }

                    }

                } while (!antIsSpawned);

            }

        }

        public IEnumerator<Ant> GetEnumerator()

        {

            foreach (Ant ant in \_ants)

            {

                yield return ant;

            }

        }

        IEnumerator IEnumerable.GetEnumerator()

        {

            return GetEnumerator();

        }

    }

}

namespace TravelingSalesmanProblemLogic

{

    public class EdgeSelector

    {

        public int AlphaParameter { get; } = 3;

        public int BetaParameter { get; } = 2;

        private GraphOfSites \_graph;

        public EdgeSelector(GraphOfSites graph)

        {

            \_graph = graph;

        }

        public AntGraphEdge AntSelectOptimalEdge(bool isAntElite, int[] visitedVerticesNumbers)

        {

            var lastVisitedVertexNumber = visitedVerticesNumbers.Last();

            var applicableEdges =

                \_graph.GetEdgesWithSpecifiedVertexExcept(lastVisitedVertexNumber,

                visitedVerticesNumbers);

            var transitionProbabilities = GetTransitionProbabilities(applicableEdges);

            if (isAntElite)

            {

                return transitionProbabilities.First

                    (p => p.Value == transitionProbabilities.Max(tp => tp.Value)).Key;

            }

            else

            {

                var random = new Random();

                var randomProbabilityNumber = random.NextDouble();

                double probabilitySum = 0;

                foreach (KeyValuePair<AntGraphEdge, double> edgeProbability

                    in transitionProbabilities)

                {

                    probabilitySum += edgeProbability.Value;

                    if (randomProbabilityNumber <= probabilitySum)

                    {

                        return edgeProbability.Key;

                    }

                }

                return transitionProbabilities.Last().Key;

            }

        }

        private Dictionary<AntGraphEdge, double> GetTransitionProbabilities

            (List<AntGraphEdge> applicableEdges)

        {

            var result = new Dictionary<AntGraphEdge, double>();

            var transitionProbabilityDenominator = GetTransitionProbabilityDenominatorValue(applicableEdges);

            foreach (AntGraphEdge edge in applicableEdges)

            {

                var selectionProbability = (Math.Pow(edge.PheromonLevel, AlphaParameter)

                    \* Math.Pow(1 / (double)edge.Length, BetaParameter)) / transitionProbabilityDenominator;

                result.Add(edge, selectionProbability);

            }

            return result;

        }

        private double GetTransitionProbabilityDenominatorValue(List<AntGraphEdge> edges)

        {

            double denominator = 0;

            foreach (AntGraphEdge edge in edges)

            {

                denominator +=

                    (Math.Pow(edge.PheromonLevel, AlphaParameter)

                        \* Math.Pow(1 / (double)edge.Length, BetaParameter));

            }

            return denominator;

        }

        public AntGraphEdge GreedySelectOptimalEdge(int[] visitedVerticesNumbers)

        {

            var lastVisitedVertexNumber = visitedVerticesNumbers.Last();

            var vertexEdges = \_graph

                .GetEdgesWithSpecifiedVertexExcept(lastVisitedVertexNumber,

                    visitedVerticesNumbers);

            return vertexEdges.First(e => e.Length == vertexEdges.Min(e => e.Length));

        }

    }

}

namespace TravelingSalesmanProblemLogic

{

    public static class FileHandler

    {

        public static string FileNameToWorkWith { get; set; } = "iterations\_data.txt";

        public static void WriteLine(string text)

        {

            using (var output = new StreamWriter(FileNameToWorkWith, true))

            {

                output.WriteLine(text);

            }

        }

    }

}

using System.Collections;

using System.Text;

namespace TravelingSalesmanProblemLogic

{

    public class GraphOfSites : IEnumerable<AntGraphEdge>

    {

        private List<GraphVertex> \_vertices = new List<GraphVertex>();

        private List<AntGraphEdge> \_edges = new List<AntGraphEdge>();

        public double PheromonEvaporationRate { get; } = 0.7;

        public int NextVertexNumber { get; private set; } = 1;

        public GraphOfSites(int initialVerticesCount)

        {

            for (var i = 0; i < initialVerticesCount; i++)

            {

                AddVertex();

            }

            CreateEdgesBetweenAllVertices();

        }

        public void UpdatePheromonLevel(int minimalSolutionPrice,

            AntGraphEdge edgeToUpdatePheromoneLevelOn)

        {

            var adjustedPheromonSum =

                GetAdjustedPheromonSum(minimalSolutionPrice, edgeToUpdatePheromoneLevelOn);

            edgeToUpdatePheromoneLevelOn.PheromonLevel = (1 - PheromonEvaporationRate)

                        \* edgeToUpdatePheromoneLevelOn.PheromonLevel + adjustedPheromonSum;

        }

        private double GetAdjustedPheromonSum(int minimalSolutionPrice, AntGraphEdge edge)

        {

            double result = 0;

            foreach (Ant ant in edge)

            {

                result += (double)minimalSolutionPrice

                    / GetEdgesWhichConnectVertices(ant.GetVerticesNumbers().ToArray())

                    .Sum(e => e.Length);

            }

            return result;

        }

        public void CreateEdgesBetweenAllVertices()

        {

            for (var i = 1; i < \_vertices.Count; i++)

            {

                for (var j = i + 1; j < \_vertices.Count + 1; j++)

                {

                    TryAddEdge(i, j);

                }

            }

        }

        public List<AntGraphEdge> GetEdgesWhichConnectVertices(int[] verticesNumbers)

        {

            var result = new List<AntGraphEdge>();

            for (var i = 0; i < verticesNumbers.Length - 1; i++)

            {

                result.Add(GetEdgeByEndsNumbers(verticesNumbers[i], verticesNumbers[i + 1]));

            }

            return result;

        }

        public void AddVertex()

        {

            \_vertices.Add(new GraphVertex(NextVertexNumber++));

        }

        public bool VertexExists(int vertexNumber)

        {

            return \_vertices.Exists(v => v.Number == vertexNumber);

        }

        public List<AntGraphEdge> GetEdgesWithSpecifiedVertex(int vertexNumber)

        {

            return \_edges.FindAll(e => (e.FirstVertex.Number == vertexNumber)

                || (e.SecondVertex.Number == vertexNumber)).ToList();

        }

        public List<AntGraphEdge> GetEdgesWithSpecifiedVertexExcept(int vertexNumber,

            int[] exceptionalVerticesNumbers)

        {

            return GetEdgesWithSpecifiedVertex(vertexNumber).Except

                    (GetEdgesWithSpecifiedVertex(vertexNumber)

                    .Where(e => ((e.FirstVertex.Number == vertexNumber)

                        && (exceptionalVerticesNumbers.Contains(e.SecondVertex.Number)))

                        || ((e.SecondVertex.Number == vertexNumber)

                        && (exceptionalVerticesNumbers.Contains(e.FirstVertex.Number))))).ToList();

        }

        public int GetVerticesCount()

        {

            return \_vertices.Count;

        }

        public GraphVertex FindVertexByNumber(int vertexNumber)

        {

            return \_vertices.Find(v => v.Number == vertexNumber);

        }

        public bool ExistsEdgeWithVertex(int vertexNumber)

        {

            return \_edges.Exists(e =>

                (e.FirstVertex.Number == vertexNumber) || (e.SecondVertex.Number == vertexNumber));

        }

        public void TryAddEdge(int firstVertexNumber, int secondVertexNumber)

        {

            if (!EdgeExists(firstVertexNumber, secondVertexNumber))

            {

                var firstVertexOfEdge = \_vertices.Find(v => v.Number == firstVertexNumber);

                var secondVertexOfEdge = \_vertices.Find(v => v.Number == secondVertexNumber);

                if (firstVertexOfEdge is not null && secondVertexOfEdge is not null)

                {

                    \_edges.Add(new AntGraphEdge(firstVertexOfEdge, secondVertexOfEdge));

                }

            }

        }

        public bool EdgeExists(int firstVertexNumber, int secondVertexNumber)

        {

            return \_edges.Exists(e => (e.FirstVertex.Number == firstVertexNumber)

                && (e.SecondVertex.Number == secondVertexNumber));

        }

        public AntGraphEdge GetEdgeByEndsNumbers(int firstVertexNumber, int secondVertexNumber)

        {

            return \_edges.Find(e =>

                ((e.FirstVertex.Number == firstVertexNumber)

                    && (e.SecondVertex.Number == secondVertexNumber))

                || ((e.FirstVertex.Number == secondVertexNumber)

                    && (e.SecondVertex.Number == firstVertexNumber)));

        }

        public override string ToString()

        {

            var result = new StringBuilder();

            foreach (AntGraphEdge edge in \_edges)

            {

                result.Append($" {edge} |");

            }

            return result.ToString();

        }

        public IEnumerator<AntGraphEdge> GetEnumerator()

        {

            foreach (AntGraphEdge edge in \_edges)

            {

                yield return edge;

            }

        }

        IEnumerator IEnumerable.GetEnumerator()

        {

            return GetEnumerator();

        }

    }

}

namespace TravelingSalesmanProblemLogic

{

    public class GraphVertex

    {

        public int Number { get; }

        public GraphVertex(int number)

        {

            Number = number;

        }

        public override string ToString()

        {

            return Number.ToString();

        }

    }

}

namespace TravelingSalesmanProblemRunner

{

    public class DataCapturer

    {

        public static int CaptureInitialVerticesCount(int antsCount)

        {

            int initialVerticesCount = 200;

            bool verticesAreCaptured;

            do

            {

                verticesAreCaptured = true;

                System.Console.Write("Enter the initial count of vertices (required 200): ");

                try

                {

                    initialVerticesCount = Convert.ToInt32(Console.ReadLine());

                    if (initialVerticesCount < antsCount)

                    {

                        throw new ArgumentOutOfRangeException

                            (nameof(initialVerticesCount), "The initial vertices count can't be less than ants count");

                    }

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("You entered not a number");

                    verticesAreCaptured = false;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    verticesAreCaptured = false;

                }

            } while (!verticesAreCaptured);

            return initialVerticesCount;

        }

        public static int CaptureIterationsCount()

        {

            int iterationsCount = 1000;

            bool iterationsAreCaptured;

            do

            {

                iterationsAreCaptured = true;

                System.Console.Write("Enter the count of iterations (required 1000): ");

                try

                {

                    iterationsCount = Convert.ToInt32(Console.ReadLine());

                    if (iterationsCount < 20)

                    {

                        throw new ArgumentOutOfRangeException

                            (nameof(iterationsCount), "The iterations' count can't be less than 20");

                    }

                }

                catch (FormatException)

                {

                    System.Console.WriteLine("You entered not a number");

                    iterationsAreCaptured = false;

                }

                catch (ArgumentOutOfRangeException ex)

                {

                    System.Console.WriteLine(ex.Message);

                    iterationsAreCaptured = false;

                }

            } while (!iterationsAreCaptured);

            return iterationsCount;

        }

    }

}

namespace TravelingSalesmanProblemRunner;

using System.Diagnostics;

using TravelingSalesmanProblemLogic;

public class Program

{

    public static void Main(string[] args)

    {

        int defaultAntsCount = 35;

        int eliteAntsCount = 10;

        int initialVerticesCount = DataCapturer.CaptureInitialVerticesCount(defaultAntsCount + eliteAntsCount);

        var graph = new GraphOfSites(initialVerticesCount);

        var edgeSelector = new EdgeSelector(graph);

        var antSpawner = new AntSpawner(graph, defaultAntsCount, eliteAntsCount);

        var antMover = new AntMover(edgeSelector, graph);

        var solver = new AntProblemSolver(graph, edgeSelector, antSpawner, antMover);

        // System.Console.WriteLine(graph);

        var iterationsCount = DataCapturer.CaptureIterationsCount();

        System.Console.WriteLine($"Minimal solution price - {solver.MinimalSolutionPrice}");

        try

        {

            antSpawner.SpawnAnts();

        }

        catch (InvalidOperationException ex)

        {

            System.Console.WriteLine(ex.Message);

            return;

        }

        var timer = new Stopwatch();

        timer.Start();

        System.Console.WriteLine("Please, wait. Solving traveling salesman problem...");

        var shortestGraphBypass = solver.Solve(iterationsCount);

        timer.Stop();

        System.Console.WriteLine($"Solving took {timer.Elapsed} time");

        System.Console.WriteLine($"The shortest path length: {shortestGraphBypass.Sum(e => e.Length)}");

        System.Console.WriteLine($"The shortest graph bypass:");

        foreach (AntGraphEdge edge in shortestGraphBypass)

        {

            System.Console.Write($" {edge} |");

        }

    }

}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

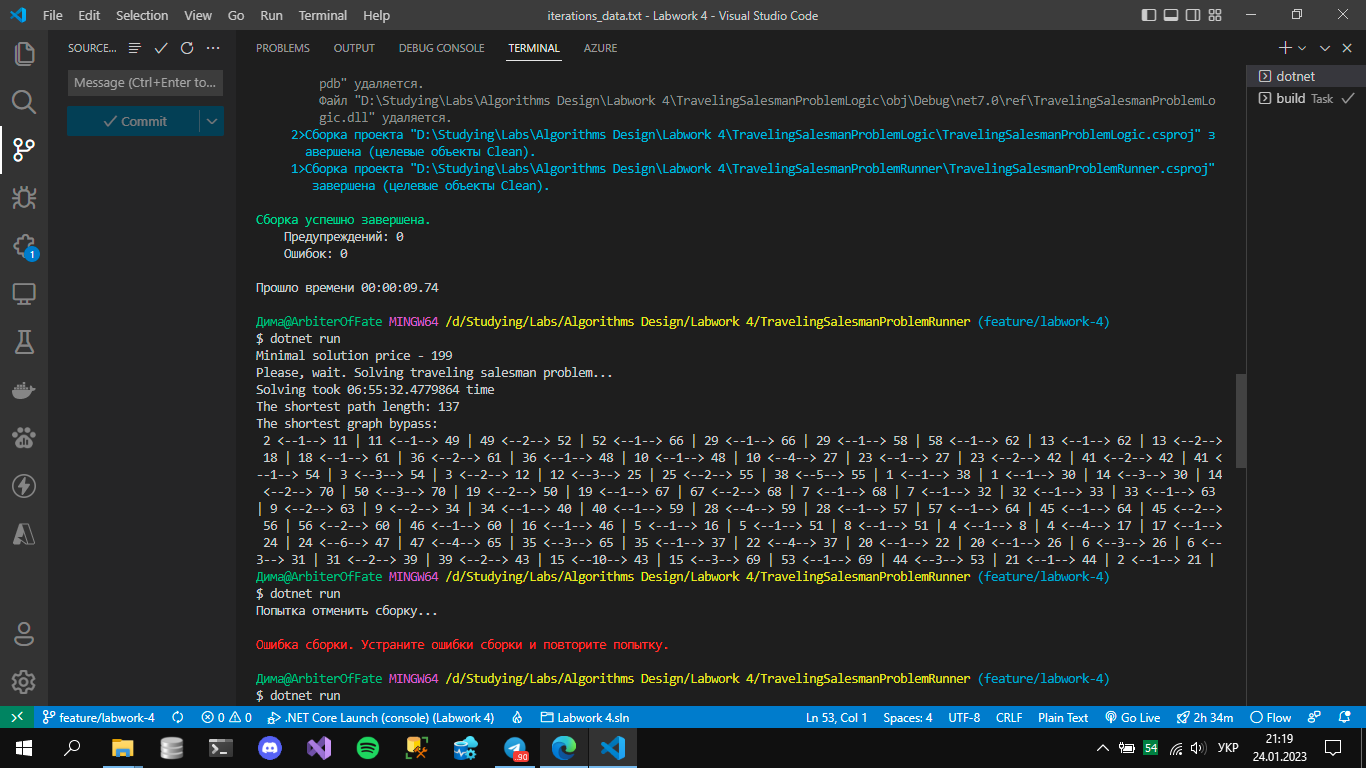


Рисунок 3.1 – результат роботи програми за 70 вершин та 1000 ітерацій

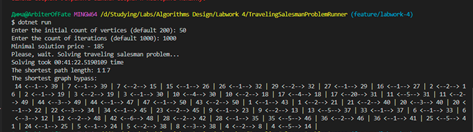


Рисунок 3.2 – результат роботи програми за 50 вершин та 1000 ітерацій

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| Номер ітерації | Значення цільової функції |
| 20 | 132 |
| 40 | 135 |
| 60 | 126 |
| 80 | 126 |
| 100 | 126 |
| 120 | 126 |
| 140 | 122 |
| 160 | 122 |
| 180 | 122 |
| 200 | 126 |
| 220 | 126 |
| 240 | 126 |
| 260 | 126 |
| 280 | 126 |
| 300 | 126 |
| 320 | 126 |
| 340 | 126 |
| 360 | 122 |
| 380 | 126 |
| 400 | 126 |
| 420 | 122 |
| 440 | 122 |
| 460 | 126 |
| 480 | 133 |
| 500 | 126 |
| 520 | 126 |
| 540 | 122 |
| 560 | 126 |
| 580 | 122 |
| 600 | 126 |
| 620 | 126 |
| 640 | 126 |
| 660 | 126 |
| 680 | 126 |
| 700 | 117 |
| 720 | 126 |
| 740 | 126 |
| 760 | 126 |
| 780 | 126 |
| 800 | 126 |
| 820 | 133 |
| 840 | 126 |
| 860 | 122 |
| 880 | 126 |
| 900 | 122 |
| 920 | 126 |
| 940 | 122 |
| 960 | 126 |
| 980 | 126 |
| 1000 | 126 |

Таблиця 3.1 – значення цільових функцій результату роботи програми за 50 вершин та 1000 ітерацій

### 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 даної лабораторної роботи мною було розроблено програму для вирішення проблеми комівояжера мурашиним алгоритмом.

Особливість цього алгоритму полягає в імітації способу пошуку найкоротшого шляху між вершинами графу, яким наділила мурах природа в результаті багаторічного процесу еволюції.

Я зрозумів, що основою будь-якого мурашиного алгоритму є створення мурах, їх пересування з метою знайти рішення та оновлення феромону в кінці ітерації. Насправді, ще можна застосовувати вкінці додаткові дії, але я цього не робив.

Мною було модифіковано цей алгоритм шляхом розміщення мурах на різних випадкових вершинах (максимум 1 мураха на вершину). Особливістю елітних мурах у мене став вибір дуги із найбільшою ймовірністю (без генерації додаткового випадкового числа).

Ідеальну ціну рішення в задачі було знайдено завдяки обранню випадкової створеної вершини та застосуванню алгоритму жадібного пошуку.

Оскільки потужностей моєї машини не вистачило для пошуку рішення задачі із 200 вершинами, я виконав його на 70 та 50 вершинах відповідно.

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%.