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 Паламарчук Олександр* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.Н.* _____
(прізвище, ім'я, по батькові)

Київ 2022

Зміст

Лабораторна робота №5

Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2

Варіант 21

1. Мета

Мета роботи – вивчити основні підходи розробки метаевристичних 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 степенем 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.

3. Виконання

3.1. Покроковий алгоритм

Алгоритм мурашиної колонії (Vertex: start, Vertex: end):

Якщо start = end, то повернути start;

Для i = 0, поки i < iterations:

Для j = 0, поки j < numberOfAnts:

paths[j] <- findPath(start, end);

update();

Кінець

Кінець

path <- найкращий шлях із масиву paths[], який містить найменшу довжину

Повернути path;

Кінець алгоритму;

Алгоритм findPath(start, end):

Повернути findPathRecursive(start, end);

Кінець алгоритму;

Алгоритм findPathRecursive(curr, end):

visited[] <- curr;

path[] <- curr;

Якщо вершина curr є рішенням, то повернути path, та індикатор result;

adjacentVertices[] <- отримати усі сусідні, ще нерозглянуті вершини.

Якщо нерозглянутих сусідніх вершин немає, то повернути індикатор невдачі.

nextVertices[] <- отримати ймовірності переходу з поточної вершини до кожної вершини з adjacentVertices[].

Поки nextVertices[] не пустий:

random <- згенерувати випадкове число в діапазоні від 0 до 1.

nextVertex <- Опираючись на random обрати наступну вершину.

indicator <- findPathRecursive(nextVertex, end).

Якщо indicator = result, то повернути path та індикатор result

adjacentVertices[].remove(nextVertex)

перерахувати ймовірності переходу.

Кінець

Повернути індикатор невдачі.

Кінець алгоритму findPathRecursive;

Алгоритм update():

Випарувати феромон на кожному зв’язку.

Додати додатковий феромон, до кожного зв’язку.

Кінець алгоритму update;

3.2. Програмна реалізація алгоритму.

3.2.1. Програмний код алгоритму.

```
package org.example.graph.algorithm.exception;

public class RouteAlgorithmException extends Exception {
    public RouteAlgorithmException() {
    }

    public RouteAlgorithmException(String message) {
        super(message);
    }

    public RouteAlgorithmException(String message, Throwable cause) {
        super(message, cause);
    }

    public RouteAlgorithmException(Throwable cause) {
        super(cause);
    }
}
package org.example.graph.algorithm.exception;

public class RouteAlgorithmFactoryException extends Exception {
    public RouteAlgorithmFactoryException() {
    }

    public RouteAlgorithmFactoryException(String message) {
        super(message);
    }

    public RouteAlgorithmFactoryException(String message, Throwable cause) {
        super(message, cause);
    }

    public RouteAlgorithmFactoryException(Throwable cause) {
        super(cause);
    }
}
package org.example.graph.algorithm.exporter.factory.exception;

public class GraphExporterFactoryException extends Exception {
```

```
    public GraphExporterFactoryException() {
    }

    public GraphExporterFactoryException(String message) {
        super(message);
    }

    public GraphExporterFactoryException(String message, Throwable cause) {
        super(message, cause);
    }

    public GraphExporterFactoryException(Throwable cause) {
        super(cause);
    }
}
```

```
package org.example.graph.algorithm.exporter.factory.exception;

public class GraphExporterFactoryImplException extends
GraphExporterFactoryException {
    public GraphExporterFactoryImplException() {
    }

    public GraphExporterFactoryImplException(String message) {
        super(message);
    }

    public GraphExporterFactoryImplException(String message, Throwable cause) {
        super(message, cause);
    }

    public GraphExporterFactoryImplException(Throwable cause) {
        super(cause);
    }
}
```

```
package org.example.graph.algorithm.exporter.factory.impl;

import org.example.Constants;
import org.example.graph.algorithm.exporter.GraphExporter;
import org.example.graph.algorithm.exporter.factory.GraphExporterFactory;
import
org.example.graph.algorithm.exporter.factory.exception.GraphExporterFactoryImplE
xception;

import java.io.FileReader;
import java.io.IOException;
import java.util.Properties;

import static org.example.Constants.OUTPUT_FILE_NAME;

public class GraphExporterFactoryImpl extends GraphExporterFactory {

    private final String fileName;

    public GraphExporterFactoryImpl() throws GraphExporterFactoryImplException {
        Properties props = new Properties();
        try {
            props.load(new FileReader(Constants.APP_PROPS_FILE_NAME));
            this.fileName = props.getProperty(OUTPUT_FILE_NAME);
        } catch (IOException e) {
            throw new GraphExporterFactoryImplException(e);
        }
    }

    @Override
    public GraphExporter newGraphExporter() {
        return new GraphExporter(fileName);
```

```
    }
}
package org.example.graph.algorithm.exporter.factory;

import org.example.graph.algorithm.exporter.GraphExporter;
import
org.example.graph.algorithm.exporter.factory.exception.GraphExporterFactoryExcep
tion;
import
org.example.graph.algorithm.exporter.factory.impl.GraphExporterFactoryImpl;

public abstract class GraphExporterFactory {

    public static GraphExporterFactory newInstance() throws
GraphExporterFactoryException {
        return new GraphExporterFactoryImpl();
    }

    public abstract GraphExporter newGraphExporter();

}
package org.example.graph.algorithm.exporter;

import com.google.common.graph.EndpointPair;
import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.LinkedList;
import java.util.List;
import java.util.Objects;
import java.util.Set;

import static java.util.Objects.requireNonNull;

public class GraphExporter {

    private MutableValueGraph<Vertex, Edge> graph;
    private Set<Vertex> path;
    private List<String> lines;
    private final String fileName;

    public GraphExporter(String fileName) {
        this.fileName = requireNonNull(fileName);
        this.lines = new LinkedList<>();
    }

    public void export() {
        build();
        write();
    }

    private void build() {
        List<String> lines = new LinkedList<>();
        lines.add("graph G {");
        for (Vertex v : graph.nodes()) {
            if (path.contains(v))
                lines.add("  %d [style=filled, color=blue4, fillcolor=yellow,
penwidth=3]".formatted(v.getId()));
            else
                lines.add("  %d [style=filled, color=blue4, fillcolor=white,
penwidth=3]".formatted(v.getId()));
        }
```

```
        lines.add("");

        for (EndpointPair<Vertex> endpointPair : graph.edges())
            lines.add("  %d -- %d [label=\"%d\", penwidth=%s,
color=aquamarine3]".formatted(
                    endpointPair.nodeU().getId(),
                    endpointPair.nodeV().getId(),
                    graph.edgeValue(endpointPair).orElseThrow().getDistance(),

formatPheromone(graph.edgeValue(endpointPair).orElseThrow().getPheromone())));

        lines.add("}");
        this.lines = lines;
    }

    private String formatPheromone(double pheromone) {
        if (Double.compare(pheromone, 10.0) > 0)
            return String.valueOf(10);
        else if (Double.compare(pheromone, 0.5) < 0)
            return String.valueOf(0.5);
        else {
            String result = String.valueOf(pheromone).replace(',', '.');
            return result.substring(0, 7);
        }
    }

    private void write() {
        try {
            Files.write(Path.of(fileName), lines);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public void setGraph(MutableValueGraph<Vertex, Edge> graph) {
        this.graph = Objects.requireNonNull(graph);
    }

    public void setPath(Set<Vertex> path) {
        this.path = path;
    }
}
package org.example.graph.algorithm.factory;

import org.example.graph.algorithm.RouteAlgorithm;
import org.example.graph.algorithm.exception.RouteAlgorithmFactoryException;
import org.example.graph.parser.GraphParser;

public interface RouteAlgorithmFactory {
    RouteAlgorithm createRouteAlgorithm() throws RouteAlgorithmFactoryException;
    void setGraphParser(GraphParser graphParser);
}
package org.example.graph.algorithm.impl.ant.exception;

import org.example.graph.algorithm.exception.RouteAlgorithmException;

public class AntAlgorithmException extends RouteAlgorithmException {
    public AntAlgorithmException() {
    }

    public AntAlgorithmException(String message) {
        super(message);
    }

    public AntAlgorithmException(String message, Throwable cause) {
```

```
        super(message, cause);
    }

    public AntAlgorithmException(Throwable cause) {
        super(cause);
    }
}
package org.example.graph.algorithm.impl.ant.exception;

import org.example.graph.algorithm.exception.RouteAlgorithmFactoryException;

public class AntAlgorithmFactoryException extends RouteAlgorithmFactoryException
{
    public AntAlgorithmFactoryException() {
    }

    public AntAlgorithmFactoryException(String message) {
        super(message);
    }

    public AntAlgorithmFactoryException(String message, Throwable cause) {
        super(message, cause);
    }

    public AntAlgorithmFactoryException(Throwable cause) {
        super(cause);
    }
}
package org.example.graph.algorithm.impl.ant.factory;

import org.example.graph.algorithm.RouteAlgorithm;
import org.example.graph.algorithm.factory.RouteAlgorithmFactory;
import org.example.graph.algorithm.impl.ant.AntAlgorithm;
import
org.example.graph.algorithm.impl.ant.exception.AntAlgorithmFactoryException;
import org.example.graph.parser.GraphParser;

import java.io.FileReader;
import java.io.IOException;
import java.util.Properties;

import static org.example.Constants.*;

public class AntAlgorithmFactory implements RouteAlgorithmFactory {

    private GraphParser graphParser;
    private AntAlgorithmFactory() {}

    public static AntAlgorithmFactory newInstance() {
        return new AntAlgorithmFactory();
    }

    @Override
    public RouteAlgorithm createRouteAlgorithm() throws
AntAlgorithmFactoryException {
        Properties props = new Properties();
        AntAlgorithm algorithm;
        try {
            props.load(new FileReader(APP_PROPS_FILE_NAME));
            algorithm = new AntAlgorithm(
                    Integer.parseInt(props.getProperty(BETA)),
                    Integer.parseInt(props.getProperty(ALFA)),
                    Integer.parseInt(props.getProperty(L_MIN)),
                    Integer.parseInt(props.getProperty(NUMBER_OF_ANTS)),
                    Integer.parseInt(props.getProperty(ITERATIONS)),
                    Double.parseDouble(props.getProperty(P)),
```

```
                    graphParser);
        } catch (IOException e) {
            throw new AntAlgorithmFactoryException(e);
        }
        return algorithm;
    }

    @Override
    public void setGraphParser(GraphParser graphParser) {
        this.graphParser = graphParser;
    }
}
package org.example.graph.algorithm.impl.ant;

import com.google.common.graph.EndpointPair;
import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;
import org.example.graph.algorithm.Path;
import org.example.graph.algorithm.RouteAlgorithm;
import org.example.graph.algorithm.impl.ant.exception.AntAlgorithmException;
import org.example.graph.parser.GraphParser;
import org.example.graph.parser.strategy.exception.GraphParserStrategyException;

import java.util.*;
import java.util.stream.Collectors;

import static java.util.function.Function.identity;
import static org.example.graph.algorithm.impl.ant.Indicator.FAILURE;
import static org.example.graph.algorithm.impl.ant.Indicator.RESULT;

public class AntAlgorithm extends RouteAlgorithm {
    private Vertex start;
    private Vertex terminal;
    private MutableValueGraph<Vertex, Edge> graph;
    private List<Path> paths;
    private final double p;
    private final int alfa;
    private final int beta;
    private final int lMin;
    private final int iterations;
    private final int numberOfAnts;

    public AntAlgorithm(
            int alfa,
            int beta,
            int lMin,
            int numberOfAnts,
            int iterations,
            double p,
            GraphParser parser) {
        super(parser);
        this.alfa = alfa;
        this.beta = beta;
        this.lMin = lMin;
        this.numberOfAnts = numberOfAnts;
        this.iterations = iterations;
        this.p = p;
    }

    @Override
    public List<Vertex> buildRoute(Vertex start, Vertex terminal) throws
AntAlgorithmException {
        this.start = Objects.requireNonNull(start);
        this.terminal = Objects.requireNonNull(terminal);
```

```
        if (start.equals(terminal))
            return List.of(start);

        buildGraph();
        checkPoints();
        startToMove();

        Path best = findBest();
        Stack<Vertex> bestRoute = best.getPath();

        System.out.println("Length: " + best.getLength());

        return bestRoute;
    }

    private Path findBest() {
        return paths.stream().min(Comparator.comparing(Path::getLength))
                .orElseThrow();
    }

    @Override
    public MutableValueGraph<Vertex, Edge> getGraph() {
        return graph;
    }

    private void buildGraph() throws AntAlgorithmException {
        try {
            this.graph = graphParser.parse();
        } catch (GraphParserStrategyException e) {
            throw new AntAlgorithmException(e);
        }
    }

    private void checkPoints() throws AntAlgorithmException {
        Set<Vertex> vertices = graph.nodes();

        String message = "Invalid %s point";

        if (!vertices.contains(start))
            throw new AntAlgorithmException(message.formatted("start"));

        if (!vertices.contains(terminal))
            throw new AntAlgorithmException(message.formatted("terminal"));
    }

    private void startToMove() {
        for (int i = 0; i < iterations; i++) {
            findPaths();
            updateEdges();
        }
    }

    private void findPaths() {
        this.paths = new LinkedList<>();
        for (int i = 0; i < numberOfAnts; i++) paths.add(findPath());
    }

    private Path findPath() {
        Stack<Vertex> path = new Stack<>();
        Indicator i = findPathRecursive(start, new HashSet<>(), path);

        if (i.equals(FAILURE))
            throw new IllegalStateException("Cannot find path");

        return Path.valueOf(path, graph);
    }
```

```
    private Indicator findPathRecursive(Vertex curr, HashSet<Vertex> visited,
Stack<Vertex> path) {
        visited.add(curr);
        path.push(curr);

        if (curr.equals(terminal))
            return RESULT;

        Set<Vertex> notVisitedAdjacentVertices =
getNotVisitedAdjacentVertices(curr, visited);

        if (notVisitedAdjacentVertices.isEmpty())
            return FAILURE;

        Map<Vertex, Double> transitionProbabilities
                = getTransitionProbabilities(curr, notVisitedAdjacentVertices);
        while (transitionProbabilities.size() > 0) {
            Indicator i = findPathRecursive(takeStep(transitionProbabilities),
visited, path);

            if (i.equals(RESULT))
                return i;

            notVisitedAdjacentVertices.remove(path.pop());
            transitionProbabilities = getTransitionProbabilities(curr,
notVisitedAdjacentVertices);
        }

        return FAILURE;
    }

    private Vertex takeStep(Map<Vertex, Double> transitionProbabilities) {
        double random = calculateRandomDouble();
        Vertex last = null;
        for (Map.Entry<Vertex, Double> e : transitionProbabilities.entrySet()) {
            random -= e.getValue();
            last = e.getKey();
            if (random <= 0)
                break;
        }
        return last;
    }

    private Set<Vertex> getNotVisitedAdjacentVertices(Vertex curr, Set<Vertex>
visited) {
        return graph.adjacentNodes(curr).stream()
                .filter(v -> !visited.contains(v))
                .collect(Collectors.toSet());
    }

    private Map<Vertex, Double> getTransitionProbabilities(Vertex curr,
Set<Vertex> adjacentVertices) {
        double sumOfWish = getSumOfWish(curr, adjacentVertices);
        return adjacentVertices.stream()
                .collect(Collectors.toMap(
                        identity(), vertex -> calculateProbability(getEdge(curr,
vertex), sumOfWish)));
    }

    public double getSumOfWish(Vertex curr, Set<Vertex> adjacentVertices) {
        return adjacentVertices.stream()
                .map(vertex -> getEdge(curr, vertex))
                .mapToDouble(e -> e.calculateWish(alfa, beta))
                .sum();
    }
```

```
    private void updateEdges() {
        evaporatePheromone();
        for (Path path : paths) {
            double extraPheromone = path.getExtraPheromone(lMin);
            for (EndpointPair<Vertex> endpointPair : path.getEdges()) {
                Edge edge = getEdge(endpointPair.nodeV(),
                        endpointPair.nodeU());
                edge.addPheromone(extraPheromone);
            }
        }
    }

    private void evaporatePheromone() {
        graph.edges().stream()
                .map(e -> getEdge(e.nodeV(), e.nodeU()))
                .forEach(e -> e.evaporatePheromone(p));
    }

    private double calculateRandomDouble() {
        int randomInt = new Random().nextInt(1001);
        return randomInt / 1000.0;
    }

    private double calculateProbability(Edge edge, double sum) {
        return edge.calculateWish(alfa, beta) / sum;
    }

    private Edge getEdge(Vertex v1, Vertex v2) {
        return graph.edgeValue(v1, v2).orElseThrow();
    }
}
```

```
package org.example.graph.algorithm.impl.ant;

public enum Indicator {
    FAILURE,
    RESULT
}
```

```
package org.example.graph.algorithm;

import org.example.graph.Vertex;

import java.util.Comparator;

public class Entry implements Comparable<Entry> {
    private final Vertex vertex;
    private final int distance;
    private final double pheromone;

    private Entry(Vertex vertex, double pheromone, int distance) {
        this.vertex = vertex;
        this.pheromone = pheromone;
        this.distance = distance;
    }

    public static Entry from(Vertex vertex, double pheromone, int distance) {
        return new Entry(vertex, pheromone, distance);
    }

    public Vertex getVertex() {
        return vertex;
    }

    public double getPheromone() {
        return pheromone;
    }
```

```
    public int getDistance() {
        return distance;
    }

    @Override
    public int compareTo(Entry o) {
        return comparator().compare(this, o);
    }

    public Comparator<Entry> comparator() {
        return Comparator.comparing(Entry::getPheromone)
                .reversed().thenComparing(Entry::getDistance);
    }
}
package org.example.graph.algorithm;

import com.google.common.graph.EndpointPair;
import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;

import java.util.LinkedList;
import java.util.List;
import java.util.Stack;

import static com.google.common.graph.EndpointPair.*;

public class Path {
    private final MutableValueGraph<Vertex, Edge> graph;
    private final List<EndpointPair<Vertex>> edges;
    private final Stack<Vertex> path;
    private int length;

    private Path(MutableValueGraph<Vertex, Edge> graph, Stack<Vertex> path) {
        this.path = path;
        this.edges = new LinkedList<>();
        this.graph = graph;
    }

    public void addEdge(Vertex v1, Vertex v2) {
        edges.add(unordered(v1, v2));
        length += graph.edgeValue(v1, v2)
                .orElseThrow()
                .getDistance();
    }

    public List<EndpointPair<Vertex>> getEdges() {
        return edges;
    }

    @SuppressWarnings("unchecked")
    public static Path valueOf(Stack<Vertex> path, MutableValueGraph<Vertex,
Edge> graph) {
        Path p = new Path(graph, (Stack<Vertex>) path.clone());

        Vertex curr = path.pop(), next;
        while (!path.empty()) {
            next = path.pop();
            p.addEdge(curr, next);
            curr = next;
        }

        return p;
    }
```

```
    public int getLength() {
        return length;
    }

    public double getExtraPheromone(int lMin) {
        return lMin / (double) length;
    }

    public Stack<Vertex> getPath() {
        return path;
    }
}
```

```
package org.example.graph.algorithm;

import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;
import org.example.graph.algorithm.exception.RouteAlgorithmException;
import org.example.graph.parser.GraphParser;

import java.util.List;

import static java.util.Objects.requireNonNull;


public abstract class RouteAlgorithm {
    protected GraphParser graphParser;
    public RouteAlgorithm(GraphParser graphParser) {
        this.graphParser = requireNonNull(graphParser);
    }
    public abstract List<Vertex> buildRoute(Vertex start, Vertex terminal)
throws RouteAlgorithmException;
    public abstract MutableValueGraph<Vertex, Edge> getGraph();
}
```

```
package org.example.graph.parser.strategy.exception;

public class GraphParserStrategyException extends Exception {
    public GraphParserStrategyException() {
    }

    public GraphParserStrategyException(String message) {
        super(message);
    }

    public GraphParserStrategyException(String message, Throwable cause) {
        super(message, cause);
    }

    public GraphParserStrategyException(Throwable cause) {
        super(cause);
    }
}
```

```
package org.example.graph.parser.strategy.factory.exception;

public class GraphParserStrategyFactoryException extends Exception {
    public GraphParserStrategyFactoryException() {
    }

    public GraphParserStrategyFactoryException(String message) {
        super(message);
    }

    public GraphParserStrategyFactoryException(String message, Throwable cause)
{
        super(message, cause);
    }
```

```
    public GraphParserStrategyFactoryException(Throwable cause) {
        super(cause);
    }
}
```

```
package org.example.graph.parser.strategy.factory.exception;

public class XmlGraphParserStrategyFactoryException extends
GraphParserStrategyFactoryException {
    public XmlGraphParserStrategyFactoryException() {
    }

    public XmlGraphParserStrategyFactoryException(String message) {
        super(message);
    }

    public XmlGraphParserStrategyFactoryException(String message, Throwable
cause) {
        super(message, cause);
    }

    public XmlGraphParserStrategyFactoryException(Throwable cause) {
        super(cause);
    }
}
```

```
package org.example.graph.parser.strategy.factory.impl;

import org.example.Constants;
import org.example.graph.parser.strategy.GraphParserStrategy;
import org.example.graph.parser.strategy.factory.GraphParserStrategyFactory;
import
org.example.graph.parser.strategy.factory.exception.XmlGraphParserStrategyFactor
yException;
import org.example.graph.parser.strategy.impl.xml.XmlGraphParserStrategy;

import java.io.FileReader;
import java.io.IOException;
import java.util.Properties;

public class XmlGraphParserStrategyFactory extends GraphParserStrategyFactory {

    @Override
    public GraphParserStrategy newStrategy() throws
XmlGraphParserStrategyFactoryException {
        String fileName;
        Properties props = new Properties();
        try {
            props.load(new FileReader(Constants.APP_PROPS_FILE_NAME));
            fileName = props.getProperty(Constants.GRAPH_SOURCE_FILE_NAME);
        } catch (IOException e) {
            throw new XmlGraphParserStrategyFactoryException(e);
        }
        return new XmlGraphParserStrategy(fileName);
    }
}
```

```
package org.example.graph.parser.strategy.factory;

import org.example.graph.parser.strategy.GraphParserStrategy;
import
org.example.graph.parser.strategy.factory.exception.GraphParserStrategyFactoryEx
ception;
import
org.example.graph.parser.strategy.factory.impl.XmlGraphParserStrategyFactory;

public abstract class GraphParserStrategyFactory {
```

```
    public static GraphParserStrategyFactory newInstance() {
        return new XmlGraphParserStrategyFactory();
    }
    public abstract GraphParserStrategy newStrategy() throws
GraphParserStrategyFactoryException;
}
package org.example.graph.parser.strategy.impl.xml.exception;

import org.example.graph.parser.strategy.exception.GraphParserStrategyException;

public class XmlGraphParserStrategyException extends
GraphParserStrategyException {
    public XmlGraphParserStrategyException() {
    }

    public XmlGraphParserStrategyException(String message) {
        super(message);
    }

    public XmlGraphParserStrategyException(String message, Throwable cause) {
        super(message, cause);
    }

    public XmlGraphParserStrategyException(Throwable cause) {
        super(cause);
    }
}
package org.example.graph.parser.strategy.impl.xml;

import com.google.common.graph.MutableValueGraph;
import com.google.common.graph.ValueGraphBuilder;
import org.example.graph.Edge;
import org.example.graph.Vertex;
import org.example.graph.parser.strategy.GraphParserStrategy;
import
org.example.graph.parser.strategy.impl.xml.exception.XmlGraphParserStrategyExcep
tion;
import org.xml.sax.SAXException;

import javax.xml.namespace.QName;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.events.Attribute;
import javax.xml.stream.events.EndElement;
import javax.xml.stream.events.StartElement;
import javax.xml.stream.events.XMLEvent;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import javax.xml.validation.Validator;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import static javax.xml.XMLConstants.W3C_XML_SCHEMA_NS_URI;
import static org.example.Constants.XSD_FILE_NAME;
import static org.example.Utils.parseIntValueFrom;

public class XmlGraphParserStrategy extends GraphParserStrategy {

    private final XMLInputFactory factory;
    private final List<Vertex> twoAdjacentVertices;
```

```
    private int distance;
    private MutableValueGraph<Vertex, Edge> valueGraph;

    public XmlGraphParserStrategy(String fileName) {
        super(fileName);
        factory = XMLInputFactory.newInstance();
        twoAdjacentVertices = new ArrayList<>(2);
    }

    @Override
    public void parse() throws XmlGraphParserStrategyException {
        validate(fileName);
        build(fileName);
    }

    private void build(String fileName) throws XmlGraphParserStrategyException {
        try {
            XMLEventReader reader = factory.createXMLEventReader(new
FileReader(fileName));
            while (reader.hasNext()) {
                XMLEvent event = reader.nextEvent();
                if (event.isStartElement()) {
                    handleStartElement(event.asStartElement());
                } else if (event.isEndElement()) {
                    handleEndElement(event.asEndElement());
                }
            }
        } catch (XMLStreamException | FileNotFoundException e) {
            throw new XmlGraphParserStrategyException(e);
        }
    }

    private void handleStartElement(StartElement startElement) throws
XmlGraphParserStrategyException {
        XmlTag tag = XmlTag.from(startElement.getName());
        switch (tag) {
            case GRAPH -> this.valueGraph = ValueGraphBuilder
                    .undirected().allowsSelfLoops(false).build();
            case EDGE -> {
                twoAdjacentVertices.clear();
                distance = getEdgeValue(startElement);
            }
            case VERTEX -> twoAdjacentVertices.add(createVertex(startElement));
        }
    }

    private int getEdgeValue(StartElement startElement) {
        Attribute attribute =
startElement.getAttributeByName(QName.valueOf("value"));
        return parseIntValueFrom(attribute);
    }

    private void handleEndElement(EndElement endElement) throws
XmlGraphParserStrategyException {
        XmlTag tag = XmlTag.from(endElement.getName());
        if (tag == XmlTag.EDGE)
            valueGraph.putEdgeValue(
                    twoAdjacentVertices.get(0), twoAdjacentVertices.get(1),
Edge.valueOf(distance));
    }

    private Vertex createVertex(StartElement startElement) {
        Attribute attribute =
startElement.getAttributeByName(QName.valueOf("id"));
        return Vertex.valueOf(parseIntValueFrom(attribute));
    }
```

```
    private void validate(String fileName) throws
XmlGraphParserStrategyException {
        SchemaFactory schemaFactory =
SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI);
        try {
            Schema schema = schemaFactory.newSchema(new File(XSD_FILE_NAME));
            Validator validator = schema.newValidator();
            validator.validate(new StreamSource(fileName));
        } catch (SAXException | IOException e) {
            throw new XmlGraphParserStrategyException(e);
        }
    }

    @Override
    public MutableValueGraph<Vertex, Edge> getValueGraph() {
        return valueGraph;
    }
}
```

```
package org.example.graph.parser.strategy.impl.xml;

import
org.example.graph.parser.strategy.impl.xml.exception.XmlGraphParserStrategyExcep
tion;

import javax.xml.namespace.QName;

public enum XmlTag {
    GRAPH("graph"),
    VERTEX("vertex"),
    EDGE("edge");

    private final String name;
    XmlTag(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public static XmlTag from(QName name) throws XmlGraphParserStrategyException
{
        return switch (name.getLocalPart()) {
            case "graph" -> GRAPH;
            case "vertex" -> VERTEX;
            case "edge" -> EDGE;
            default -> throw new XmlGraphParserStrategyException("Unknown tag
name");
        };
    }
}
```

```
package org.example.graph.parser.strategy;

import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;
import org.example.graph.parser.strategy.exception.GraphParserStrategyException;

import static java.util.Objects.requireNonNull;

public abstract class GraphParserStrategy {
    protected String fileName;

    public GraphParserStrategy(String fileName) {
        this.fileName = requireNonNull(fileName);
    }
```

```
    public abstract void parse() throws GraphParserStrategyException;
    public abstract MutableValueGraph<Vertex, Edge> getValueGraph();
}
```

```
package org.example.graph.parser;

import com.google.common.graph.MutableValueGraph;
import org.example.graph.Edge;
import org.example.graph.Vertex;
import org.example.graph.parser.strategy.GraphParserStrategy;
import org.example.graph.parser.strategy.exception.GraphParserStrategyException;

public class GraphParser {

    private GraphParserStrategy strategy;

    public GraphParser(GraphParserStrategy strategy) {
        this.strategy = strategy;
    }

    public void setStrategy(GraphParserStrategy strategy) {
        this.strategy = strategy;
    }

    public MutableValueGraph<Vertex, Edge> parse() throws
GraphParserStrategyException {
        strategy.parse();
        return strategy.getValueGraph();
    }

}
```

```
package org.example.graph;

import static java.lang.Math.pow;

public class Edge {
    private double pheromone;
    private final int distance;
    private final double proximity;
    private Edge(double pheromone, double proximity, int distance) {
        this.pheromone = pheromone;
        this.proximity = proximity;
        this.distance = distance;
    }

    public static Edge valueOf(int distance) {
        return new Edge(0.2, 1 / (double) distance, distance);
    }

    public double getPheromone() {
        return pheromone;
    }

    public int getDistance() {
        return distance;
    }

    public double getProximity() {
        return proximity;
    }

    public void evaporatePheromone(double k) {
        this.pheromone *= k;
    }

    public double calculateWish(int alfa, int beta) {
        return pow(pheromone, alfa) * pow(proximity, beta);
```

```
    }

    public void addPheromone(double extraPheromone) {
        this.pheromone += extraPheromone;
    }

    public void setPheromone(double pheromone) {
        this.pheromone = pheromone;
    }

    @Override
    public String toString() {
        final StringBuffer sb = new StringBuffer("Edge{");
        sb.append("pheromone=").append(pheromone);
        sb.append(", distance=").append(distance);
        sb.append(", proximity=").append(proximity);
        sb.append('}');
        return sb.toString();
    }
}
```

```
package org.example.graph;

import lombok.Setter;

@Setter
public class Vertex {
    private final int id;

    private Vertex(int id) {
        this.id = id;
    }

    public static Vertex valueOf(int id) {
        return new Vertex(id);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Vertex vertex)) return false;

        return id == vertex.id;
    }

    public int getId() {
        return id;
    }

    @Override
    public int hashCode() {
        return id;
    }

    @Override
    public String toString() {
        return String.valueOf(id);
    }
}
```

```
package org.example;

public class Constants {
    public static final String P = "p";
    public static final String ALFA = "alfa";
    public static final String BETA = "beta";
    public static final String L_MIN = "lMin";
    public static final String ITERATIONS = "iterations";
```

```
    public static final String NUMBER_OF_ANTS = "number.of.ants";
    public static final String OUTPUT_FILE_NAME = "output.file.name";
    public static final String GRAPH_SOURCE_FILE_NAME =
"graph.source.file.name";
    public static final String XSD_FILE_NAME = "src/main/resources/graph.xsd";
    public static final String APP_PROPS_FILE_NAME =
"src/main/resources/app.properties";
}
package org.example;

import org.example.graph.Vertex;
import org.example.graph.algorithm.RouteAlgorithm;
import org.example.graph.algorithm.exception.RouteAlgorithmException;
import org.example.graph.algorithm.exception.RouteAlgorithmFactoryException;
import org.example.graph.algorithm.exporter.GraphExporter;
import org.example.graph.algorithm.exporter.factory.GraphExporterFactory;
import
org.example.graph.algorithm.exporter.factory.exception.GraphExporterFactoryExcep
tion;
import org.example.graph.algorithm.factory.RouteAlgorithmFactory;
import org.example.graph.algorithm.impl.ant.factory.AntAlgorithmFactory;
import org.example.graph.parser.GraphParser;
import org.example.graph.parser.strategy.GraphParserStrategy;
import org.example.graph.parser.strategy.factory.GraphParserStrategyFactory;
import
org.example.graph.parser.strategy.factory.exception.GraphParserStrategyFactoryEx
ception;

import java.util.HashSet;
import java.util.InputMismatchException;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try {
            GraphParserStrategyFactory strategyFactory =
GraphParserStrategyFactory.newInstance();
            GraphParserStrategy graphParserStrategy;
            graphParserStrategy = strategyFactory.newStrategy();
            GraphParser graphParser = new GraphParser(graphParserStrategy);

            RouteAlgorithmFactory factory = AntAlgorithmFactory.newInstance();
            factory.setGraphParser(graphParser);

            String action;
            do {
                try {
                    System.out.print("Write start point: ");
                    Vertex start = Vertex.valueOf(new
Scanner(System.in).nextInt());
                    System.out.print("Write terminal point: ");
                    Vertex terminal = Vertex.valueOf(new
Scanner(System.in).nextInt());
                    RouteAlgorithm routeAlgorithm =
factory.createRouteAlgorithm();
                    List<Vertex> path = routeAlgorithm.buildRoute(start,
terminal);
                    System.out.println("Result route: " + path);
                    System.out.print("You want to export graph? [yes/no]: ");
                    String choice = new Scanner(System.in).nextLine();
                    if (choice.equalsIgnoreCase("yes")) {
                        GraphExporterFactory exporterFactory =
GraphExporterFactory.newInstance();
                        GraphExporter graphExporter =
exporterFactory.newGraphExporter();
```

```
                        graphExporter.setGraph(routeAlgorithm.getGraph());
                        graphExporter.setPath(new HashSet<>(path));
                        System.out.println("Exporting...");
                        graphExporter.export();
                        System.out.println("Done!");
                    }
                } catch (InputMismatchException e) {
                    System.out.println("Invalid input");
                } catch (RouteAlgorithmException
                         | GraphExporterFactoryException e) {
                    System.out.println(e.getMessage());
                }
                System.out.print("You want to continue? [yes/no]: ");
                action = new Scanner(System.in).nextLine();
            } while (action.equalsIgnoreCase("yes"));
        } catch (GraphParserStrategyFactoryException
                 | RouteAlgorithmFactoryException e) {
            e.printStackTrace();
        }
    }
}
```

```
package org.example;

import java.util.LinkedList;
import java.util.List;

public class Statistics {
    private final List<Double> extraPheromone = new LinkedList<>();
    private final List<Integer> distances = new LinkedList<>();
    private final List<Double> wishes = new LinkedList<>();

    public void addExtraPheromone(double extraPheromone) {
        this.extraPheromone.add(extraPheromone);
    }

    public void addDistances(List<Integer> distances) {
        this.distances.addAll(distances);
    }

    public void addWish(double wish) {
        this.wishes.add(wish);
    }

    private double getAverageExtraPheromone() {
        return extraPheromone.stream().mapToDouble(e -> e).average().orElse(0);
    }

    private double getAverageWish() {
        return wishes.stream().mapToDouble(w -> w).average().orElse(0);
    }

    private double getAverageDistance() {
        return distances.stream().mapToInt(d -> d).average().orElse(0);
    }

    @Override
    public String toString() {
        final StringBuffer sb = new StringBuffer("Statistics{");
        sb.append("averageExtraPheromone=").append(getAverageExtraPheromone());
        sb.append(", averageDistance=").append(getAverageDistance());
        sb.append(", averageWish=").append(getAverageWish());
        sb.append('}');
        return sb.toString();
    }
}
```

```
package org.example;

import javax.xml.stream.events.Attribute;

public class Utils {

    public static int parseIntValueFrom(Attribute attribute) {
        return Integer.parseInt(attribute.getValue());
    }

}
```

3.2.2. Приклад роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
C:\Users\sasha\.jdks\openjdk-18.0.1\bin\java.exe ...
Write start point: 5
Write terminal point: 8
Length: 199
Result route: [5, 268, 206, 181, 176, 8]
You want to export graph? [yes/no]: no
You want to continue? [yes/no]: no

Process finished with exit code 0
```

Рисунок 3.1

```
C:\Users\sasha\.jdks\openjdk-18.0.1\bin\java.exe ...
Write start point: 7
Write terminal point: 299
Length: 214
Result route: [7, 289, 247, 105, 163, 296, 19, 299]
You want to export graph? [yes/no]: no
You want to continue? [yes/no]: no

Process finished with exit code 0
```

Рисунок 3.2

3.3. Тестування алгоритму.

Таблиця 3.1 – Залежність рішення від параметра α.

Таблиця 3.1

Ціна рішення	Параметр α
200	1
242	2
273	3

296	4
305	5

Рисунок 3.3 – графік залежності рішення від параметра α.

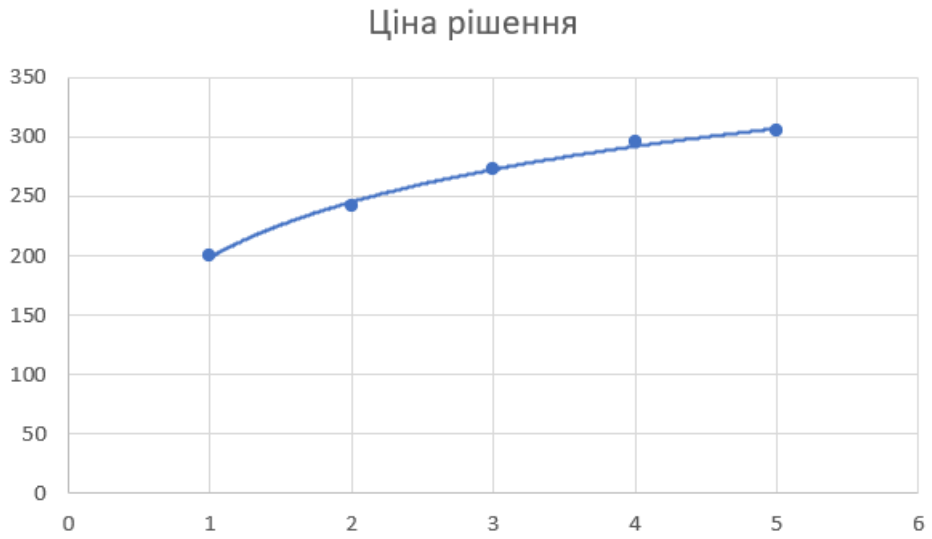

Рисунок 3.3

Таблиця 3.2 – Залежність рішення від параметра β.

Таблиця 3.2

Ціна рішення	Параметр β
242	1
204	2
240	3
269	4
248	5

Рисунок 3.4 – графік залежності рішення від параметра β.

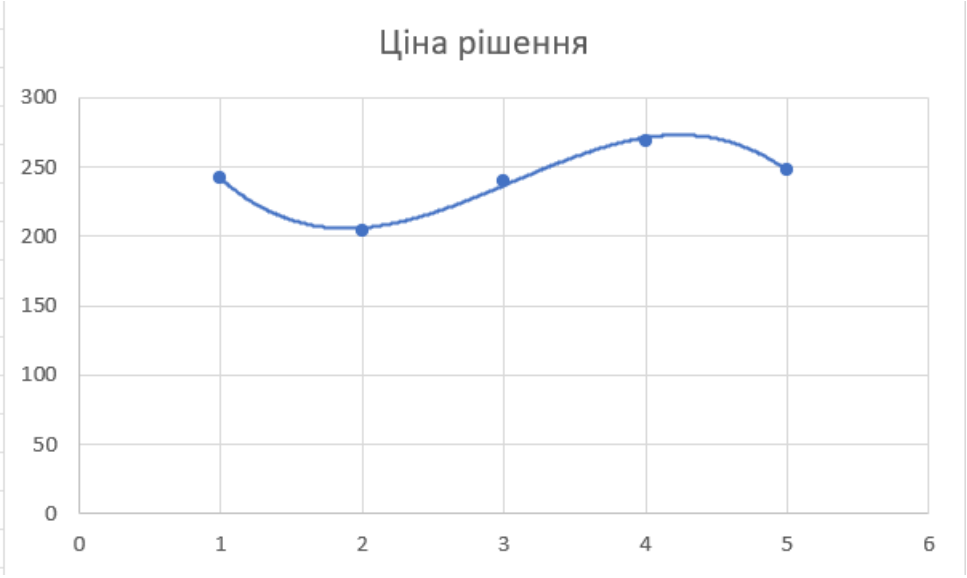

Рисунок 3.4

Таблиця 3.3 – Залежність рішення від параметра $lMin$.

Таблиця 3.3

Ціна рішення	Параметр $lMin$
236	10
205	20
200	30
216	40
203	50

Рисунок 3.5 – графік залежності рішення від параметра $lMin$.

Рисунок 3.5

Таблиця 3.4 – Залежність рішення від параметра p.

Таблиця 3.4

Ціна рішення	Параметр p
232	0.1
222	0.2
241	0.3
212	0.4
206	0.5
214	0.6
196	0.7
192	0.8
187	0.9

Рисунок 3.6 – графік залежності рішення від параметра p.

Рисунок 3.6

Таблиця 3.5 – Залежність рішення від параметра "кількість мурах".

Таблиця 3.5

Ціна рішення	Параметр "кількість мурах"
421	6
283	12
225	18
200	24

204	30

Рисунок 3.7 – графік залежності рішення від параметра "кількість мурах".

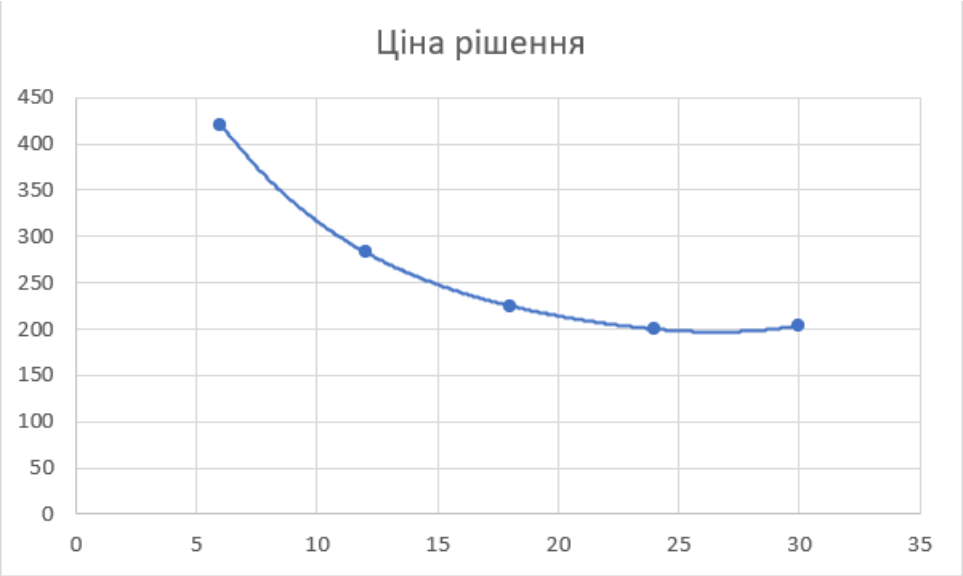

Рисунок 3.7

Таблиця 3.6 – маршрути з однієї вершини.

Таблиця 3.6

Номер початкової вершини	Номер кінцевої вершини	Маршрут
1	299	[1, 257, 299]
24	8	[24, 89, 73, 93, 176, 8]
76	100	[76, 271, 224, 166, 274, 9, 62, 258, 246, 6, 100]
34	87	[34, 163, 105, 94, 32, 87]
98	156	[98, 99, 248, 132, 238, 117, 294, 43, 229, 156]

4. Висновок

На лабораторній роботі було вивчено основні підходи розробки метаевристичних алгоритмів для типових прикладних задач, а саме алгоритм мурашиної колонії, та було опрацьовано методологію підбору прийнятних параметрів алгоритму. Було спроектовано та розроблено програмне забезпечення згідно із варіантом і детально описано кроки роботи алгоритму. Було проведено тестування алгоритму в ході якого ми дослідили такі параметри:

$\alpha, \beta, lMin, p,$ "кількість мурах", а також за допомогою алгоритма було побудовано шляхи від початкових до кінцевих заданих вершин. За допомогою тестування ми вияснили, що при збільшенні параметра α мурахи більше реагують на запах феромону, при збільшенні параметра β мурахи більше звертають увагу на близькість іншої вершини, параметр p впливає на те, як сильно випаровується феромон, параметр $lMin$ впливає на кількість феромону, що буде додано на конкретну ділянку а при збільшенні кількості мурах алгоритм хоч і працює довше, але вірогідність знайти наближене до оптимального, або оптимальне рішення зростає.