ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Вирішення задачі комівояжера за допомогою генетичного алгоритму

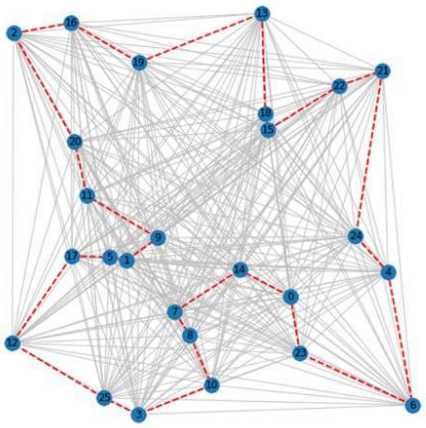
Виконав:
ст. гр. СПм-22-2
Онипченко А.О.

Науковий керівник
доц. каф. ЕОМ
Іващенко Г.С.

Актуальність проблеми

На сьогоднішній день одна з найпоширеніших задач теорії графів, задача комівояжера, залишається актуальною через її широке застосування в сферах логістики, організації виробничих процесів, вирішення телекомунікаційних проблем, при плануванні подорожей та в багатьох інших областях.

Головна мета задачі полягає в знаходженні найвигіднішого маршруту, який проходить через всі вузли у графі по одному разу та повертається у вихідну точку.



Існуючі методи вирішення задачі

- ✓ Генетичні алгоритми
- ✓ Метод вставки
- ✓ Метод найближчого сусіда
- ✓ Метод імітації відпалу
- ✓ Мурашиний алгоритм

3

Постановка задачі

Метою кваліфікаційної роботи є реалізація та дослідження генетичних алгоритмів для вирішення задачі комівояжера.

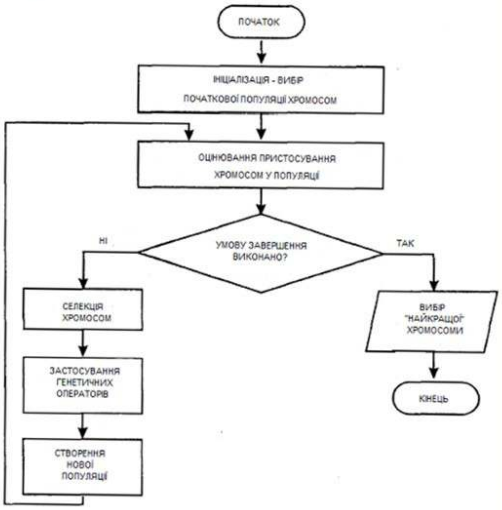
Етапи дослідження

- ❑ проаналізувати існуючі варіанти генетичного алгоритму;
- ❑ реалізувати генетичний алгоритм для вирішення задачі комівояжера;
- ❑ дослідити вплив використання різних варіантів кросовера та налаштувань параметрів ГА на результати рішення задачі;
- ❑ дослідити вплив використання поєднання генетичного та жадібного алгоритмів на результати рішення задачі.

4

Генетичні алгоритми

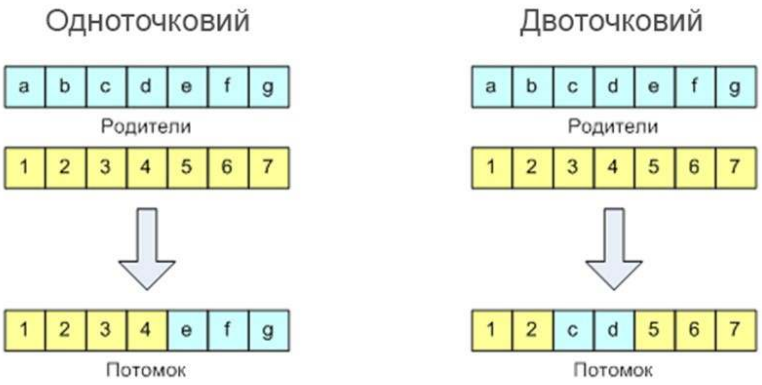
Генетичні алгоритми – адаптивні методи пошуку, що використовуються для вирішення завдань оптимізації та пошуку. В їх основі лежать ідеї біологічної еволюції та природного відбору і використовуються для знаходження найкращих рішень у задачах, де існує велика кількість можливих варіантів.



5

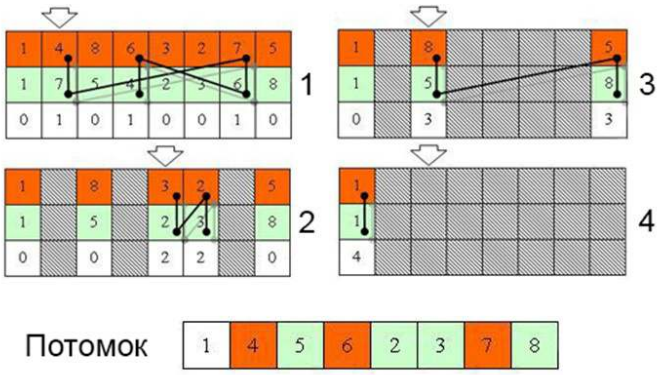
Варіанти кросовера

В даній роботи використано 3 варіанти кросовера, такі як одноточковий, двоточковий та циклічний кросовер.



6

Варіанти кросовера



Приклад роботи циклічного кросовера.

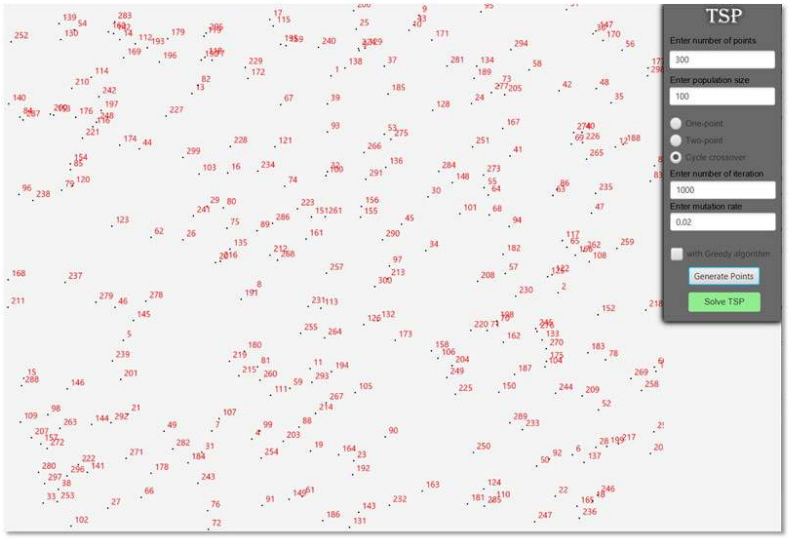
7

Перелік використаних технологій

- Мова програмування Java;
- Платформа JavaFX;
- SceneBuilder;
- IntelliJ IDEA Community Edition 2023.

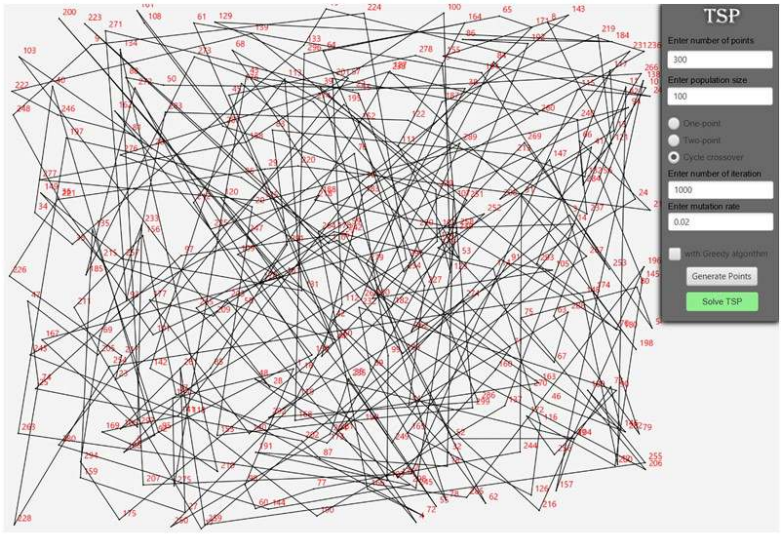
8

Процес роботи застосунку



9

Процес роботи застосунку



10

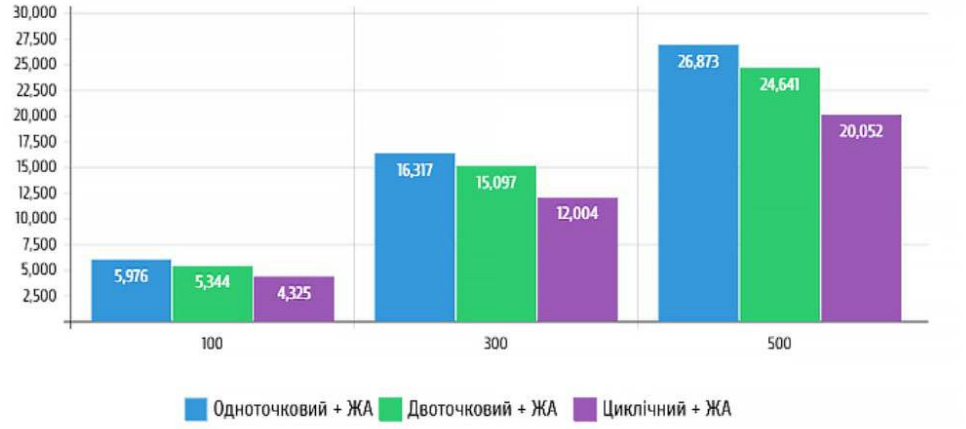
Результати досліджень

Дослідження впливу обраного варіанту кросовера на вирішення задачі

11

Результати досліджень

Дослідження роботи генетичного алгоритму у поєднанні із жадібним алгоритмом



12

Результати досліджень

Дослідження впливу кількості ітерацій

13

Результати досліджень

Дослідження впливу вірогідності мутації

Вірогідність мутації	Критерії ефективності	Кількість вершин			Кросовер
		200	350	500	
0,02	Час, мс	5364	14433	14266	Одноточковий
	Відстань, мм	52199,1	106692,2	110722,8	
0,5	Час, мс	5072	14133	13689	
	Відстань, мм	57453,1	106651,4	106388,5	
1,0	Час, мс	4983	14229	14324	
	Відстань, мм	61070,8	112733,8	108467,1	
0,02	Час, мс	933	927	864	Циклічний
	Відстань, мм	98168,4	97547,4	100473,5	
0,5	Час, мс	801	798	812	
	Відстань, мм	77479,6	74787,9	76391,5	
1,0	Час, мс	957	886	924	
	Відстань, мм	79872,0	76857,6	76959,1	

14

Висновки

В ході кваліфікаційної роботи було проведено аналіз можливості вирішення задачі комівояжера шляхом використання різних видів кросоверів генетичного алгоритму таких, як одноточковий, двоточковий та циклічний кросовер. Досліджено використання жадібного алгоритму у поєднанні з генетичним алгоритмом для покращення шуканого рішення.

Для проведення аналізу ефективності обраних варіантів кросоверів генетичного алгоритму було реалізовано програмний застосунок на основі платформи JavaFX на мові програмування Java.

Дослідження проведені на графах різних розмірностей та для різних розмірів популяції. Проведено аналіз впливу кількості ітерацій та ймовірності мутації на якість знайденого рішення.

Проведено аналіз отриманих результатів та визначення можливостей використання реалізованих генетичних алгоритмів для покращення пошуку ефективного рішення та зниження витрат у практичних сценаріях.

Результати роботи представлені у рамках одинадцятої міжнародної науково-технічної конференції «Проблеми інформатизації».

ДОДАТОК Б

Вихідний код програмного забезпечення

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class TSPGeneticAlgorithmApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {
        FXMLLoader loader = new
FXMLLoader(getClass().getResource("design.fxml"));
        Parent root = loader.load();
        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}


import javafx.fxml.FXML;
import javafx.scene.control.RadioButton;

import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class TSPGeneticAlgorithmModel {
    private List<Point2D> points;
    private List<Individual> population;
    private RadioButton onePointRadioButton;
    private RadioButton twoPointRadioButton;
    private RadioButton cycleCrossoverRadioButton;
    private boolean useGreedyAlgorithm;
    private int populationSize;
    private int numOfIteration;
    private double mutationRate;
    private int elitismCount = 5;
    private int greedyPopulationSize = 10;

    public TSPGeneticAlgorithmModel(List<Point2D> points, int
```

```
populationSize,int numOfIteration, double mutationRate,
RadioButton onePointRadioButton, RadioButton
twoPointRadioButton, RadioButton cycleCrossoverRadioButton,
boolean useGreedyAlgorithm) {
        this.points = points;
        this.populationSize=populationSize;
        this.numOfIteration=numOfIteration;
        this.mutationRate=mutationRate;
        this.population = initializePopulation(points.size());
        this.onePointRadioButton = onePointRadioButton;
        this.twoPointRadioButton = twoPointRadioButton;
        this.cycleCrossoverRadioButton =
cycleCrossoverRadioButton;
        this.useGreedyAlgorithm = useGreedyAlgorithm;
    }


    private List<Individual> initializePopulation(int size) {
        List<Individual> population = new ArrayList<>();
        for (int i = 0; i < populationSize; i++) {
            List<Point2D> randomPath = new ArrayList<>(points);
            Collections.shuffle(randomPath);
            population.add(new Individual(randomPath));
        }
        return population;
    }

    public void evolve() {
        if (onePointRadioButton.isSelected()) {
            System.out.println("One-point");
        } else if (twoPointRadioButton.isSelected()) {
            System.out.println("Two-point");
        } else if (cycleCrossoverRadioButton.isSelected()) {
            System.out.println("Uniform");
        } else {
            throw new IllegalStateException("No crossover option
selected");
        }
        System.out.println("Number of
iteration"+numOfIteration);
        System.out.println("Mutation Rate"+mutationRate);
        if(useGreedyAlgorithm){
            System.out.println("with Greedy Algorithm");
        }
        for (int i = 0; i < numOfIteration; i++) {
List<Individual> newPopulation = new
ArrayList<>(populationSize);
            List<Individual> elites = getElites();
            newPopulation.addAll(elites);

            if (useGreedyAlgorithm) {
                List<Individual> greedyPopulation =
population.subList(0, greedyPopulationSize);
```

```
                for (Individual greedyIndividual :
greedyPopulation) {
                    List<Point2D> greedyPath =
greedyAlgorithm(greedyIndividual.getPath());
                    newPopulation.add(new
Individual(greedyPath));
                }
            }

            while (newPopulation.size() < populationSize) {
                Individual parent1 = selectParent();
                Individual parent2 = selectParent();
                List<Point2D> childPath;

                if (onePointRadioButton.isSelected()) {
                    childPath = crossover(parent1, parent2);
                } else if (twoPointRadioButton.isSelected()) {
                    childPath = twoPointCrossover(parent1,
parent2);
                } else if
(cycleCrossoverRadioButton.isSelected()) {
                    childPath = cycleCrossover(parent1,
parent2);
                } else {
                    throw new IllegalStateException("No
crossover option selected");
                }

                mutate(childPath);
                newPopulation.add(new Individual(childPath));
            }

            population = newPopulation;
        }
    }

    public List<Point2D> greedyAlgorithm(List<Point2D>
initialPath) {
        List<Point2D> remainingPoints = new
ArrayList<>(initialPath);
        List<Point2D> greedyPath = new ArrayList<>();

        Point2D currentPoint = remainingPoints.remove((int)
(Math.random() * remainingPoints.size()));
        greedyPath.add(currentPoint);

        while (!remainingPoints.isEmpty()) {
            double minDistance = Double.MAX_VALUE;
            Point2D nextPoint = null;

            for (Point2D point : remainingPoints) {
                double distance =
TSPUtils.calculateDistance(currentPoint, point);
```

```
                if (distance < minDistance) {
                    minDistance = distance;
                    nextPoint = point;
                }
            }

            remainingPoints.remove(nextPoint);
            greedyPath.add(nextPoint);
            currentPoint = nextPoint;
        }
        return greedyPath;
    }


    private List<Individual> getElites() {
        List<Individual> elites = new ArrayList<>(elitismCount);
        population.sort(Individual::compareTo);
        elites.addAll(population.subList(0, elitismCount));
        return elites;
    }

    private Individual selectParent() {
        return population.get((int) (Math.random() *
populationSize));
    }

    private List<Point2D> crossover(Individual parent1,
Individual parent2) {
        int start = (int) (Math.random() * points.size());
        int end = (int) (Math.random() * points.size());
        if (start > end) {
            int temp = start;
            start = end;
            end = temp;
        }

        List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
        for (int i = start; i <= end; i++) {
            childPath.set(i, parent1.getPath().get(i));
        }

        int currentIndex = 0;
        for (Point2D point : parent2.getPath()) {
            if (!childPath.contains(point)) {
                while (childPath.get(currentIndex) != null) {
                    currentIndex = (currentIndex + 1) %
points.size();
                }
                childPath.set(currentIndex, point);
                currentIndex = (currentIndex + 1) %
points.size();
            }
```

```
        }

        return childPath;
    }

    private List<Point2D> twoPointCrossover(Individual parent1,
Individual parent2) {
        int point1 = (int) (Math.random() * points.size());
        int point2 = (int) (Math.random() * points.size());
        int start = Math.min(point1, point2);
        int end = Math.max(point1, point2);

        List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
        for (int i = start; i <= end; i++) {
            childPath.set(i, parent1.getPath().get(i));
        }

        int currentIndex = 0;
        for (Point2D point : parent2.getPath()) {
            if (!childPath.contains(point)) {
                while (childPath.get(currentIndex) != null) {
                    currentIndex = (currentIndex + 1) %
points.size();
                }
                childPath.set(currentIndex, point);
                currentIndex = (currentIndex + 1) %
points.size();
            }
        }

        return childPath;
    }

    private List<Point2D> cycleCrossover(Individual parent1,
Individual parent2) {
        List<Point2D> childPath = new
ArrayList<>(Collections.nCopies(points.size(), null));
        boolean[] visited = new boolean[points.size()];

        int cycleStart = (int) (Math.random() * points.size());
        int index = cycleStart;

        do {
            childPath.set(index, parent1.getPath().get(index));
            visited[index] = true;
            index =
parent2.getPath().indexOf(parent1.getPath().get(index));
        } while (index != cycleStart);

        for (int i = 0; i < points.size(); i++) {
            if (!visited[i]) {
                childPath.set(i, parent2.getPath().get(i));
```

```
            }
        }

        return childPath;
    }


    private void mutate(List<Point2D> path) {
        if (Math.random() < mutationRate) {
            int index1 = (int) (Math.random() * path.size());
            int index2 = (int) (Math.random() * path.size());
            Collections.swap(path, index1, index2);
        }
    }

    public List<Point2D> getBestSolution() {
        return
Collections.unmodifiableList(getElites().get(0).getPath());
    }

    private static class Individual implements
Comparable<Individual> {
        private List<Point2D> path;
        private double fitness = -1;

        public Individual(List<Point2D> path) {
            this.path = path;
        }

        public List<Point2D> getPath() {
            return path;
        }

        public double getFitness() {
            if (fitness == -1) {
                fitness = calculateFitness();
            }
            return fitness;
        }

        private double calculateFitness() {
            double totalDistance = 0;
            for (int i = 0; i < path.size() - 1; i++) {
                totalDistance += calculateDistance(path.get(i),
path.get(i + 1));
            }

            return 1 / totalDistance;        }

        private double calculateDistance(Point2D p1, Point2D p2)
{
            double dx = p1.getX() - p2.getX();
            double dy = p1.getY() - p2.getY();
```

```
            return Math.sqrt(dx * dx + dy * dy);
        }


        @Override
        public int compareTo(Individual o) {
            return Double.compare(o.getFitness(), getFitness());
        }
    }

    public static class TSPUtils {
        private static double calculateDistance(Point2D point1,
Point2D point2) {
            double dx = point1.getX() - point2.getX();
            double dy = point1.getY() - point2.getY();
            return Math.sqrt(dx * dx + dy * dy);
        }
        public static double
calculateRouteDistance(List<Point2D.Double> route) {
            double distance = 0.0;

            for (int i = 0; i < route.size() - 1; i++) {
                Point2D.Double currentPoint = route.get(i);
                Point2D.Double nextPoint = route.get(i + 1);
                distance += calculateDistance(currentPoint,
nextPoint);
            }

            Point2D.Double firstPoint = route.get(0);
            Point2D.Double lastPoint = route.get(route.size() -
1);
            distance += calculateDistance(lastPoint,
firstPoint);

            return distance;
        }
    }
}


import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.stream.Collectors;
import javafx.fxml.FXML;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.canvas.Canvas;
```

```
import javafx.scene.paint.Color;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.control.CheckBox;



public class Controller {

    private List<Point2D.Double> points;
    private List<Integer> pointNumbers = new ArrayList<>();
    private TSPGeneticAlgorithmModel tspGeneticAlgorithm;
    public static boolean useGreedyAlgorithm = false;

    @FXML
    private Button generateButton;

    @FXML
    private TextField numPointsField;

    @FXML
    private TextField popSize;

    @FXML
    private TextField numOfIteration;

    @FXML
    private TextField mutRate;

    @FXML
    private Button solveButton;

    @FXML
    public RadioButton onePointRadioButton;

    @FXML
    public RadioButton twoPointRadioButton;

    @FXML
    public RadioButton cycleCrossoverRadioButton;

    @FXML
    private Canvas canvas;

    @FXML
    private CheckBox greedyAlgorithm;
    private Point event;

    @FXML
    void initialize() {
        points = new ArrayList<>();
        canvas.setWidth(934.0);
        canvas.setHeight(750.0);
```

```
        double maxWidth = 934.0;
        double maxHeight = 750.0;


        canvas.widthProperty().addListener((observable,
oldValue, newValue) -> {
            if (newValue.doubleValue() > maxWidth) {
                canvas.setWidth(maxWidth);
            }
        });

        canvas.heightProperty().addListener((observable,
oldValue, newValue) -> {
            if (newValue.doubleValue() > maxHeight) {
                canvas.setHeight(maxHeight);
            }
        });


canvas.setOnMouseClicked(this::handleCanvasMouseClicked);

        ToggleGroup toggleGroup = new ToggleGroup();
        onePointRadioButton.setToggleGroup(toggleGroup);
        twoPointRadioButton.setToggleGroup(toggleGroup);
        cycleCrossoverRadioButton.setToggleGroup(toggleGroup);

        generateButton.setOnAction(event -> {
            int numPoints =
Integer.parseInt(numPointsField.getText());
            generateRandomPoints(numPoints);
            drawPoints(canvas.getGraphicsContext2D());
        });

        solveButton.setOnAction(event -> solveTSP());

greedyAlgorithm.selectedProperty().addListener((observable,
oldValue, newValue) -> {

            useGreedyAlgorithm = newValue;
        });
    }

    private void
handleCanvasMouseClicked(javafx.scene.input.MouseEvent
mouseEvent) {
        double x = mouseEvent.getX();
        double y = mouseEvent.getY();

        Point2D.Double point = new Point2D.Double(x, y);
        points.add(point);
        pointNumbers.add(points.size());

        drawPoints(canvas.getGraphicsContext2D());
```

```
    }

    private void generateRandomPoints(int numPoints) {
        points.clear();
        pointNumbers.clear();
        double canvasWidth = canvas.getWidth() - 10;
double canvasHeight = canvas.getHeight() - 10;
        for (int i = 0; i < numPoints; i++) {
            double x = Math.random() * canvasWidth + 5;
            double y = Math.random() * canvasHeight + 5;

            points.add(new Point2D.Double(x, y));
            pointNumbers.add(points.size());        }


    }


    private void drawPoints(GraphicsContext gc) {
        gc.clearRect(0, 0, gc.getCanvas().getWidth(),
gc.getCanvas().getHeight());
        int index = 0;

        for (Point2D.Double point : points) {
            double x = point.getX();
            double y = point.getY();
            double radius = 1;
gc.setFill(Color.BLACK);
            gc.fillOval(x - radius, y - radius, 2 * radius, 2 *
radius);

            gc.setFill(Color.RED);

gc.fillText(Integer.toString(pointNumbers.get(index)), x + 5, y
- 5);

            index++;
        }
    }

    private void drawPath(List<Point2D.Double> path) {
        GraphicsContext gc = canvas.getGraphicsContext2D();

        if (path == null || path.size() < 2) {
            System.out.println("Invalid path. Unable to draw.");
            return;
        }

        drawPoints(gc);

        for (int i = 0; i < path.size() - 1; i++) {
            Point2D.Double p1 = path.get(i);
            Point2D.Double p2 = path.get(i + 1);
```

```
            gc.strokeLine(p1.getX(), p1.getY(), p2.getX(),
p2.getY());
        }
        Point2D.Double firstPoint = path.get(0);
        Point2D.Double lastPoint = path.get(path.size() - 1);
        gc.strokeLine(lastPoint.getX(), lastPoint.getY(),
firstPoint.getX(), firstPoint.getY());
    }

    private void solveTSP() {
        if (points.size() < 2) {
            System.out.println("Add at least two points before
solving TSP.");
            return;
        }

        int populationSize =
Integer.parseInt(popSize.getText());
        int numberOfIteration=
Integer.parseInt(numOfIteration.getText());
        double
mutationRate=Double.parseDouble(mutRate.getText());

        tspGeneticAlgorithm = new TSPGeneticAlgorithmModel(new
ArrayList<>(points), populationSize, numberOfIteration,
mutationRate, onePointRadioButton, twoPointRadioButton,
cycleCrossoverRadioButton, useGreedyAlgorithm);
        long startTime = System.currentTimeMillis();
        tspGeneticAlgorithm.evolve();
        long endTime = System.currentTimeMillis();

        List<Point2D.Double> bestSolution =
convertToPoint2D(tspGeneticAlgorithm.getBestSolution());

        System.out.println("Best Solution: " + bestSolution);
        for (Point2D.Double point : bestSolution) {
            int index = points.indexOf(point);
            System.out.print(index + " ");
        }
        System.out.println("\nTime taken: " + (endTime -
startTime) + " milliseconds");

        double distance =
TSPGeneticAlgorithmModel.TSPUtils.calculateRouteDistance(bestSol
ution);
        System.out.println("Route Distance: " + distance);

        drawPath(bestSolution);
    }

    private List<Point2D.Double> convertToPoint2D(List<Point2D>
points) {
        return points.stream()
```

```
                .map(point -> new Point2D.Double(point.getX(),
point.getY()))
                .collect(Collectors.toList());
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.canvas.Canvas?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.CheckBox?>
<?import javafx.scene.control.RadioButton?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.effect.DropShadow?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.Pane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane prefHeight="750.0" prefWidth="1100.0"
xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="Controller">
    <children>
        <Canvas fx:id="canvas" height="750.0" width="934.0"
AnchorPane.leftAnchor="0.0" AnchorPane.topAnchor="0.0" />
        <Pane maxHeight="750.0" maxWidth="166.0"
prefHeight="400.0" prefWidth="166.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
            <effect>
                <DropShadow />
            </effect>
            <children>
                <AnchorPane prefHeight="450.0" prefWidth="166.0"
style="-fx-background-color: #696969;">
                    <effect>
                        <DropShadow />
                    </effect>
                    <children>
                        <Text fill="WHITE" layoutX="60.0"
layoutY="27.0" strokeType="OUTSIDE" strokeWidth="0.0"
text="TSP">
                            <font>
                                <Font name="Algerian"
size="30.0" />
                            </font>
                            <effect>
                                <DropShadow />
                            </effect>
                        </Text>
                        <Text layoutX="9.0" layoutY="56.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Enter number of
points">
```

```
                            <font>
                                <Font name="Arial" size="12.0"
/>
                            </font>
                        </Text>
                        <TextField fx:id="numPointsField"
layoutX="8.0" layoutY="66.0" />
                        <Button fx:id="generateButton"
layoutX="35.0" layoutY="373.0" mnemonicParsing="false"
text="Generate Points" />
                        <Button fx:id="solveButton"
layoutX="35.0" layoutY="409.0" mnemonicParsing="false"
prefHeight="26.0" prefWidth="101.0" style="-fx-background-color:
#90EE90;" text="Solve TSP" textOverrun="CLIP">
                            <font>
                                <Font name="Arial" size="12.0"
/>
                            </font>
                        </Button>
                        <RadioButton fx:id="onePointRadioButton"
layoutX="9.0" layoutY="161.0" mnemonicParsing="false"
selected="true" text="One-point">
                            <font>
                                <Font name="Arial" size="12.0"
/>
                            </font>
                        </RadioButton>
                        <RadioButton fx:id="twoPointRadioButton"
layoutX="9.0" layoutY="185.0" mnemonicParsing="false" text="Two-
point">
                            <font>
                                <Font name="Arial" size="12.0"
/>
                            </font>
                        </RadioButton>
                        <RadioButton
fx:id="cycleCrossoverRadioButton" layoutX="9.0" layoutY="209.0"
mnemonicParsing="false" text="Cycle crossover">
                            <font>
                                <Font name="Arial" size="12.0"
/>
                            </font>
                        </RadioButton>
                        <CheckBox fx:id="greedyAlgorithm"
layoutX="10.0" layoutY="345.0" mnemonicParsing="false"
text="with Greedy algorithm" />
                    <TextField fx:id="popSize" layoutX="8.0"
layoutY="118.0" />
                    <Text layoutX="9.0" layoutY="112.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Enter population
size">
                        <font>
                            <Font name="Arial" size="12.0" />
```

```
                    </font>
                  </Text>
                  <TextField fx:id="numOfIteration"
layoutX="9.0" layoutY="251.0" />
                  <Text layoutX="9.0" layoutY="245.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Enter number of
iteration">
                    <font>
                       <Font name="Arial" size="12.0" />
                    </font>
                  </Text>
                  <TextField fx:id="mutRate" layoutX="9.0"
layoutY="296.0" />

                   </children>
                </AnchorPane>
            <Text layoutX="9.0" layoutY="291.0"
strokeType="OUTSIDE" strokeWidth="0.0" text="Enter mutation
rate">
               <font>
                  <Font name="Arial" size="12.0" />
               </font>
            </Text>
            </children>
        </Pane>
    </children>
</AnchorPane>
```