

**Міністерство освіти і науки України**  
**Національний технічний університет України «Київський політехнічний**  
**інститут імені Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни  
«Проектування алгоритмів»

**„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1”**

**Виконав(ла)**

ІП-13 Дойчев Костянтин

(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Головченко М.Н.

(прізвище, ім'я, по батькові)

Київ 2022

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ</b>	<b>10</b>
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	10
3.1.1	<i>Вихідний код</i>	<i>10</i>
3.1.2	<i>Приклади роботи</i>	<i>10</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ	11
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій</i>	<i>11</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій</i>	<i>11</i>
	<b>ВИСНОВОК</b>	<b>12</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ</b>	<b>13</b>

## 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

## 2 ЗАВДАННЯ

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

№	Задача і алгоритм
1	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
2	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 30$ , починають маршрут в різних випадкових вершинах).
3	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
4	Задача про рюкзак (місткість $P=200$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з

	ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
5	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 35$ , починають маршрут в різних випадкових вершинах).
6	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
7	Задача про рюкзак (місткість $P=150$ , 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,3$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ , починають маршрут в різних випадкових вершинах).
9	Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники).
10	Задача про рюкзак (місткість $P=150$ , 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.

11	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,6$ , $L_{min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ , починають маршрут в різних випадкових вершинах).
12	Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники).
13	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
14	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ( $\alpha = 4$ , $\beta = 2$ , $\rho = 0,3$ , $L_{min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
15	Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники).
16	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.

17	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,7$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
18	Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники).
19	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
20	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,7$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
21	Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники).
22	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.

23	Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм ( $\alpha = 3$ , $\beta = 2$ , $\rho = 0,6$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 45$ (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
24	Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники).
25	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
26	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 30$ , починають маршрут в різних випадкових вершинах).
27	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
28	Задача про рюкзак (місткість $P=200$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.



29	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 35$ , починають маршрут в різних випадкових вершинах).
30	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
31	Задача про рюкзак (місткість $P=250$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
32	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 4$ , $\rho = 0,4$ , $L_{\min}$ знайти жадібним алгоритмом, кількість мурах $M = 30$ , починають маршрут в різних випадкових вершинах).
33	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
34	Задача про рюкзак (місткість $P=200$ , 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
35	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ( $\alpha = 2$ , $\beta = 3$ , $\rho = 0,4$ , $L_{\min}$ знайти

	жадібним алгоритмом, кількість мурах $M = 35$ , починають маршрут в різних випадкових вершинах).
--	--

### 3.1 Програмна реалізація алгоритму

#### 3.1.1 Вихідний код у [GitHub репозиторії](#)

```
// main.ts
import {Problem} from "./problem";
import {AntAlgorithm} from "./ant-algorithm";
import * as fs from "node:fs";
import {NUMBER_OF_ITERATIONS, RESULTS_CHECKPOINT} from "./constants";

const main = () => {
  const problem = new Problem();
  const algorithm = new AntAlgorithm(problem);

  let csv = "iteration, solution\n";
  const iterations = NUMBER_OF_ITERATIONS / RESULTS_CHECKPOINT;
  for (let i = 0; i < iterations; i++) {
    for (let j = 0; j < RESULTS_CHECKPOINT; j++) {
      algorithm.iterate();
    }

    const line = RESULTS_CHECKPOINT * (i + 1) + "," +
problem.getCost(algorithm.getSolution()) + "\n";
    console.log(line);
    csv += line;
  }

  fs.writeFileSync("results.csv", csv);
}

main();

// problem.ts
import * as fs from "node:fs"
import {MAX_DISTANCE, MIN_DISTANCE, NUMBER_OF_CITIES} from "./constants";
import {getRandomInt} from "./utils";

export class Problem {
  private static readonly fileName = "problem.json";
  public matrix: number[][] = [];
  public optimalSolution: number = 0;

  constructor() {
```

```

        this.initializeMatrix();
        this.optimalSolution = this.findOptimalSolution();
    }

    public getCost(path: number[]): number {
        let solution = 0;
        for (let i = 0; i < NUMBER_OF_CITIES; i++) {
            solution += this.getDistance(path[i], path[i + 1]);
        }

        return solution;
    }

    public getDistance(source: number, destination: number): number {
        return this.matrix[source][destination];
    }

    private initializeMatrix(): void {
        if (fs.existsSync(Problem.fileName)) {
            const content = fs.readFileSync(Problem.fileName);
            this.matrix = JSON.parse(content.toString());
            return;
        }

        this.generateMatrix();
        fs.writeFileSync(Problem.fileName, JSON.stringify(this.matrix));
    }

    private generateMatrix(): void {
        this.matrix = new Array<number[]>(NUMBER_OF_CITIES);
        for (let i = 0; i < NUMBER_OF_CITIES; i++) {
            this.matrix[i] = new Array<number>(NUMBER_OF_CITIES);
            for (let j = 0; j < NUMBER_OF_CITIES; j++) {
                /**
                 * The cost of going between cities
                 *
                 * |   | 1 | 2 | 3 |
                 * | 1 | ∞ | x | x |
                 * | 2 | x | ∞ | x |
                 * | 3 | x | x | ∞ |
                 *
                 * x = random number between MIN_DISTANCE and MAX_DISTANCE
                 */
                this.matrix[i][j] = i == j ? Number.MAX_VALUE :
getRandomInt(MIN_DISTANCE, MAX_DISTANCE);
            }
        }
    }

```

```

    }
}

private findOptimalSolution(): number {
    const solutions: number[] = [];
    const nodes = Array.from({length: NUMBER_OF_CITIES}, (_, i) => i);
    for (let j = 0; j < NUMBER_OF_CITIES; j++) {
        let currentNode = j;
        const path: number[] = [];
        path.push(currentNode);
        for (let i = 0; i < NUMBER_OF_CITIES - 1; i++) {
            const node = currentNode;
            currentNode = nodes
                .filter(x => !path.includes(x))
                .reduce((prev, curr) => this.getDistance(node, prev) <
this.getDistance(node, curr) ? prev : curr);
            path.push(currentNode);
        }
        path.push(j);
        solutions.push(this.getCost(path));
    }

    return Math.min(...solutions);
}
}

// constants.ts
export const NUMBER_OF_CITIES = 200;
export const NUMBER_OF_ANTS = 45;
export const MIN_DISTANCE = 0;
export const MAX_DISTANCE = 50
export const INITIAL_PHEROMONE = 0.1;
export const CONSTANT_ARGUMENTS = {
    alpha: 3,
    beta: 2,
    p: 0.3,
};

export const RESULTS_CHECKPOINT = 20;

export const NUMBER_OF_ITERATIONS = 1000;

/**
 * according to the formula:
 * pheromone[i][j] = (1 - p) * pheromone[i][j] + deltaPheromone[i][j]
 */
export const PHEROMONE_DISAPPEARANCE_COEFFICIENT = 1 - CONSTANT_ARGUMENTS.p;

```

```

// ant-algorithm.ts
import {Problem} from "../problem";
import {getAntSight, getRandomInt} from "../utils";
import {
    CONSTANT_ARGUMENTS,
    INITIAL_PHEROMONE,
    NUMBER_OF_ANTS,
    NUMBER_OF_CITIES,
    PHEROMONE_DISAPPEARANCE_COEFFICIENT
} from "../constants";

export class AntAlgorithm {
    private problem: Problem;
    public pheromoneMatrix: number[][];

    constructor(problem: Problem) {
        this.problem = problem;
        this.pheromoneMatrix = new Array<number[]>(NUMBER_OF_CITIES);
        this.initializePheromoneMatrix();
    }

    public iterate() {
        // each ant finds a path
        const paths: number[][] = []
        for (let i = 0; i < NUMBER_OF_ANTS; i++) {
            const initialCity = getRandomInt(0, NUMBER_OF_CITIES - 1);
            const path = this.findPath(initialCity);
            paths.push(path);
        }

        this.updatePheromones(paths);
    }

    public getSolution() {
        const path: number[] = []
        path.push(0);
        for (let i = 1; i < NUMBER_OF_CITIES; i++) {
            const currentIndex = path[i - 1];
            const pheromoneArray = this.pheromoneMatrix[currentIndex];
            let max = Number.MIN_VALUE;
            let maxIndex = -1;
            for (let j = 1; j < NUMBER_OF_CITIES; j++) {
                if (path.includes(j)) {
                    continue;
                }
            }
        }
    }
}

```

```

        if (pheromoneArray[j] > max) {
            max = pheromoneArray[j];
            maxIndex = j;
        }
    }

    path.push(maxIndex);
}

path.push(0);
return path;
}

private getProbabilities(currentNode: number, allowedNodes: number[]) {

    const probabilities = new Array<number>(allowedNodes.length);
    let sum = 0.0;
    for (let i = 0; i < allowedNodes.length; i++) {
        const destinationNode = allowedNodes[i];
        const pheromone =
this.pheromoneMatrix[currentNode][destinationNode];
        const antSight =
getAntSight(this.problem.matrix[currentNode][destinationNode]);
        /* based on the formula:  $P_{ij} = (t_{ij})^{\alpha} * (n_{ij})^{\beta} /$ 
sum( $(t_{ij})^{\alpha} * (n_{ij})^{\beta}$ )
        * where:
        *  $p_{ij}$  - probability of choosing the edge (i, j)
        *  $t_{ij}$  - pheromone
        *  $n_{ij}$  - ant sight
        *  $\alpha, \beta$  - constant argument
        */
        probabilities[i] = Math.pow(pheromone, CONSTANT_ARGUMENTS.alpha)
*
        Math.pow(antSight, CONSTANT_ARGUMENTS.beta);
        sum += probabilities[i];
    }

    for (let i = 0; i < probabilities.length; i++) {
        probabilities[i] /= sum;
    }

    return probabilities;
}

private updatePheromones(paths: number[][]) {

```

```

        for (let i = 0; i < NUMBER_OF_CITIES; i++) {
            for (let j = 0; j < NUMBER_OF_CITIES; j++) {
                this.pheromoneMatrix[i][j] *=
    PHEROMONE_DISAPPEARANCE_COEFFICIENT;
            }
        }

        paths.forEach(path => {
            const cost = this.problem.getCost(path);
            for (let i = 0; i < NUMBER_OF_CITIES; i++) {
                /**
                 * based on the formula: Lmin/Lk
                 * where:
                 * Lmin - the length of the shortest path
                 * Lk - the length of the path for k-th ant
                 */
                this.pheromoneMatrix[path[i]][path[i + 1]] +=
    this.problem.optimalSolution / cost;
            }
        })
    }

    private findPath(initialCity: number) {
        /**
         * The sequence of cities that the ant has visited.
         */
        const result: number[] = new Array<number>(NUMBER_OF_CITIES + 1);
        const citiesToVisit =
    Array.from(Array(NUMBER_OF_CITIES).keys()).filter(n => n !== initialCity);
        result[0] = initialCity;

        for (let i = 1; i < NUMBER_OF_CITIES; i++) {
            const probabilities = this.getProbabilities(result[i - 1],
    citiesToVisit);
            const nodeIndex = this.chooseNode(probabilities);
            result[i] = citiesToVisit[nodeIndex];
            citiesToVisit.splice(nodeIndex, 1);
        }
        const lastIndex = result.length - 1;
        result[lastIndex] = initialCity;
        return result;
    }

    /**
     * Chooses a node based on the probabilities.

```



```

    * The higher the probability, the higher the chance of being chosen.
    *
    * @returns index of the chosen node
    */
    private chooseNode(probabilities: number[]) {
        if (probabilities.length === 0) {
            throw new Error('No probabilities');
        }
        const random = Math.random();
        let sum = 0;
        for (let i = 0; i < probabilities.length; i++) {
            sum += probabilities[i];
            if (sum > random) {
                return i;
            }
        }

        return probabilities.length - 1;
    }

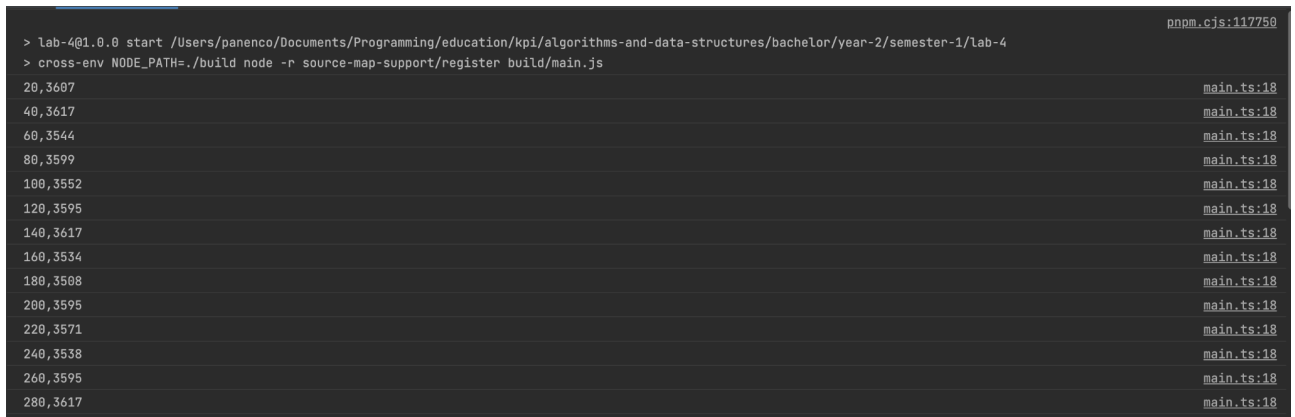
    private initializePheromoneMatrix() {
        this.pheromoneMatrix = new Array<number[]>(NUMBER_OF_CITIES);
        for (let i = 0; i < NUMBER_OF_CITIES; i++) {
            this.pheromoneMatrix[i] = new Array<number>(NUMBER_OF_CITIES);
            for (let j = 0; j < NUMBER_OF_CITIES; j++) {
                this.pheromoneMatrix[i][j] = i === j ? 0 : INITIAL_PHEROMONE
            }
        }
    }
}

```

### 3.1.2 Приклади роботи

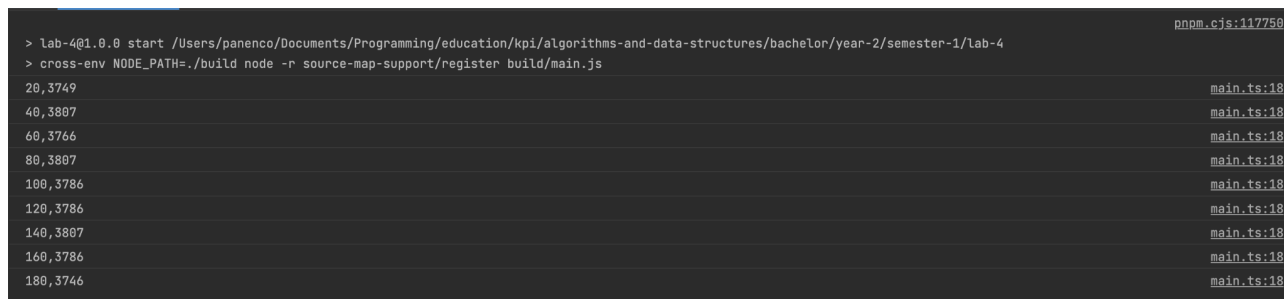
На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –



> lab-4@1.0.0 start /Users/panenco/Documents/Programming/education/kpi/algorithms-and-data-structures/bachelor/year-2/semester-1/lab-4	pnpm.cjs:117750
> cross-env NODE_PATH=./build node -r source-map-support/register build/main.js	
20,3607	main.ts:18
40,3617	main.ts:18
60,3544	main.ts:18
80,3599	main.ts:18
100,3552	main.ts:18
120,3595	main.ts:18
140,3617	main.ts:18
160,3534	main.ts:18
180,3508	main.ts:18
200,3595	main.ts:18
220,3571	main.ts:18
240,3538	main.ts:18
260,3595	main.ts:18
280,3617	main.ts:18

Рисунок 3.2 –



> lab-4@1.0.0 start /Users/panenco/Documents/Programming/education/kpi/algorithms-and-data-structures/bachelor/year-2/semester-1/lab-4	pnpm.cjs:117750
> cross-env NODE_PATH=./build node -r source-map-support/register build/main.js	
20,3749	main.ts:18
40,3807	main.ts:18
60,3766	main.ts:18
80,3807	main.ts:18
100,3786	main.ts:18
120,3786	main.ts:18
140,3807	main.ts:18
160,3786	main.ts:18
180,3746	main.ts:18

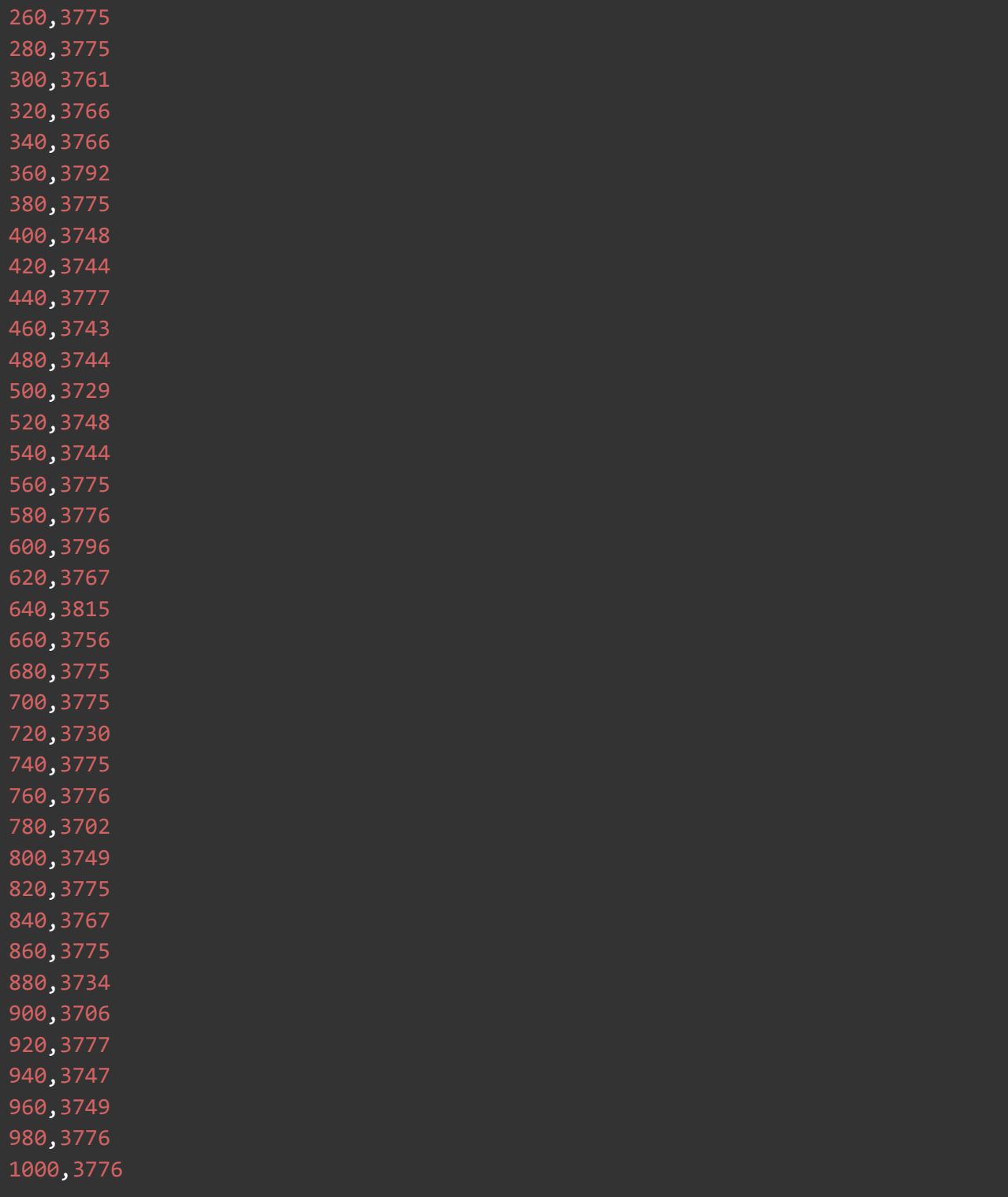
## Тестування алгоритму

### 3.1.3 Значення цільової функції зі збільшенням кількості ітерацій

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

iteration, solution
---------------------

20,3749
40,3807
60,3766
80,3807
100,3786
120,3786
140,3807
160,3786
180,3746
200,3786
220,3767
240,3767

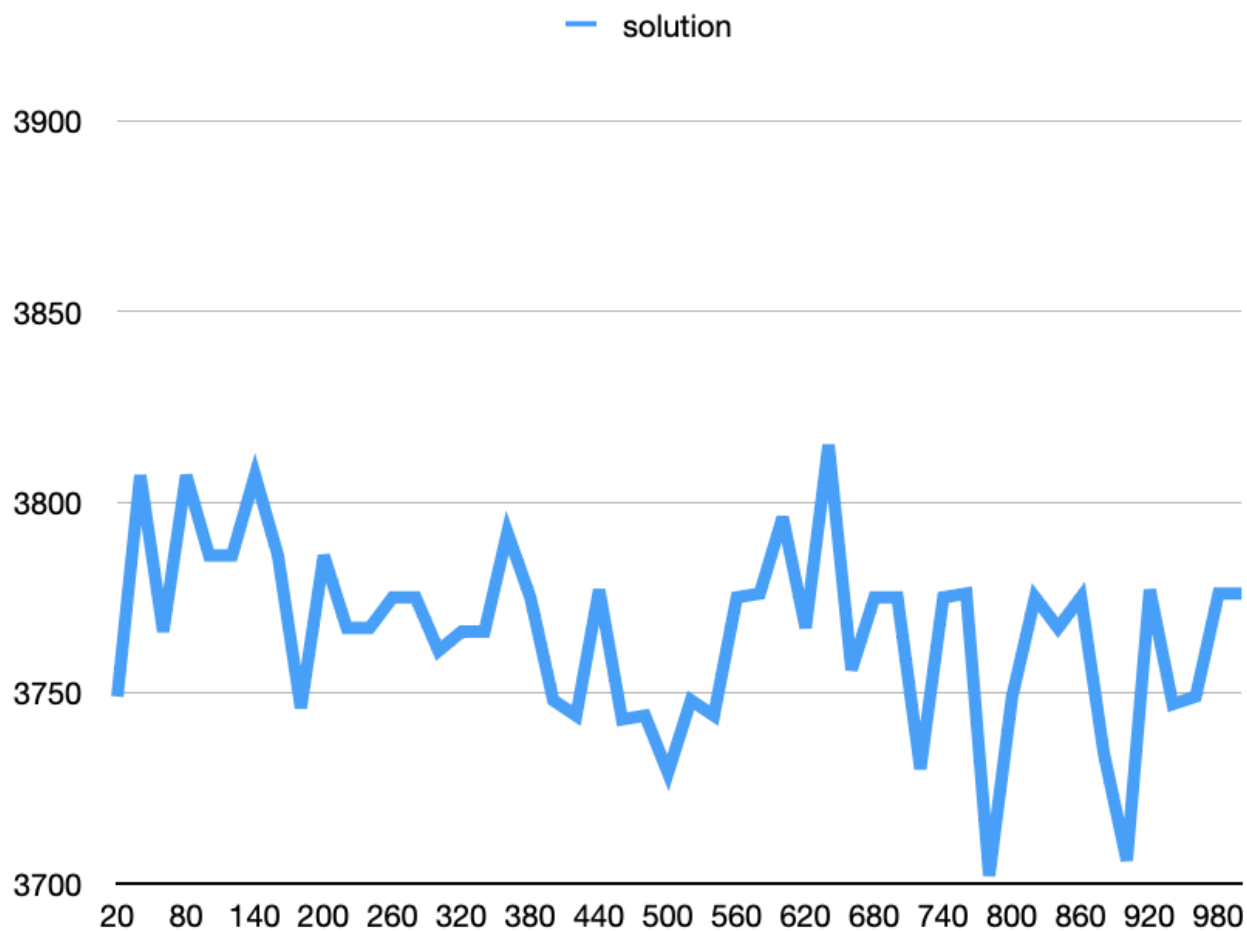


260	3775
280	3775
300	3761
320	3766
340	3766
360	3792
380	3775
400	3748
420	3744
440	3777
460	3743
480	3744
500	3729
520	3748
540	3744
560	3775
580	3776
600	3796
620	3767
640	3815
660	3756
680	3775
700	3775
720	3730
740	3775
760	3776
780	3702
800	3749
820	3775
840	3767
860	3775
880	3734
900	3706
920	3777
940	3747
960	3749
980	3776
1000	3776

#### 3.1.4 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 навчився використовувати ant colony optimization для задачі з комівояжером (traveling salesman problem)

### 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%.