

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)

ІІІ-13 Дойчев Костянтин
(шифр, прізвище, ім'я, по батькові)

Перевірів

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

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....	3
2	ЗАВДАННЯ.....	4
3	ВИКОНАННЯ	11
3.1	ПОКРОКОВИЙ АЛГОРИТМ	10
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	10
3.2.1	<i>Вихідний код.....</i>	<i>11</i>
3.2.2	<i>Приклади роботи.....</i>	<i>17</i>
3.3	ТЕСТУВАННЯ АЛГОРИТМУ	18
	ВИСНОВОК.....	19
	КРИТЕРІЇ ОЦІНЮВАННЯ.....	20

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метасвистичних 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 ступенем 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.

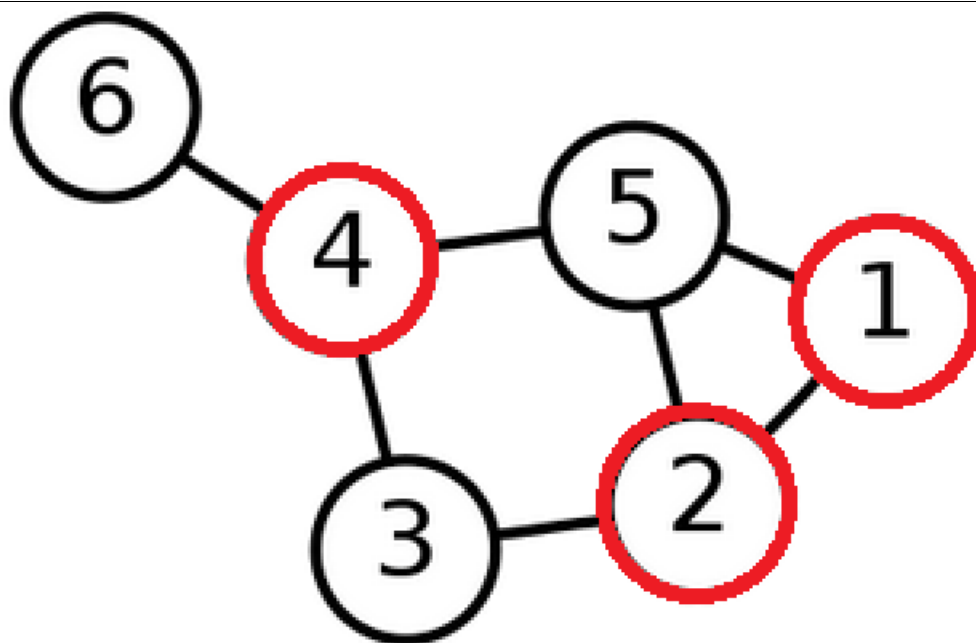
Зробити узагальнений висновок в якому обов'язково описати залежність якості розв'язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	Задача
1	Задача про рюкзак (місткість $P=500$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не

	<p>перевищувала задану, а сумарна цінність була максимальною.</p> <p>Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.</p>
2	<p>Задача комівояжера (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів.</p> <p>Розглядається симетричний, асиметричний та змішаний варіанти.</p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p> <p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води; – моніторинг об'єктів;

	<ul style="list-style-type: none"> – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.
3	<p>Розфарбовування графа (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.</p> <p>Застосування:</p> <ul style="list-style-type: none"> – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – розкладу для комунальних служб;
4	<p>Задача вершинного покриття (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа $G = (V, E)$ - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.</p> <p>Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).</p> <p>На вході: Граф $G = (V, E)$.</p> <p>Результат: множина $C \subseteq V$ - найменше вершинне покриття графа G.</p>



Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;
- проектування інтегральних схем і конвеєрних ліній.

5

Задача про кліку (300 вершин, степінь вершини не більше 30, але не менше 2). Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.

Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k , тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).

Застосування:

- біоінформатика;
- електротехніка;

6

Задача про найкоротший шлях (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) -

	<p>задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.</p> <p>Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.</p>
--	--

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
1	<p>Генетичний алгоритм:</p> <ul style="list-style-type: none"> - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).
2	<p>Мурашиний алгоритм:</p> <ul style="list-style-type: none"> - α; - β; - ρ; - L_{min}; - кількість мурах M і їх типи (елітні, тощо...); - маршрути з однієї чи різних вершин.
3	<p>Бджолиний алгоритм:</p> <ul style="list-style-type: none"> - кількість ділянок; - кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм

28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3 ВИКОНАННЯ

3.1 Вихідний код

[GitHub repo](#)

```
// constants.ts

export const NUMBER_OF_CITIES = 300;
export const NUMBER_OF_ANTS = 45;
export const MIN_DISTANCE = 5;
export const MAX_DISTANCE = 150;
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:  $L_{min}/L_k$ 
             * where:
             *  $L_{min}$  - the length of the shortest path
             *  $L_k$  - 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
            }
        }
    }
}

// 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 = 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);
}
}

// utils.ts

export const getRandomInt = (min: number, max: number) => {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

export const getAntSight = (distance: number) => {
    return 1 / distance;
}

// 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;
    }

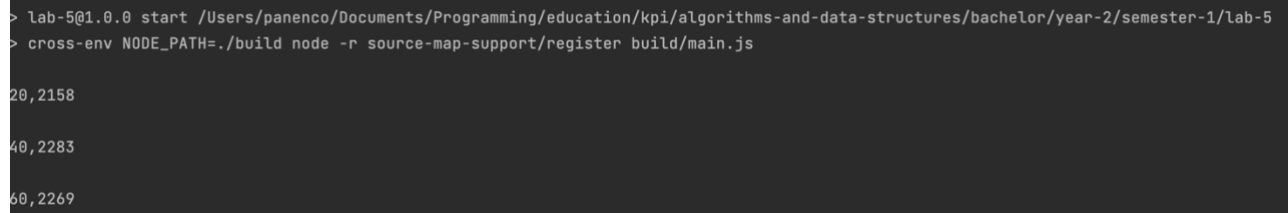
    fs.writeFileSync("results.csv", csv);
}

main();

```


3.1.1 Приклади роботи

На рисунку 3.1 показані приклади роботи програми.



```
> lab-5@1.0.0 start /Users/panenco/Documents/Programming/education/kpi/algorithms-and-data-structures/bachelor/year-2/semester-1/lab-5
> cross-env NODE_PATH=./build node -r source-map-support/register build/main.js

20,2158

40,2283

60,2269
```

Рисунок 3.1 – Термінал

Тестування алгоритму

```
iteration, solution
```

```
20,2158  
40,2283  
60,2269  
80,2269  
100,2269  
120,2269  
140,2269  
160,2269  
180,2269  
200,2269  
220,2269  
240,2269  
260,2269  
280,2269  
300,2269  
320,2269  
340,2269  
360,2269  
380,2269  
400,2269  
420,2269  
440,2269  
460,2269  
480,2269  
500,2269  
520,2269  
540,2269  
560,2269  
580,2269  
600,2269  
620,2269  
640,2269  
660,2269  
680,2269  
700,2269  
720,2269  
740,2269  
760,2269  
780,2269  
800,2269  
820,2269  
840,2269  
860,2269  
880,2269  
900,2269  
920,2269  
940,2269  
960,2269  
980,2269  
1000,2269
```

ВИСНОВОК

В рамках даної лабораторної роботи навчився оптимізувати алгоритм завдяки ant colony optimization

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму – 30%;
- висновок – 5%.