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-12 Волков Вадим Всеволодович 7.12.2022
(шифр, прізвище, ім’я, по батькові)

Перевірив Сопов Олексій Олександрович 8.12.2022
(прізвище, ім’я, по батькові)

Київ 2022

ЗМІСТ

1. МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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.

Таблиця 2.1 – Прикладні задачі

№	**Задача**
2	**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. **Розглядається симетричний, асиметричний та змішаний варіанти.** В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також

	зважати і на те, в якому напрямку знаходяться ребра. У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу. У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху. Застосування: - доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); - доставка води; - моніторинг об'єктів; - поповнення банкоматів готівкою; - збір співробітників для доставки вахтовим методом.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
2	**Мурашиний алгоритм**: - α; - β; - ρ; - Lmin; - кількість мурах M і їх типи (елітні, тощо…); - маршрути з однієї чи різних вершин.

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3. ВИКОНАННЯ
    1. Покроковий алгоритм
        1. **Ініціювання даних для роботи алгоритму (функція init):**
            1. Задати змінні на значення введені користувачем
            2. Задати кількість виконаних ітерація на 0
            3. Задати найкоротшу відстань на Infinity
            4. Заповнити масив вершин (для кожної вершини згенерувати 20 випадкових позицій і обрати ту, що найдальше від всіх вже існуючих)
            5. Заповнити матрицю відстаней відстанями між вершинами (при цьому, з випадковим шансом вказаним користувачем, додати чи відняти випадкове значення з діапазону вказаного користувачем)
            6. Заповнити матрицю феромонів значенням вказаним користувачем
        2. **Один крок алгоритму (функція step):**
            1. Заповнити матрицю додаваних феромонів нулями
            2. Порахувати кількість елітних мурах
            3. Повторювати для кожної мурахи
                1. Задати пройдену мурахою відстань на 0
                2. Встановити початкову і поточну позицію на випадкове число
                3. Створити список з відмітками про відвідання вершин і позначити в ньому поточну позицію
                4. Створити список відвіданих вершин/шляху і додати в нього номер поточної вершини
                5. Повторювати кількість вершин разів
                    1. Задати нову вершину на 0
                    2. Якщо всі вершини відвідано задати нову вершину на початкову вершину

3. Інакше якщо мураха елітна, порахувати шанси всіх шляхів до невідвіданих вершин з поточної вершини та обрати нову вершину з найбільшим шансом
4. Інакше порахувати шанси всіх шляхів до невідвіданих вершин з поточної вершини та обрати нову вершину випадково згідно ймовірностям
5. Збільшити пройдену відстань мурахою на значення з матриці відстаней
6. Залишити слід феромонів в матриці додання феромонів
7. Помітити нову вершину як відвідану
8. Додати нову вершину в список відвіданих вершин/шляху
9. Задати поточну вершину на нову вершину

6. Якщо пройдена відстань менше найкоротшої, запам’ятати її і список шляху до неї

4. Пройтись по всім елементам матриці феромонів, зменшуючи їх та додаючи нові феромони з матриці додаваних феромонів
5. Збільшити кількість пройдених ітерацій на 1
6. Якщо ітерацій менше 50 то перезапустити step з затримкою (кінець виконання)
7. Якщо потрібно ще проводити тестування, то зменшити змінну потрібних тестувань на 1, додати кількість покращення шляху цієї колонії, скинути змінні на початкові значення, викликати init, викликати step (кінець виконання)
8. Повністю кінець. Вивести результат

2. Програмна реалізація алгоритму

1. Вихідний код

index.html

```
<!doctype html>
<html lang="en">
	<head>
		<title>L5V8</title>
		<meta charset="UTF-8">
		<script src="main.js" defer></script>
	</head>
	<body style="display:flex; margin:0">
		<canvas id="canvas" width="700" height="700" style="height: 100vh;"></canvas>
		<div style="display:flex; flex-direction: column; padding: 10px;">
			<table>
				<tr><td>Вершин</td><td><input type="number" id="vertexCount"
value="100"></td></tr>
				<tr><td>α</td><td><input type="number" id="alpha"
value="1"></td></tr>
				<tr><td>β</td><td><input type="number" id="beta"
value="2"></td></tr>
				<tr><td>P</td><td><input type="number" id="ro"
value="0.1"></td></tr>
				<tr><td>M</td><td><input type="number" id="antCount"
value="20"></td></tr>
				<tr><td>M<sub>елітн</sub></td><td><input type="number"
id="eliteAntRatio" value="0"></td></tr>
				<tr><td>L<sub>min</sub></td><td><input type="number" id="Lmin"
value="1"></td></tr>
				<tr><td>Ферамон</td><td><input type="number" id="feramone"
value="1"></td></tr>
				<tr><td>Довжина відхиленнь</td><td><input type="number" id="scatter"
value="1"></td></tr>
				<tr><td>Шанс відхиленнь</td><td><input type="number" id="chance"
value="0.5"></td></tr>
				<tr><td>Візуалізація</td><td><input type="checkbox" id="visualize"
checked></td></tr>
				<tr><td>Замірів</td><td><input type="number" id="tests"
value="25"></td></tr>
			</table>
			<button id="test">Заміряти</button>
			<span id="state"></span>
		</div>
	</body>
</html>
```

main.js

```
<!do
"use strict";
let id = id => document.getElementById(id);
let randomInt = upTo => Math.floor(Math.random() * upTo);

let canvas = id("canvas");
let ctx = canvas.getContext("2d");
let size = canvas.width = canvas.height = window.innerHeight;
window.addEventListener("resize", () => {
	if(size == window.innerHeight) return;
	size = canvas.width = canvas.height = window.innerHeight
	draw();
});

let points = [];
let distanceMatrix = [];
let feramoneMatrix = [];
let Lmin = 0;
let alpha = 3;
let beta = 2;
let ro = 0.3;
let vertexCount = 200;
let antCount = 45;
let eliteAntRatio = 0.5;
let scatter = 1;
let iterations = 0;
let output = "";
```

```
let moreRuns = 0;
let score = 0;
let startShortest = 0;
let shortest = 0;
let visualize = true;
let tests = 25;

let rand = 0.1;
function random() {
	return rand =
Math.abs((Math.SQRT2*Math.sin(rand*6712534.28346+(rand*rand/3)%1)+rand*rand*1000 + 0.123)%1);
}

function init() {
	alpha = +id("alpha").value;
	beta = +id("beta").value;
	ro = +id("ro").value;
	vertexCount = +id("vertexCount").value;
	antCount = +id("antCount").value;
	eliteAntRatio = +id("eliteAntRatio").value;
	Lmin = +id("Lmin").value;
	let feramone = +id("feramone").value;
	scatter = +id("scatter").value;
	let chance = +id("chance").value;
	visualize = id("visualize").checked;
	shortest = Infinity;
	iterations = 0;

	points = [];
	distanceMatrix = [];
	feramoneMatrix = [];
	output = "";

	for(let i=0; i<vertexCount; i++) {
		let maxDist = 0;
		let maxPos;
		for(let j=0; j<20; j++) {
			let pos = [random()*35+0.5, random()*35+0.5, 0, -1]
			let minDist = Infinity;
			for(let k=0; k<points.length; k++) {
				let dx = pos[0]-points[k][0];
				let dy = pos[1]-points[k][1];
				let curDist = dx*dx + dy*dy
				if(curDist < minDist) {
					minDist = curDist;
				}
			}
			if(minDist > maxDist) {
				maxDist = minDist;
				maxPos = pos;
			}
		}
		points.push(maxPos);
	}

	for(let i=0; i<vertexCount; i++) {
		let distanceMatrixRow = [];
		let feramoneMatrixRow = [];
		for(let j=0; j<vertexCount; j++) {
			let dx = points[i][0]-points[j][0];
			let dy = points[i][1]-points[j][1];
			let dist = Math.sqrt(dx*dx + dy*dy);
			if(j>i && random() < chance) {
				dist += random()*scatter*2-scatter;
			}
			distanceMatrixRow.push(Math.min(5, dist));
			feramoneMatrixRow.push(feramone);
		}
		distanceMatrix.push(distanceMatrixRow);
		feramoneMatrix.push(feramoneMatrixRow);
	}
}

function draw() {
	ctx.save();
	ctx.scale(size / 256, size / 256);
	ctx.clearRect(0, 0, 256, 256);
	let scaler = 256 / 36;
	ctx.textAlign = "center";
	ctx.textBaseline = "middle";
```

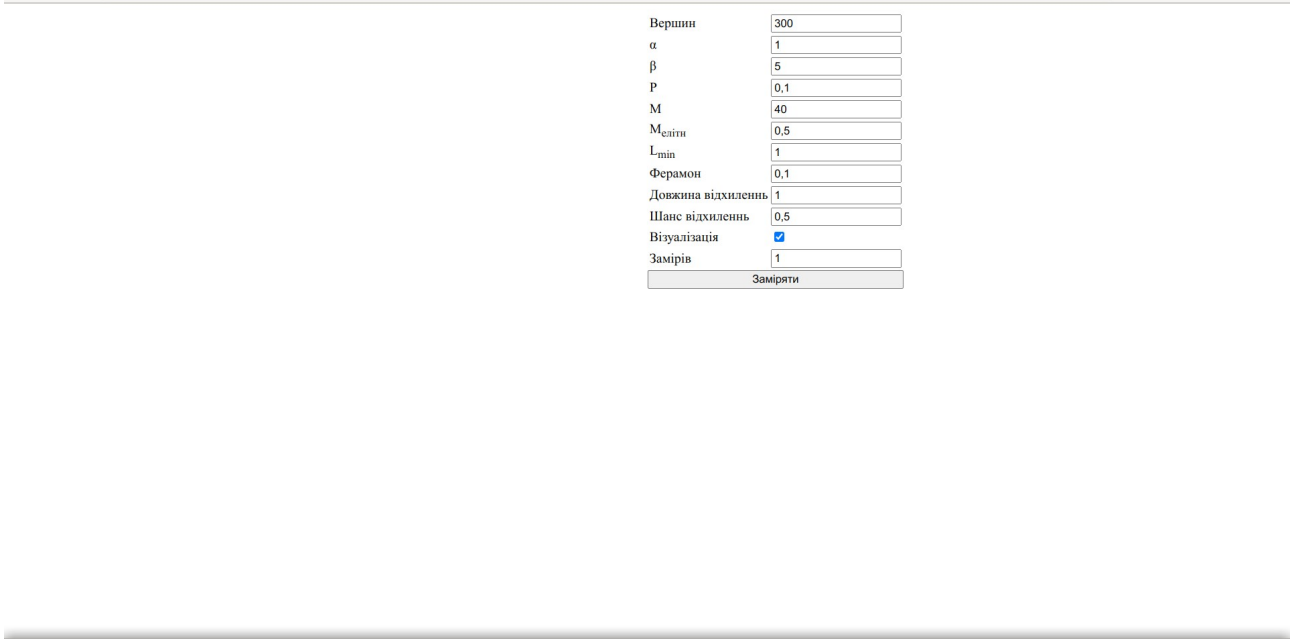
```
        ctx.font = "6px sans serif";
        ctx.strokeStyle = "#000000";
        let lastColor = false;

        for(let i=0; i<vertexCount; i++) {
                for(let j=i+1; j<vertexCount; j++) {
                        let thisColor = points[i][3] == j || points[j][3] == i;
                        if(feramoneMatrix[i][j] > 0.01 || thisColor) {
                                if(lastColor != thisColor) {
                                        ctx.strokeStyle = thisColor ? "#0000ff" : "#000000";
                                        lastColor = thisColor;
                                }

                                ctx.globalAlpha = thisColor ? 1 : Math.atan(feramoneMatrix[i][j] /
5) / Math.PI * 2;
                                ctx.beginPath();
                                ctx.moveTo(points[i][0] * scaler, points[i][1] * scaler);
                                ctx.lineTo(points[j][0] * scaler, points[j][1] * scaler);

                                if(distanceMatrix[i][j] !== distanceMatrix[j][i] &&
feramoneMatrix[i][j] > 0.2) {
                                        let p1, p2;
                                        if(distanceMatrix[i][j] < distanceMatrix[j][i]) {
                                                p1 = points[i];
                                                p2 = points[j];
                                        } else {
                                                p1 = points[j];
                                                p2 = points[i];
                                        }
                                        let dx = p2[0] - p1[0];
                                        let dy = p2[1] - p1[1];
                                        let length = Math.sqrt(dx*dx+dy*dy);
                                        dx /= length;
                                        dy /= length;

                                        ctx.moveTo(p2[0] * scaler - 3*dx     , p2[1] * scaler -
3*dy);
                                        ctx.lineTo(p2[0] * scaler - 4*dx - dy, p2[1] * scaler - 4*dy
+ dx);
                                        ctx.lineTo(p2[0] * scaler - 4*dx     , p2[1] * scaler -
4*dy);

                                        ctx.moveTo(p1[0] * scaler + 3*dx     , p1[1] * scaler +
3*dy);
                                        ctx.lineTo(p1[0] * scaler + 4*dx + dy, p1[1] * scaler + 4*dy
- dx);
                                        ctx.lineTo(p1[0] * scaler + 4*dx     , p1[1] * scaler +
4*dy);
                                }
                                ctx.closePath();
                                ctx.stroke();
                        }
                }
        }
        for(let i=0; i<vertexCount; i++) {
                for(let j=i+1; j<vertexCount; j++) {
                        let thisColor = points[i][3] == j || points[j][3] == i;
                        if(feramoneMatrix[i][j] || thisColor) {
                                ctx.globalAlpha = thisColor ? 1 : Math.atan(feramoneMatrix[i][j] /
5) / Math.PI * 2;
                                if(distanceMatrix[i][j] !== distanceMatrix[j][i]) {
                                        let p1, p2, v1, v2;
                                        if(distanceMatrix[i][j] < distanceMatrix[j][i]) {
                                                p1 = points[i];
                                                p2 = points[j];
                                                v1 = distanceMatrix[i][j];
                                                v2 = distanceMatrix[j][i];
                                        } else {
                                                p1 = points[j];
                                                p2 = points[i];
                                                v1 = distanceMatrix[j][i];
                                                v2 = distanceMatrix[i][j];
                                        }
                                        let dx = p2[0] - p1[0];
                                        let dy = p2[1] - p1[1];
                                        let length = Math.sqrt(dx*dx+dy*dy);
                                        dx /= length;
                                        dy /= length;
                                        ctx.fillStyle = "#00ff00";
```

```
                                        ctx.fillText(Math.round(v1*10)/10, (p1[0] + p2[0])/2 *
scaler - 3*dy, (p1[1] + p2[1])/2 * scaler + 3*dx);
                                        ctx.fillStyle = "#ff0000";
                                        ctx.fillText(Math.round(v2*10)/10, (p1[0] + p2[0])/2 *
scaler + 3*dy, (p1[1] + p2[1])/2 * scaler - 3*dx);
                                }
                        }
                }
        }
        ctx.fillStyle = "red";
        ctx.globalAlpha = 1;
        ctx.beginPath();
        for(let i=0; i<vertexCount; i++) {
                ctx.moveTo(points[i][0] * scaler + 3, points[i][1] * scaler);
                ctx.arc(points[i][0] * scaler, points[i][1] * scaler, 3, 0, Math.PI*2);
        }
        ctx.closePath();
        ctx.fill();

        ctx.restore();

        id("state").textContent = "Ітерація ("+iterations+"/50) Колонія ("+(tests - moreRuns)
+"/"+tests+") Покращення: "+score.toFixed(2);
}

function step() {
        let feramoneAddMatrix = [];
        for(let i=0; i<vertexCount; i++) {
                let row = [];
                for(let j=0; j<vertexCount; j++) {
                        row.push(0);
                }
                feramoneAddMatrix.push(row);
        }

        let eliteAntCount = antCount*eliteAntRatio;

        for(let i=0; i<antCount; i++) {
                let walked = 0;
                let startPos = randomInt(vertexCount);
                let curPos = startPos;
                let visited = new Array(vertexCount).fill(false);
                visited[curPos] = true;
                let path = [curPos];

                for(let k=1; k<=vertexCount; k++) {
                        let newPos = 0;
                        if(k < vertexCount) {
                                if(i <= eliteAntCount) {
                                        let maxChance = 0;
                                        for(let j=0; j<vertexCount; j++) {
                                                if(!visited[j]) {
                                                        let chance = (feramoneMatrix[curPos][j] **
alpha) * ((1 / distanceMatrix[curPos][j]) ** beta);
                                                        if(chance > maxChance) {
                                                                maxChance = chance;
                                                                newPos = j;
                                                        }
                                                }
                                        }
                                } else {
                                        let chances = [];
                                        let chanceSum = 0;
                                        for(let j=0; j<vertexCount; j++) {
                                                if(visited[j]) {
                                                        chances.push(0);
                                                } else {
                                                        let chance = (feramoneMatrix[curPos][j] **
alpha) * ((1 / distanceMatrix[curPos][j]) ** beta);
                                                        chanceSum += chance;
                                                        chances.push(chance);
                                                }
                                        }
                                        let chanceRanges = [];
                                        for(let j=0; j<vertexCount; j++) {
                                                chances[j] /= chanceSum;
                                                if(j == 0) {
                                                        chanceRanges.push(0);
                                                } else {
```

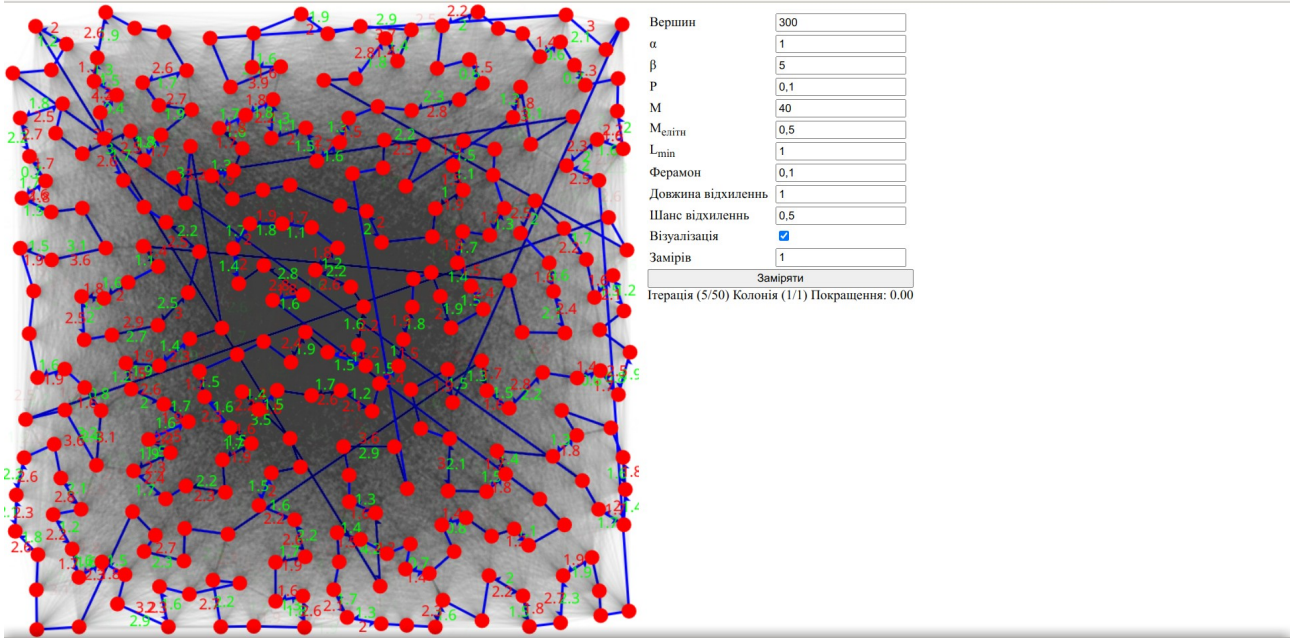
```
                                                        chanceRanges.push(chanceRanges[j-1] +
chances[j-1]);
                                                }
                                        }

                                        let random = Math.random();
                                        while(random > chanceRanges[newPos]) {
                                                newPos++;
                                        }
                                        newPos--;
                                }
                        } else {
                                newPos = startPos;
                        }

                        walked += distanceMatrix[curPos][newPos];
                        let add = Lmin / walked;
                        feramoneAddMatrix[curPos][newPos] += add;
                        feramoneAddMatrix[newPos][curPos] += add;
                        visited[newPos] = true;
                        path.push(newPos);
                        curPos = newPos;
                }
                if(walked < shortest) {
                        shortest = walked;
                        console.log(shortest);
                        for(let j=1; j<path.length; j++) {
                                points[path[j-1]][3] = path[j];
                        }
                }
        }
        for(let i=0; i<vertexCount; i++) {
                for(let j=0; j<vertexCount; j++) {
                        feramoneMatrix[i][j] = feramoneMatrix[i][j]*(1 - ro) + feramoneAddMatrix[i]
[j];
                }
        }

        if(iterations == 0) {
                startShortest = shortest;
        }

        iterations++;
        if(iterations < 50) {
                if(visualize) {
                        draw();
                }
                window.requestAnimationFrame(step);
        } else if(moreRuns > 0) {
                score += startShortest - shortest;
                console.log("result", startShortest - shortest);
                iterations = 0;
                moreRuns--;
                draw();
                init();
                window.setTimeout(step, 0);
        } else {
                score += startShortest - shortest;
                console.log("total score", score);
                alert("total score "+score);
        }
}

id("test").addEventListener("click", () => {
        rand = 0.15;
        score = 0;
        tests = +id("tests").value;
        moreRuns = tests - 1;
        init();
        step();
});
```

2. Виконання

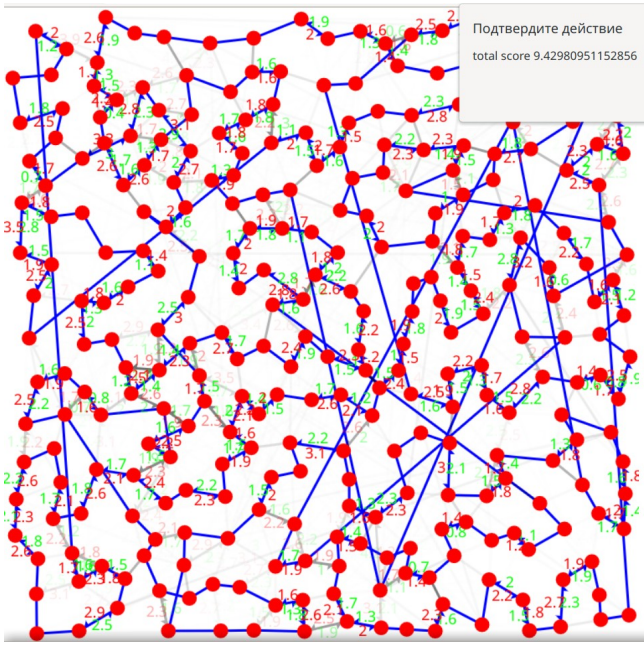
1:



2:



3:



Подтвердите действие

total score 9.42980951152856

OK

$M_{еліти}$	0,5
L_{min}	1
Ферамон	0,1
Довжина відхиленнь	1
Шанс відхиленнь	0,5
Візуалізація	☑
Замірів	1

Заміряти

Ітерація (49/50) Колонія (1/1) Покращення: 0.00

3. Тестування алгоритму

Вершин: 200

alpha	beta	P	M	Mel	Lmin	Fe	Start	Score
1	2	0.1	20	0	1	1	random	185,43
1	2	0.1	30	0	1	1	random	176,6
1	2	0.1	35	0	1	1	random	192,98
1	2	0.1	40	0	1	1	random	229,48
1	2	0.1	50	0	1	1	random	216,51
1	2	0,05	40	0	1	1	random	225,84
1	2	0,3	40	0	1	1	random	160,22
1	2	0,5	40	0	1	1	random	120,62
3	2	0,1	40	0	1	1	random	75,57
5	2	0,1	40	0	1	1	random	35,68
1	1	0,1	40	0	1	1	random	174,25
1	5	0,1	40	0	1	1	random	337,42
1	5	0,1	40	0,1	1	1	random	249,73
1	5	0,1	40	0,5	1	1	random	254,37
1	5	0,1	40	0	10	1	random	237,86
1	5	0,1	40	0	5	1	random	216,53
1	5	0,1	40	0	1	0,1	random	250,46
1	5	0,1	40	0	1	0,1	0	198,68
Результат підбору параметрів								
alpha	beta	P	M	Mel	Lmin	Fe	Start	
1	5	0,1	40	0	1	0,1	random	

4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з способом підбору аргументів для ефективної роботи метаевристичних алгоритмів. Реалізація алгоритму комівояжера на основі метаевристичного мурашиного алгоритму створеного в попередній лабораторній роботі була розширена і було додано підтримка змішаного типу цієї задачі. Потім на цій модифікованій версії алгоритму було проведено підбір різних аргументів для його оптимальної роботи. На основі результатів цього підбору можна зробити висновок, що метаеврестичні алгоритми дуже чутливі до різних вхідних аргументів і правильне їх обрання є необхідним кроком при використанні таких алгоритмів.

5. 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%.