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-12 Волков Вадим Всеволодович 27.10.2022
(шифр, прізвище, ім’я, по батькові)

Перевірив Сопов Олексій Олександрович 27.10.2022
(прізвище, ім’я, по батькові)

Київ 2022

ЗМІСТ

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.

№	**Задача і алгоритм**
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах).

3. ВИКОНАННЯ

1. Програмна реалізація алгоритму

1. Вихідний код

index.html

```
<!doctype html>
<html lang="en">
	<head>
		<title>L4V8</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="200"></td></tr>
				<tr><td>A</td><td><input type="number" id="alpha"
value="3"></td></tr>
				<tr><td>B</td><td><input type="number" id="beta"
value="2"></td></tr>
				<tr><td>P</td><td><input type="number" id="ro"
value="0.3"></td></tr>
				<tr><td>M</td><td><input type="number" id="antCount"
value="25"></td></tr>
				<tr><td>Lmin</td><td><input type="number" id="Lmin" valus="0"
readonly></td></tr>
			</table>
			<button id="init">Згенерувати</button>
			<button id="step">Крок</button>
			<button id="solve">Вирішити</button>
		</div>
	</body>
</html>
```

index.html

```
"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 int1, int2;
let iterations = 0;
let output = "";

function init() {
	alpha = +id("alpha").value;
	beta = +id("beta").value;
	ro = +id("ro").value;
	vertexCount = +id("vertexCount").value;
	antCount = +id("antCount").value;

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

	for(let i=0; i<vertexCount; i++) {
```

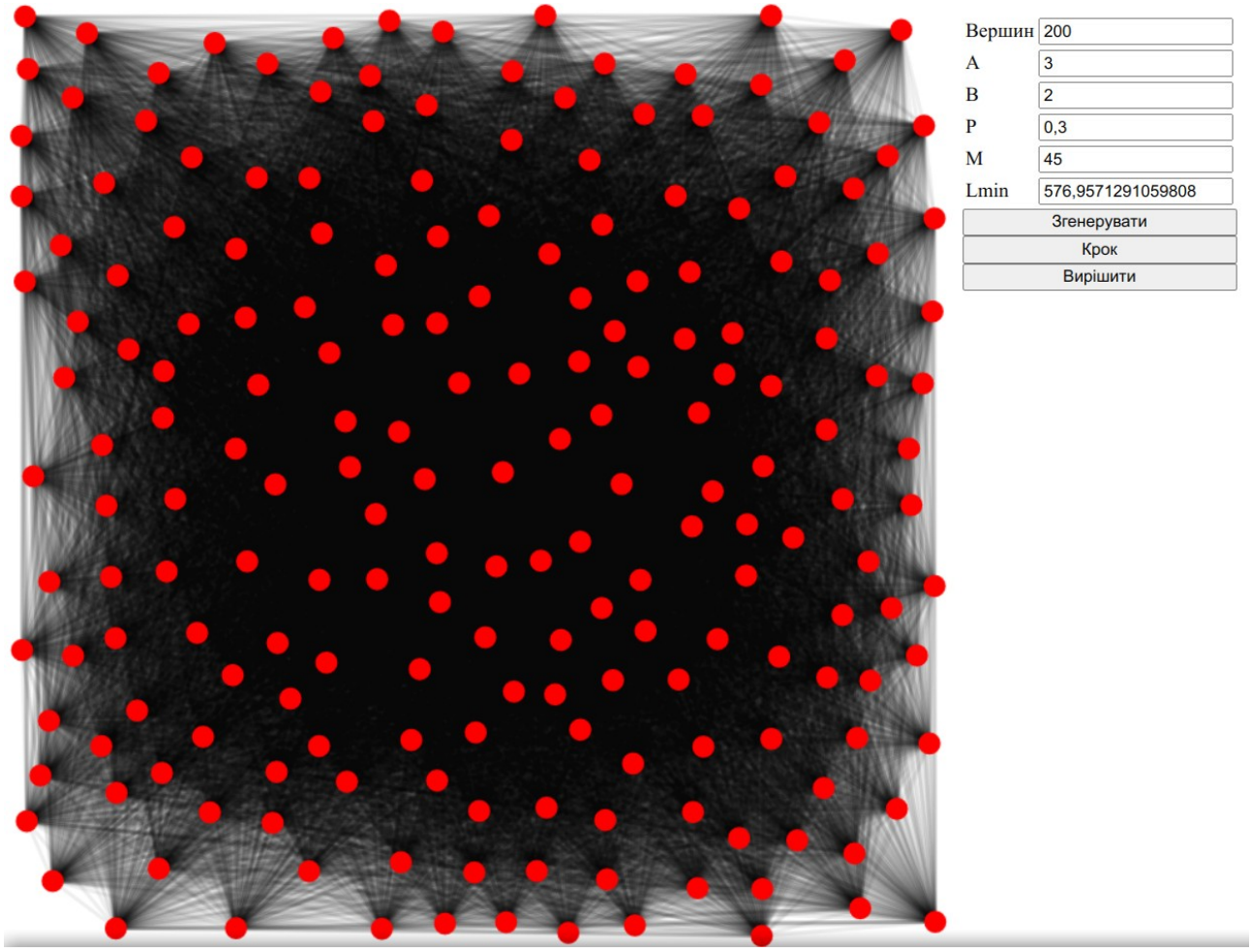
```
                let maxDist = 0;
                let maxPos;
                for(let j=0; j<20; j++) {
                        let pos = [Math.random()*35+0.5, Math.random()*35+0.5, 0]
                        let minDist = Infinity;
                        for(let k=0; k<points.length; k++) {
                                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);
                        distanceMatrixRow.push(dist);
                        feramoneMatrixRow.push(0.1);
                }
                distanceMatrix.push(distanceMatrixRow);
                feramoneMatrix.push(feramoneMatrixRow);
        }

        Lmin = 0;
        let now = 0;
        for(let i=0; i<vertexCount-1; i++) {
                let minDist = Infinity;
                let minIdx = 0;
                for(let j=1; j<vertexCount; j++) {
                        if(points[j][2] == 0) {
                                let dx = points[now][0]-points[j][0];
                                let dy = points[now][1]-points[j][1];
                                let dist = dx*dx + dy*dy;
                                if(dist < minDist) {
                                        minDist = dist;
                                        minIdx = j;
                                }
                        }
                }
                Lmin += Math.sqrt(minDist);
                //feramoneMatrix[now][minIdx] = 4;
                //feramoneMatrix[minIdx][now] = 4;
                now = minIdx;
                points[now][2] = Lmin;
        }
        //feramoneMatrix[now][0] = 4;
        //feramoneMatrix[0][now] = 4;
        id("Lmin").value = Lmin;
}

function draw() {
        ctx.save();
        ctx.scale(size / 256, size / 256);
        ctx.clearRect(0, 0, 256, 256);
        let scaler = 256 / 36;

        for(let i=0; i<vertexCount; i++) {
                for(let j=i+1; j<vertexCount; j++) {
                        let dx = points[i][0]-points[j][0];
                        let dy = points[i][1]-points[j][1];
                        if(feramoneMatrix[i][j] > 0.01) {
                                ctx.globalAlpha = Math.atan(feramoneMatrix[i][j]) / 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);
                                ctx.closePath();
                                ctx.stroke();
                        }
```

```
                }
        }
        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();
}

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);
        }

        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;

                for(let k=1; k<=vertexCount; k++) {
                        let newPos = 0;
                        if(k < vertexCount) {
                                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;
                        curPos = newPos;
                }
        }
        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];
                }
        }
```

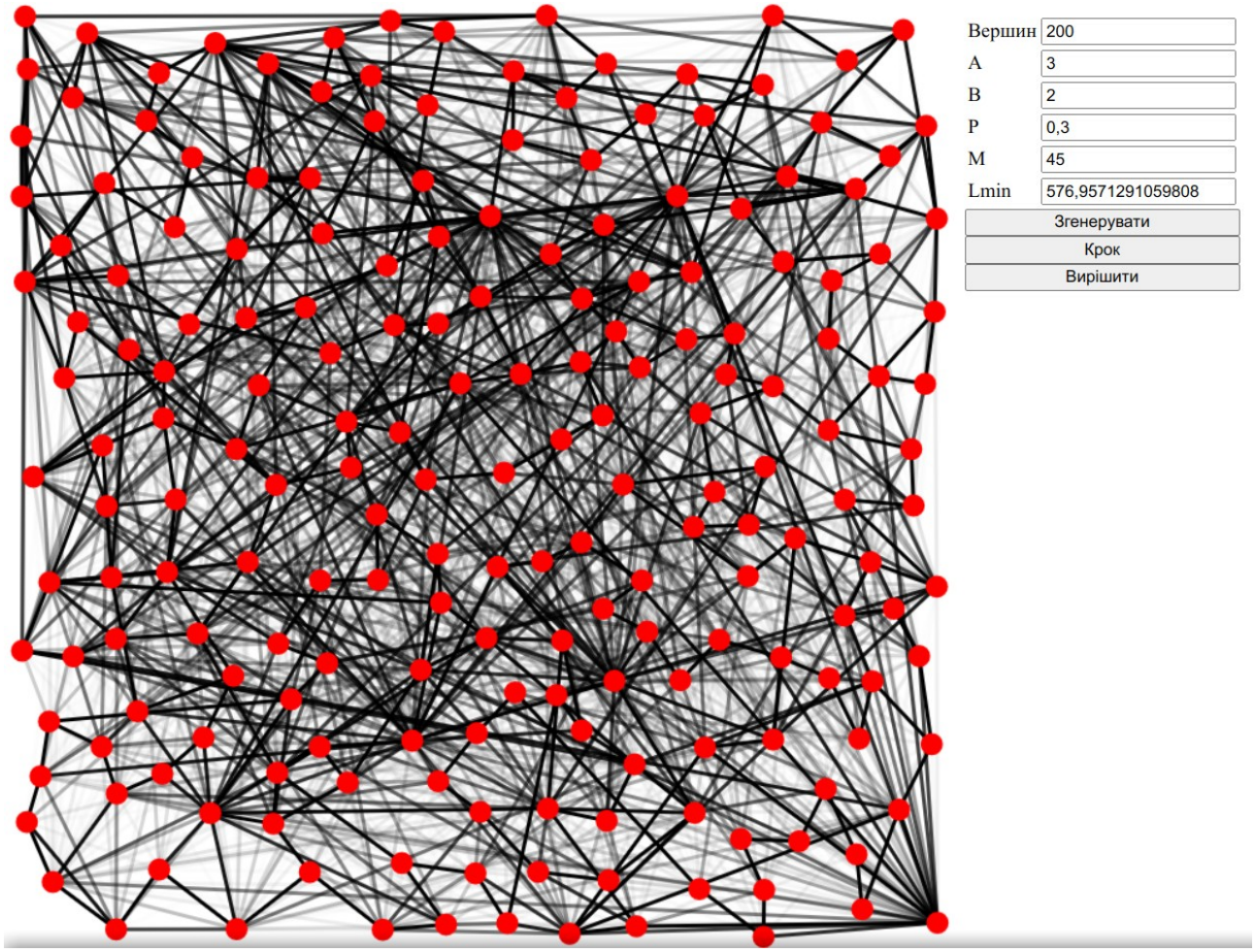
```
        if(int1) {
                iterations++;
                if(iterations % 20 == 0) {
                        let walked = evaluate();
                        output += (walked+"").replaceAll(".",",")+"\n";
                        console.log((iterations/20)+": "+walked);
                }
                if(iterations == 1000) {
                        console.log(output);
                        if(int1) {
                                clearInterval(int1);
                                clearInterval(int2);
                                int1 = null;
                                id("init").disabled = false;
                        }
                }
        }
}

function evaluate() {
        let walked = 0;
        let startPos = 0;
        let curPos = startPos;
        let visited = new Array(vertexCount).fill(false);
        visited[curPos] = true;

        for(let k=1; k<=vertexCount; k++) {
                let newPos = 0;
                if(k < vertexCount) {
                        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 {
                        newPos = startPos;
                }

                walked += distanceMatrix[curPos][newPos];
                visited[newPos] = true;
                curPos = newPos;
        }
        return walked;
}


id("step").addEventListener("click", () => {
        step();
        draw();
});
id("init").addEventListener("click", () => {
        init();
        draw();
});
id("solve").addEventListener("click", () => {
        if(int1) {
                clearInterval(int1);
                clearInterval(int2);
                int1 = null;
                id("init").disabled = false;
                id("solve").textContent = "Вирішити";
        } else {
                int1 = setInterval(step, 10);
                int2 = setInterval(draw, 100);
                id("init").disabled = true;
                id("solve").textContent = "Зупинити";
        }
});

init();
draw();
```

2. Виконання

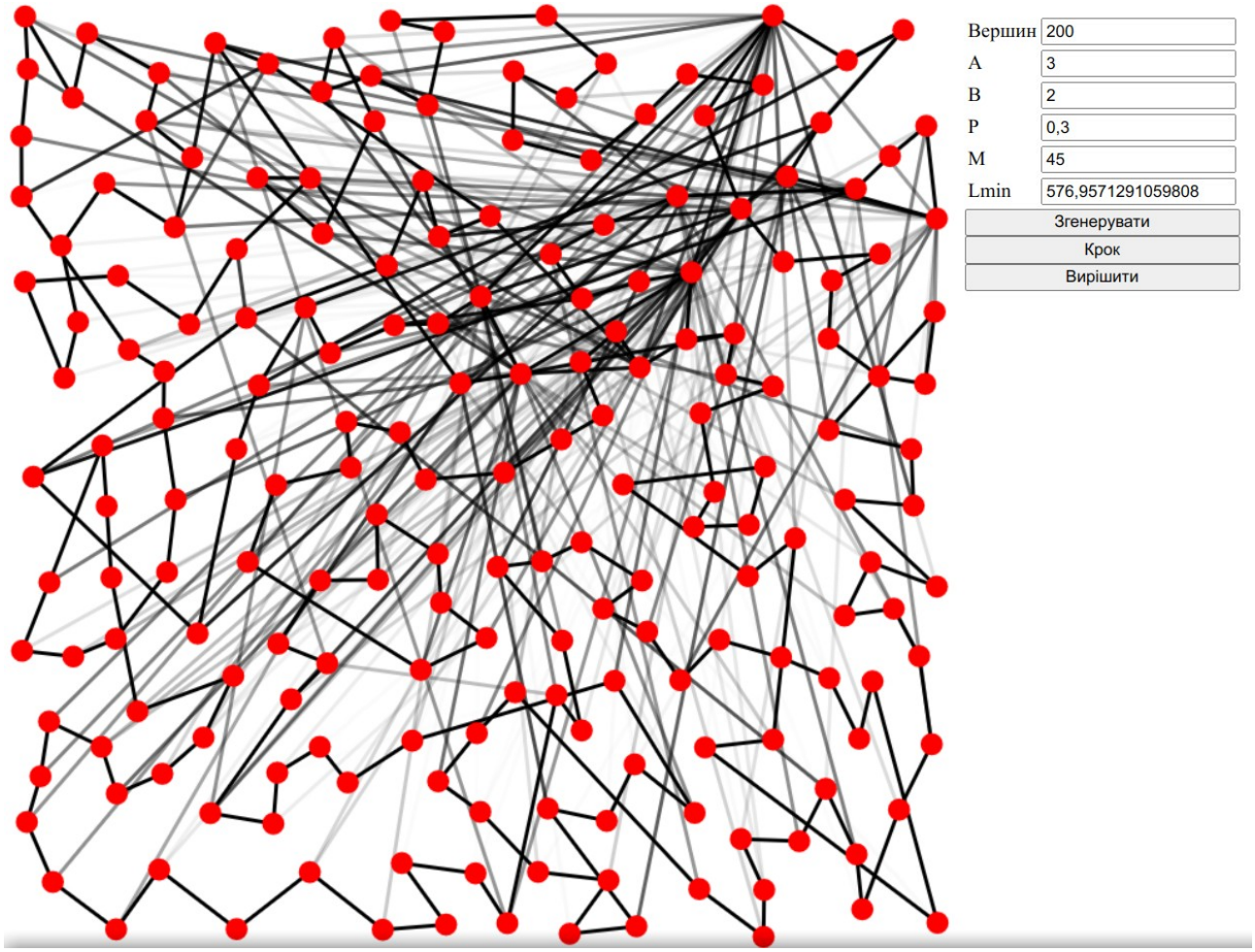
До початку:



Майже зразу після початку:



Декілька секунд пізніше:

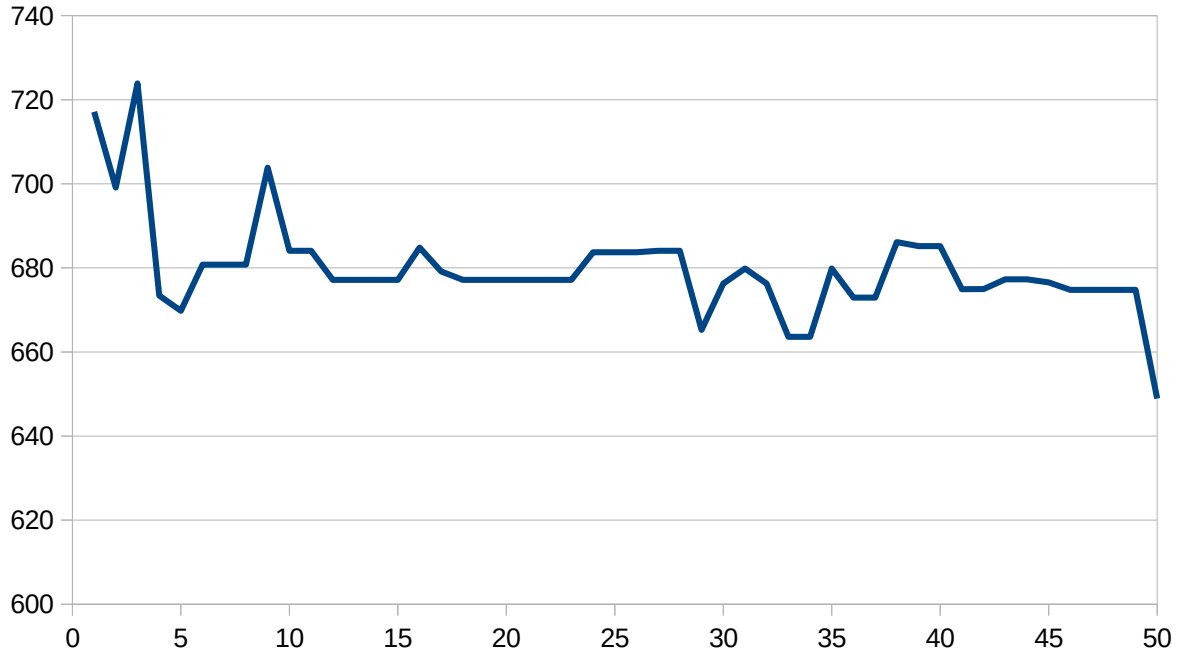


2. Тестування алгоритмів
    1. Значення цільової функції із збільшенням кількості ітерацій

Кількість ітерацій	Довжина шляху
20	717,12
40	699,12
60	723,89
80	673,4
100	669,77
120	680,77
140	680,77
160	680,77
180	703,83
200	684,07
220	684,07
240	677,15
260	677,15
280	677,15
300	677,15
320	684,82
340	679,16
360	677,15
380	677,15
400	677,15
420	677,15
440	677,15
460	677,15
480	683,72
500	683,72
520	683,72
540	684,07
560	684,07
580	665,27
600	676,24
620	679,85
640	676,24
660	663,59
680	663,59
700	679,85
720	672,93
740	672,93
760	686,12
780	685,17
800	685,17
820	674,94
840	674,98
860	677,3
880	677,3
900	676,55
920	674,77

940	674,77
960	674,77
980	674,77
1000	648,92

2. Графіки залежності розв’язку від числа ітерацій



4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з різними метаевристичними алгоритмами, а також створена реалізація на мові програмування javascript алгоритму комівояжера на основі метаевристичного мурашиного алгоритму, за допомогою якої було проведено дослідження роботи цього алгоритму. Результати було записано в таблицю в пункті 3.2.1 та на основі цих даних побудовано графік в пункті 3.2.2.

5. КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 23.10.2022 включно максимальний бал дорівнює – 5. Після 23.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 10%;
- програмна реалізація алгоритму – 60%;
- дослідження алгоритмів – 25%;
- висновок – 5%.