**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 № 3 з дисципліни

«Проектування алгоритмів»

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

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01Заранік Богдан*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2021

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 9](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 9](#_Toc51260920)

[3.1.1 Вихідний код 9](#_Toc51260921)

[3.1.2 Приклади роботи 9](#_Toc51260922)

[3.2 Тестування алгоритму 10](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 10](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 10](#_Toc51260925)

[Висновок 11](#_Toc51260926)

[Критерії оцінювання 12](#_Toc51260927)

# Мета лабораторної роботи

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

# 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** |
| 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 М = 45, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

package com.company;  
import com.company.GraphPackage.Graph;  
import com.company.GraphPackage.Way;  
  
import java.io.FileNotFoundException;  
  
import static com.company.Constants.*INF*;  
  
/\*  
5 4  
1 2 38  
1 3 47  
1 4 50  
1 5 45  
2 3 46  
2 4 16  
2 5 27  
3 4 8  
3 5 10  
4 5 42  
 \*/  
public class Main {  
 public static void main(String[] args) {  
 Graph graph = new Graph();  
 try {  
 graph.readFrom("input.txt");  
 }catch(FileNotFoundException e){  
 System.*out*.println("This file was not found!\n" + e);  
 }  
 //graph.print();  
 Way theBestWay = new Way(*INF*);  
 long timer = System.*currentTimeMillis*();  
 for(int i=1; i<=1000; i++){ //set to 1000  
 Way currBestWay = graph.iterate(2, 3, 0.3);  
 //updating the answer  
 if(theBestWay.compareTo(currBestWay) > 0){  
 theBestWay = currBestWay;  
 System.*err*.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<{==");  
 System.*err*.println("iteration " + i);  
 System.*err*.println("the best way now is " + theBestWay);  
 System.*err*.println("==}>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");  
 }  
 if(i % 20 == 0){  
 System.*out*.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<{==");  
 System.*out*.println("iteration " + i);  
 System.*out*.println("the best way now is " + theBestWay);  
 System.*out*.println("==}>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>");  
 }  
 }  
 System.*out*.println("THE BEST WAY IS: " + theBestWay);  
 System.*out*.println("Program worked for " + (System.*currentTimeMillis*() - timer));  
 }  
}

package com.company.GraphPackage;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.\*;  
  
import static java.lang.Math.abs;  
import static java.lang.Math.pow;  
  
public class Graph {  
 private Colony colony;  
 private int[][] dist;  
 private int[][] tay;  
 public void readFrom(String filePath) throws FileNotFoundException {  
 int numberOfVertexes;  
 int numberOfBeatles;  
 Scanner in = new Scanner(new File(filePath));  
 numberOfVertexes = in.nextInt();  
 numberOfBeatles = in.nextInt();  
  
 dist = new int[numberOfVertexes][numberOfVertexes];  
 tay = new int[numberOfVertexes][numberOfVertexes];  
  
 while(in.hasNextInt()){  
 int fromVert, toVert, price;  
 fromVert = in.nextInt();  
 toVert = in.nextInt();  
 price = in.nextInt();  
 //to make 0-indexation  
 fromVert--;  
 toVert--;  
 dist[fromVert][toVert] = dist[toVert][fromVert] = price;  
 colony = new Colony(numberOfBeatles);  
 }  
 //making tay array. gives start values [1..3]  
 for(int i = 0; i < numberOfVertexes; i++){  
 for(int j = i+1; j<numberOfVertexes; j++){  
 tay[i][j] = tay[j][i] = (abs(new Random().nextInt()))%3 + 1;  
 }  
 }  
 //making new colony  
 colony = new Colony(numberOfBeatles);  
  
 in.close();  
 }  
  
  
  
 /\*  
 \*This is main method.  
 \* It makes another new iteration,  
 \* changes tay array  
 \* and returns the best way after this iteration.  
 \*/  
 public Way iterate(int alpha, int beta, double p) {  
  
 List<Way> allTheWays = new ArrayList<>();  
 for(int subIt = 0; subIt < this.colony.getNumberOfBeatles(); subIt++){//making subiterations for each beatle  
  
 Way theWayOfCurrentBeatle = new Way(0); //the current way  
 int startVertex = (abs(new Random().nextInt())) % dist.length;  
 Set<Integer> usedVertexes = new HashSet<>();  
 int currentVertex = startVertex;  
  
 for(int v = 0; v < dist.length; v++){//loop for vertex to checkout  
 theWayOfCurrentBeatle.addNewVertex(currentVertex, (v==0) ? 0 : dist[theWayOfCurrentBeatle.getLastVertexInWay()][currentVertex]);  
 usedVertexes.add(currentVertex);  
 List<Integer> availableVertexes = new ArrayList<>();  
  
 for(int j = 0; j < dist.length; j++){//loop for vertexes to add into available  
 if(!usedVertexes.contains(j) && dist[currentVertex][j]>0){  
 availableVertexes.add(j);  
 }  
 }  
  
 //calculating P-function of available vertex  
  
 List<Double> probabilityOfAvailableVertex = new ArrayList<>();  
  
 double summary = 0;//summary of all (etha[i][k]^beta \* tay[i][k]^alpha)  
 for(int toVertex: availableVertexes){  
 double etha = 1. / dist[currentVertex][toVertex];  
 summary += pow(etha, beta) \* pow(tay[currentVertex][toVertex]+.0, alpha);  
 }  
 //probability calculating for every available vertex  
 for (int toVertex : availableVertexes) {  
 double etha = 1. / dist[currentVertex][toVertex];  
 double P = pow(etha, beta) \* pow(tay[currentVertex][toVertex] + .0, alpha) / summary;  
 probabilityOfAvailableVertex.add(P);  
 }  
 if(probabilityOfAvailableVertex.size()>0) {  
 /\*double randomZone = Math.random();  
 double currSum = 0;  
 int i = 0;  
 while( !(currSum < randomZone && currSum + probabilityOfAvailableVertex.get(i) > randomZone)  
 && i < probabilityOfAvailableVertex.size() ){  
 currSum += probabilityOfAvailableVertex.get(i);  
 i++;  
 }  
 i = Math.min(i, probabilityOfAvailableVertex.size()-1);//0.1 + 0.1 + 0.1 != 0.3 problems with accuracy may be  
 currentVertex = availableVertexes.get(i);\*/  
  
 //for greedy beatle  
 double maxProb = Collections.max(probabilityOfAvailableVertex);  
 int maxInProbabilityIndex = probabilityOfAvailableVertex.indexOf(maxProb);  
 currentVertex = availableVertexes.get(maxInProbabilityIndex);  
 }  
 }  
 theWayOfCurrentBeatle.addNewVertex(startVertex, dist[theWayOfCurrentBeatle.getLastVertexInWay()][startVertex]);//start vertex is also the last  
 allTheWays.add(theWayOfCurrentBeatle);  
  
 //changing tay  
 int deltaTay = dist.length \* 150 / theWayOfCurrentBeatle.getCost();  
 List<Integer> vertexesOfCurrentWay = theWayOfCurrentBeatle.getVertexes();//needed to calculate tay difference;  
 for(int i=0; i<vertexesOfCurrentWay.size()-1; i++){  
 int \_tay = tay[vertexesOfCurrentWay.get(i)][vertexesOfCurrentWay.get(i+1)];  
 tay[vertexesOfCurrentWay.get(i)][vertexesOfCurrentWay.get(i+1)] = (int)((1-p) \* \_tay + deltaTay);  
 }  
 }  
  
 return Collections.min(allTheWays);  
 }  
  
 public void print(){  
 for(int i=0; i<dist.length; i++){  
 for(int j=0; j<dist[i].length; j++){  
 System.out.format("%4d", dist[i][j]);  
 }  
 System.out.println("");  
 }  
 }  
  
 public void printTay(){  
 for(int i=0; i<tay.length; i++){  
 for(int j=0; j<tay[i].length; j++){  
 System.out.format("%4d", tay[i][j]);  
 }  
 System.out.println("");  
 }  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

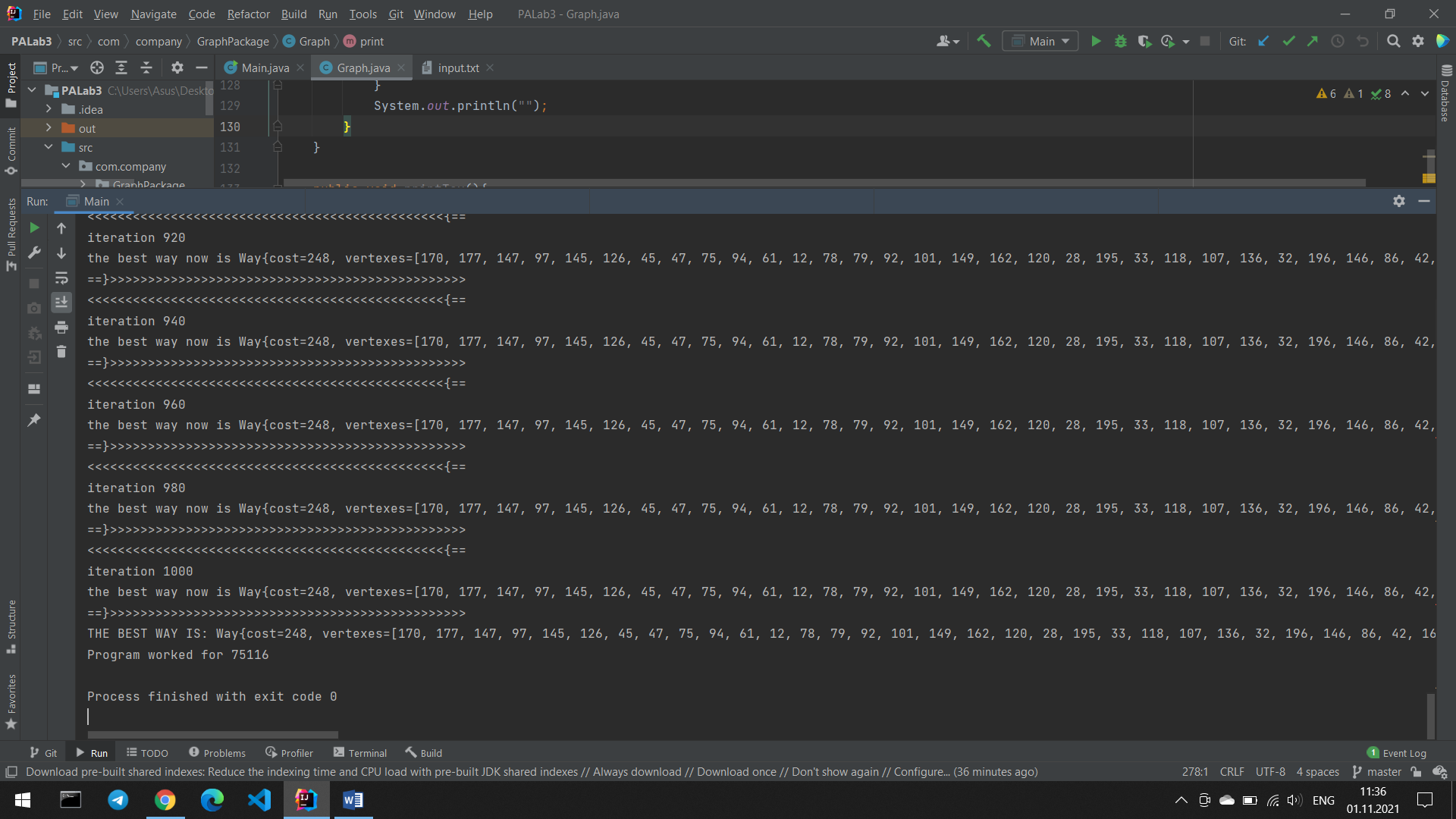


Рисунок 3.1 –

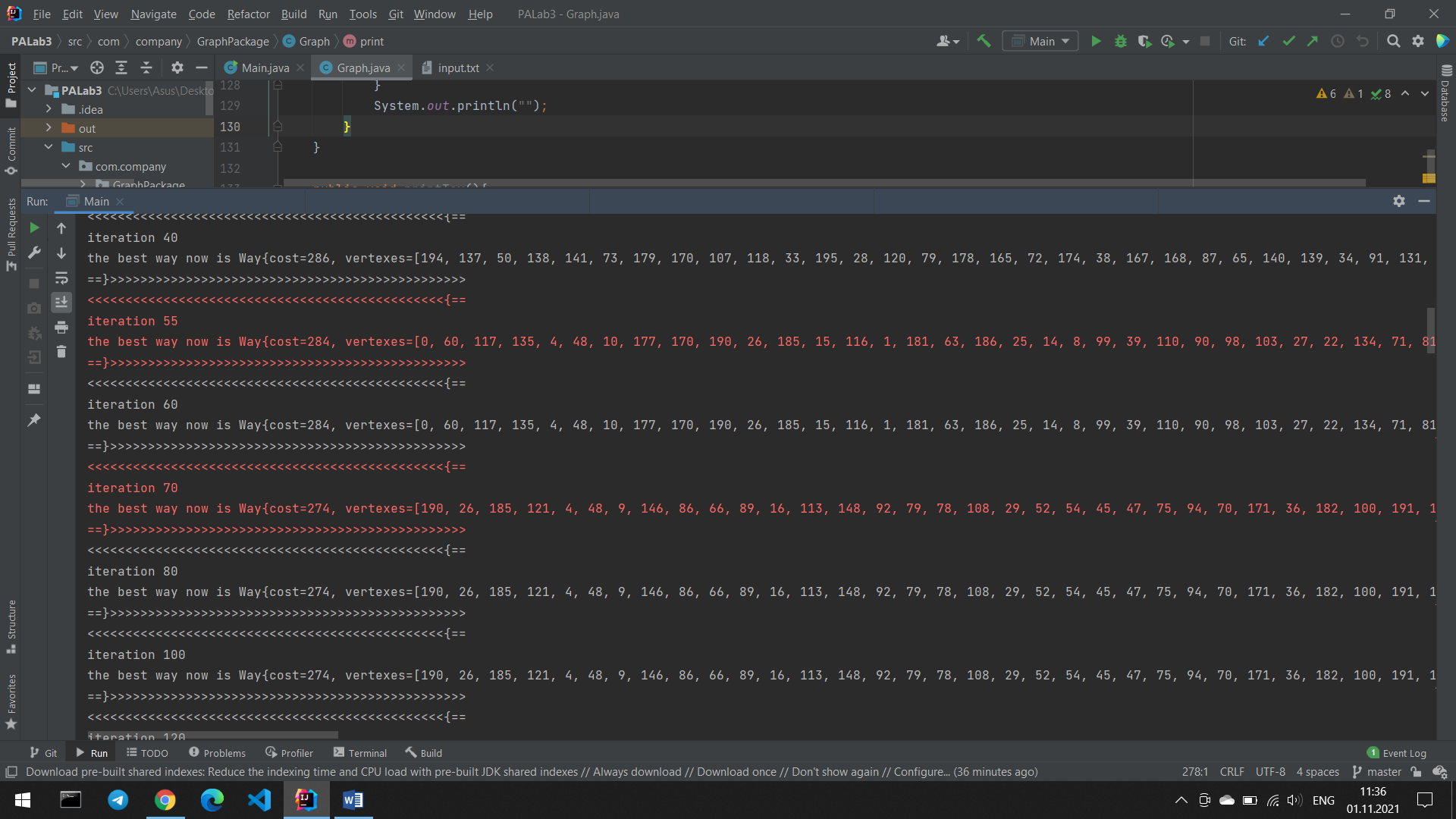


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| 20 | 327 |
| 40 | 286 |
| 60 | 284 |
| 80 | 274 |
| 100 | 274 |
| 120 | 274 |
| 140 | 274 |
| 160 | 274 |
| 180 | 274 |
| 200 | 274 |
| 220 | 265 |
| 240 | 265 |
| 260 | 265 |
| 280 | 265 |
| 300 | 265 |
| 320 | 265 |
| 340 | 265 |
| 360 | 265 |
| 380 | 265 |
| 400 | 265 |
| 420 | 265 |
| 440 | 264 |
| 460 | 264 |
| 480 | 264 |
| 500 | 264 |
| 520 | 260 |
| 540 | 260 |
| 560 | 260 |
| 580 | 260 |
| 600 | 248 |
| 620 | 248 |
| 640 | 248 |
| 660 | 248 |
| 680 | 248 |
| 700 | 248 |
| 720 | 248 |
| 740 | 248 |
| 760 | 248 |
| 780 | 248 |
| 800 | 248 |
| 820 | 248 |
| 840 | 248 |
| 860 | 248 |
| 880 | 248 |
| 900 | 248 |
| 920 | 248 |
| 940 | 248 |
| 960 | 248 |
| 980 | 248 |
| 1000 | 248 |

Ответ: 248

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

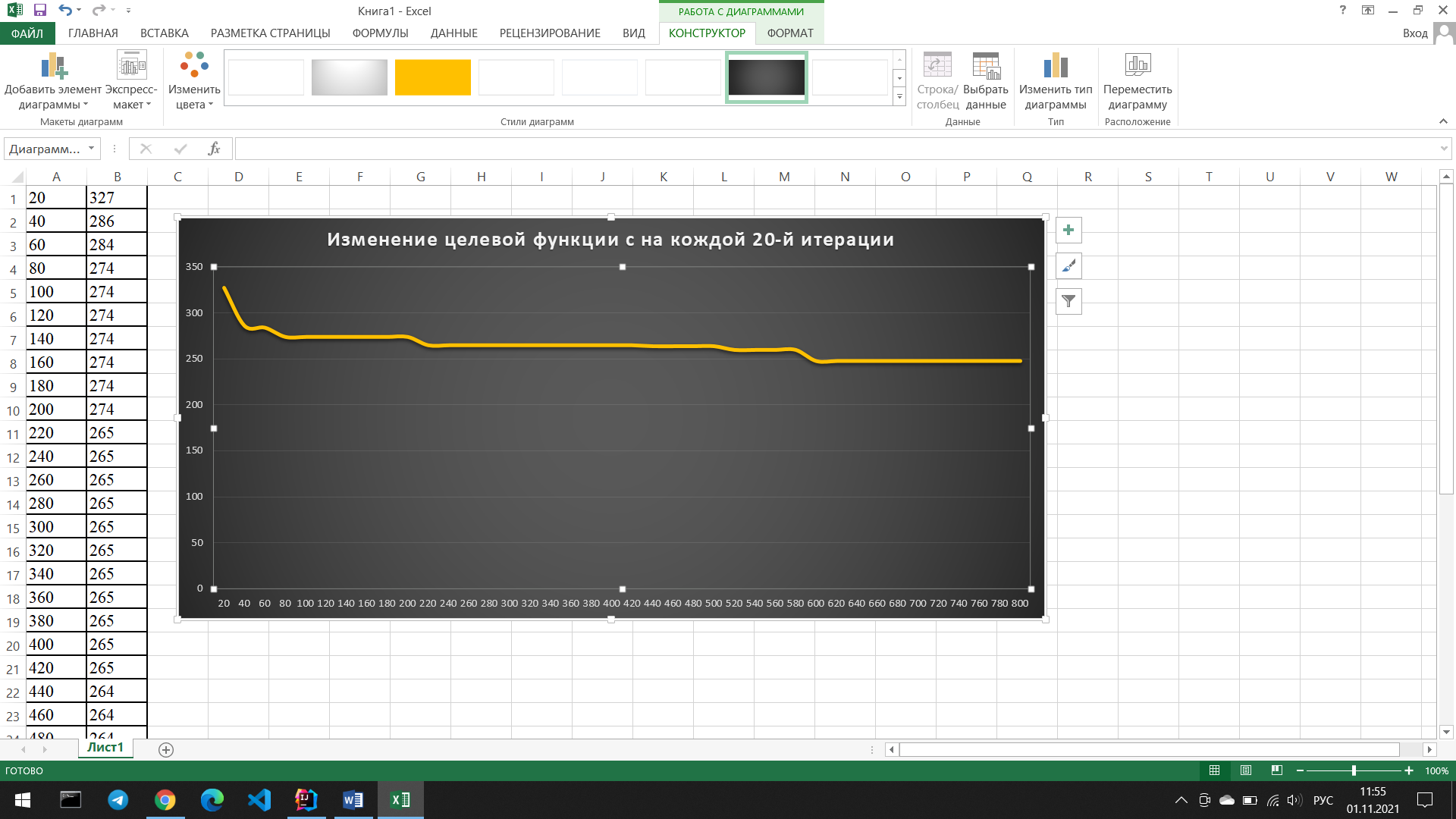


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи я навчився реалізації досить складних алгоритмів штучного інтелекту, а саме алгоритму мурашиного пошуку, на прикладі задачі про пошук оптимального шляху у задачі комівояжера. Алгоритм із початковими даними: 200 вершин, відстань між вершинами випадкова від до 50, мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах) показує відносно високу продуктивність на максимальному тесті та знаходить оптимальний(або майже оптимальний) шлях.

Проведено аналіз виконаної роботи: виведено найкращий шлях кожного 20-го з-поміж 1000 поколінь. Значення цільової функції не стрімко, але спадає, що є гарним результатом. Графік а також табуляція цільової функції наведено вище.

Критерії оцінювання

При здачі лабораторної роботи до 5.11.2021 включно максимальний бал дорівнює – 5. Після 5.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.