**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-14 Радзівіло Валерія Артемівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_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** |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

Main.java

package com.labs;  
  
import com.labs.Additionals.Graph;  
import com.labs.Additionals.main\_tsl;  
import com.labs.Additionals.print\_info;  
  
public class Main {  
 public static void main(String[] args) {  
 int antAmount = 45,  
 wildAntAmount = 10,  
 citiesAmount = 250,  
 alpha = 4,  
 beta = 2;  
 double rho = 0.3;  
  
 Graph gr = new Graph(citiesAmount);  
 gr.make\_distances(gr);  
 print\_info.*print\_task*(antAmount, wildAntAmount, citiesAmount, alpha, beta, rho);  
  
 int this\_min = main\_tsl.*start\_program*(-1, gr, antAmount, wildAntAmount, citiesAmount);  
  
  
 System.*out*.println("\nMinimum from all iterations: "+ this\_min);  
 Graph.*greedyLength*(gr,citiesAmount);  
  
  
  
 }  
  
  
  
}

main\_tsl.java

package com.labs.Additionals;  
  
import java.util.ArrayList;  
import java.util.Collections;  
  
public class main\_tsl {  
  
 public static int start\_program(int start\_pos\_not\_wild,Graph gr,int antAmount, int wildAntAmount, int citiesAmount) {  
 // set pheromone  
 double rho = 0.3;  
 double[][] pheromone = new double[citiesAmount][citiesAmount];  
 for (int i = 0; i < citiesAmount; i++) {  
  
 for (int j = 0; j < citiesAmount; j++)  
 pheromone[i][j] = 1-rho;  
  
 pheromone[i][i] = 1;  
 }  
  
 int Lmin = citiesAmount \* 40; // max possible if all have distance of 40 - max  
 int Lpr = citiesAmount\*40;  
 ArrayList<Integer>minis = new ArrayList<>();  
 ArrayList<City> cities = Ant.*fill\_cities*(citiesAmount);  
 double[][] updated\_pheromones = pheromone.clone();  
 for (int iterator = 1; iterator <= 1000; iterator++) {  
 // set pheromones for this colony  
 Ant.*pheromone* = Ant.*copy\_matrix*(updated\_pheromones);  
 ArrayList<City> starting\_positions = Ant.*set\_ants*(new ArrayList<>(cities));  
  
 // запускаємо диких, хай побігають  
 for (int i = 0; i < wildAntAmount; i++) {  
 Ant ant = new Ant(true, i, starting\_positions.get(i));  
 ant.start\_travelling(gr, new ArrayList<>(cities), start\_pos\_not\_wild - 1,updated\_pheromones);  
 if (ant.distance\_made < Lmin) Lmin = ant.distance\_made;  
 }  
  
  
  
 // let not wild out  
  
 for (int i = wildAntAmount; i < antAmount; i++) {  
 City st\_pos = starting\_positions.get(i);  
 if (start\_pos\_not\_wild != -1 && i == wildAntAmount)  
 st\_pos = cities.get(start\_pos\_not\_wild);  
 if (start\_pos\_not\_wild != -1 && i == wildAntAmount + 1)  
 st\_pos = cities.get(start\_pos\_not\_wild + 2);  
  
 Ant ant = new Ant(false, i, st\_pos);  
  
 updated\_pheromones = ant.start\_travelling(gr, new ArrayList<>(cities), start\_pos\_not\_wild - 1, updated\_pheromones);  
  
 if (ant.distance\_made < Lmin)  
 {  
 Lmin = ant.distance\_made;  
 }  
  
 }  
 minis.add(Lmin);  
 if (iterator % 20 == 0)  
 {  
 System.*out*.println("Iteration №"+iterator+ "\nLpr from these ants: " + Lmin);  
 System.*out*.println("Min distance: "+Collections.*min*(minis));  
 }  
  
  
 }  
 return Collections.*min*(minis);  
 }  
  
  
}

Ant.java

package com.labs.Additionals;  
  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Random;  
  
public class Ant {  
 int alpha = 4;  
 int beta = 2;  
  
 boolean isWild;  
 int id;  
 int distance\_made = 0;  
  
 City start\_pos;  
 ArrayList<City>visited\_places = new ArrayList<>();  
  
 static double[][] *pheromone*;  
  
  
 public Ant(boolean isWild, int id,City start\_pos)  
 {  
 this.id = id;  
 this.isWild = isWild;  
 this.start\_pos=start\_pos;  
 }  
  
 public double[][] start\_travelling(Graph graph, ArrayList<City>cities, int for\_test\_not\_zero, double[][] pheromone\_new)  
 {  
 double[][] pheromone\_old = *copy\_matrix*(*pheromone*);  
 double [][] new\_updated = *copy\_matrix*(pheromone\_new);  
 double [][] old\_shit = *copy\_matrix*(pheromone\_old);  
 if(!this.isWild)  
 {  
 new\_updated = make\_not\_random\_travel(graph,cities,pheromone\_new,pheromone\_old);  
 }  
  
 else {  
 // make random decisions  
  
 make\_random\_travelling(graph, cities);  
 }  
 *pheromone* = old\_shit.clone();  
  
 if(for\_test\_not\_zero == 0) {  
 System.*out*.print("ROUTE OF THIS ANT: ");  
 for (City c : visited\_places)  
 System.*out*.print(c.getId() + " ");  
 System.*out*.println();  
 System.*out*.println("DISTANCE: " + distance\_made);  
 if (!this.isWild) {  
 for (double[] c : *pheromone*)  
 System.*out*.println(Arrays.*toString*(c));  
 }  
 }  
 return new\_updated;  
 }  
  
 public static ArrayList<City> set\_ants(ArrayList<City>cities)  
 {  
 Random random = new Random();  
 ArrayList<City> placed = new ArrayList<>();  
 while(!cities.isEmpty())  
 {  
 City rand\_city = City.*get\_rand\_city*(cities);  
 placed.add(rand\_city);  
 cities.remove(cities.indexOf(rand\_city));  
 }  
 return placed;  
  
 }  
  
  
  
 static ArrayList<City> fill\_cities(int citiesAmount)  
 {  
 ArrayList<City> cities = new ArrayList<>();  
 for(int i =0; i< citiesAmount; i++)  
 {  
 cities.add(new City(i,i+citiesAmount^3));  
 }  
 return cities;  
 }  
  
 void make\_random\_travelling(Graph graph, ArrayList<City>cities)  
 {  
 int counter = 0;  
 visited\_places.add(this.start\_pos);  
 cities.remove(this.start\_pos);  
 while (!cities.isEmpty())  
 {  
 Random random = new Random();  
 int get\_city\_ind= random.nextInt(cities.size());  
 City get\_city = cities.get(get\_city\_ind);  
  
 City previously\_visited;  
 if(visited\_places.size()==0)  
 {  
 previously\_visited = this.start\_pos;  
 counter--;  
 }  
 else  
 previously\_visited = visited\_places.get(counter);  
 distance\_made+=graph.travelling\_distance(graph,previously\_visited.getId(),get\_city.getId());  
 visited\_places.add(get\_city);  
 cities.remove(get\_city\_ind);  
 counter++;  
  
 }  
 visited\_places.add(this.start\_pos);  
 distance\_made+=graph.travelling\_distance(graph,visited\_places.get(visited\_places.size()-2).getId(),this.start\_pos.getId());  
  
  
 }  
  
 double[][] make\_not\_random\_travel(Graph gr, ArrayList<City> cities, double[][] previously\_updated, double[][]old\_pheromone)  
 {  
 visited\_places.add(this.start\_pos);  
 cities.remove(this.start\_pos);  
 City next\_city = null;  
 for(int i =0; i<cities.size();i++)  
 {  
 City start\_city = visited\_places.get(visited\_places.size()-1);  
 ArrayList<City> available\_cities = get\_available\_cities(cities);  
 next\_city = find\_min\_probability(available\_cities, gr,start\_city, old\_pheromone);  
 distance\_made+=gr.travelling\_distance(gr,start\_city.getId(),next\_city.getId());  
 visited\_places.add(next\_city);  
 }  
 assert next\_city != null;  
 distance\_made+= gr.travelling\_distance(gr,next\_city.getId(),this.start\_pos.getId());  
 visited\_places.add(this.start\_pos);  
 return update\_pheromones(previously\_updated);  
  
  
 }  
  
 double[][] update\_pheromones(double[][] previously\_updated)  
 {  
 for(int i =1; i < visited\_places.size();i++)  
 {  
 int from\_id = visited\_places.get(i-1).getId();  
 int to\_id = visited\_places.get(i).getId();  
 double new\_pho =(1.0/distance\_made);  
  
 previously\_updated[from\_id][to\_id] += new\_pho;  
 previously\_updated[to\_id][from\_id] += new\_pho;  
 previously\_updated[i-1][i-1] = 1;  
  
 }  
  
 return previously\_updated;  
 }  
  
  
 ArrayList<City> get\_available\_cities(ArrayList<City> cities)  
 {  
 ArrayList<City> availables = new ArrayList<>();  
 for(City c : cities)  
 {  
 if(!visited\_places.contains(c))  
 availables.add(c);  
 }  
 return availables;  
 }  
  
  
 City find\_min\_probability(ArrayList<City>cities\_left\_to\_visit, Graph gr, City start\_city, double[][] old\_pheromone)  
 {  
 double min\_prob = 0;  
 double prob = 0;  
 int[]distances = new int[cities\_left\_to\_visit.size()];  
 double[] down\_values = new double[cities\_left\_to\_visit.size()];  
 int from\_id = start\_city.getId();  
 for(int i =0;i < cities\_left\_to\_visit.size();i++)  
 {  
 int to\_id = cities\_left\_to\_visit.get(i).getId();  
 distances[i] = gr.travelling\_distance(gr,start\_city.getId(),cities\_left\_to\_visit.get(i).getId());  
 down\_values[i] = Math.*pow*(old\_pheromone[from\_id][to\_id],alpha)\*Math.*pow*(1.0/distances[i],beta);  
  
 }  
 double sum\_of\_down\_values = 0;  
 for (double down\_value : down\_values) {  
 sum\_of\_down\_values += down\_value;  
 }  
  
 City min\_prob\_city = cities\_left\_to\_visit.get(0);  
 ArrayList<Double> PROBS = new ArrayList<>();  
 for(City c : cities\_left\_to\_visit)  
 {  
 int to\_id = c.getId();  
  
  
 double upper = Math.*pow*(old\_pheromone[from\_id][to\_id],alpha)\*Math.*pow*(1.0/distances[cities\_left\_to\_visit.indexOf(c)],beta);  
  
 prob = (upper/sum\_of\_down\_values);  
 PROBS.add(prob);  
 }  
 ArrayList<Double> sums = new ArrayList<>();  
 for(int k =0; k<PROBS.size();k++)  
 {  
 sums.add(sum\_till\_end(PROBS,k));  
 }  
 sums.add(0.);  
  
  
 Random random = new Random();  
 double val = random.nextDouble();  
 int right\_sum\_pos = 0;  
 for(int l =0; l<sums.size()-1;l++)  
 {  
 if(val<=sums.get(l) && val>sums.get(l+1))  
 {  
 right\_sum\_pos = l;  
 }  
  
 }  
 min\_prob\_city = cities\_left\_to\_visit.get(right\_sum\_pos);  
  
  
 return min\_prob\_city;  
 }  
  
 double sum\_till\_end(ArrayList<Double> probs, int pos)  
 {  
 double answer =0.;  
 for(int i = pos; i<probs.size();i++)  
 {  
 answer+=probs.get(i);  
 }  
 return answer;  
 }  
  
  
 static double[][]copy\_matrix(double[][] old\_matr)  
 {  
 double[][] new\_matr = new double[old\_matr.length][old\_matr.length];  
 for(int i =0; i< old\_matr.length;i++)  
 {  
 for(int j =0; j<old\_matr.length;j++)  
 {  
 new\_matr[i][j] = old\_matr[i][j];  
 }  
  
 }  
 return new\_matr;  
 }  
  
  
}

City.java

package com.labs.Additionals;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
public class City {  
 int id;  
 int value;  
  
 City(int id, int value)  
 {  
 this.id = id;  
 this.value = value;  
 }  
 int getId(){  
 return id;  
 }  
  
 public static City get\_rand\_city(ArrayList<City> cities)  
 {  
 Random random = new Random();  
 int pos = random.nextInt(cities.size());  
 return cities.get(pos);  
 }  
  
  
}

Graph.java

package com.labs.Additionals;  
  
import java.util.ArrayList;  
import java.util.Random;  
  
public class Graph {  
 private int V;  
 private ArrayList<Integer> adj[]; //Adjacency Lists  
  
 public ArrayList<Integer> get\_adj(int V)  
 {  
 return adj[V];  
 }  
  
 public void setAdj(int v, ArrayList<Integer>arr)  
 {  
 adj[v] = new ArrayList<Integer>(arr);  
 }  
  
  
 public void print\_graph()  
 {  
 System.*out*.println(V);  
 for(int i =0; i< V;i++) {  
 if(!adj[i].isEmpty())  
 for(int j : get\_adj(i))  
 {  
 System.*out*.print(j+" ");  
 }  
 else  
 i = V;  
 System.*out*.println();  
 }  
  
 }  
  
 // Constructor  
 public Graph(int v) {  
 V = v;  
 adj = new ArrayList[v];  
 for (int i = 0; i < v; ++i)  
 adj[i] = new ArrayList<>();  
 }  
  
 // Function to add an edge into the graph  
 void addEdge(int v, int w) {  
 adj[v].add(w);  
 }  
 void changeEdge(int v, int index, int new\_value) {  
 adj[v].set(index,new\_value);  
 }  
  
 public void make\_distances(Graph gr)  
 {  
 for(int i =0; i< V;i++)  
 {  
 for(int j =0; j<V;j++) {  
 gr.addEdge(i,0);  
 }  
 }  
  
 Random random\_dist = new Random();  
 for(int i =0; i< V;i++)  
 {  
 for(int j =0; j<V;j++) {  
 if (gr.get\_adj(i).get(j) == 0) {  
 if (i == j)  
 gr.changeEdge(i, j,0);  
 else {  
 int rand\_dist = random\_dist.nextInt(40)+1;  
 gr.changeEdge(i,j,rand\_dist);  
 gr.changeEdge(j,i, rand\_dist);  
 }  
  
 }  
 }  
 }  
  
  
 }  
  
 public int travelling\_distance(Graph graph, int place\_one , int place\_two)  
 {  
 return graph.get\_adj(place\_one).get(place\_two);  
 }  
  
  
 public static Graph ready\_graph(int[][]distances)  
 {  
 Graph gr = new Graph(distances.length);  
  
  
 for(int i = 0; i< distances.length;i++)  
 {  
 gr.setAdj(i,*int\_arr\_to\_list*(distances[i]));  
 }  
  
 return gr;  
  
 }  
  
 static ArrayList<Integer> int\_arr\_to\_list(int[] arr)  
 {  
 ArrayList<Integer>answer = new ArrayList<>();  
 for (int j : arr) {  
 answer.add(j);  
 }  
 return answer;  
 }  
  
 int find\_min\_in\_adj(int v, ArrayList<Integer>visited)  
 {  
 ArrayList<Integer> ar = new ArrayList<>(this.get\_adj(v));  
 int min = ar.size()\*1000000;  
 int pos = 0;  
 for(int i=0;i<ar.size();i++)  
 if(ar.get(i)<min && i!=0 && !visited.contains(i))  
 {  
 min = ar.get(i);  
 pos = i;  
 }  
  
 return pos;  
 }  
  
  
 public static void greedyLength(Graph graph, int citiesCount)  
 {  
 int min\_dist =0;  
  
 ArrayList<Integer>visited = new ArrayList<>();  
 visited.add(0);  
 for(int k =0; k<citiesCount-1;k++) {  
 int val = graph.find\_min\_in\_adj(visited.get(visited.size() - 1),visited);  
 visited.add(val);  
 min\_dist+=graph.travelling\_distance(graph,visited.get(visited.size()-2),visited.get(visited.size()-1));  
  
 }  
 visited.add(0);  
 min\_dist+=graph.travelling\_distance(graph,visited.get(visited.size()-2),visited.get(visited.size()-1));  
  
// System.out.println("Greedy path: "+ visited);  
 System.*out*.println("Greedy Lmin: "+ min\_dist);  
  
 }  
  
  
}

print\_info.java

package com.labs.Additionals;  
  
public class print\_info {  
  
 public static void print\_task(int antAmount,int wildAntAmount, int citiesAmount, int alpha, int beta, double rho)  
 {  
  
 System.*out*.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n TASK:");  
 System.*out*.println("ANT COLONY OPTIMIZATION");  
 System.*out*.println("Amount of ants: "+antAmount);  
 System.*out*.println("Wild ants: "+wildAntAmount);  
 System.*out*.println("Amount of cities: "+citiesAmount);  
 System.*out*.println("Alpha: "+alpha);  
 System.*out*.println("Beta: "+beta);  
 System.*out*.println("Pheromone evaporation rate(rho): "+rho);  
 System.*out*.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  
  
  
 }  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

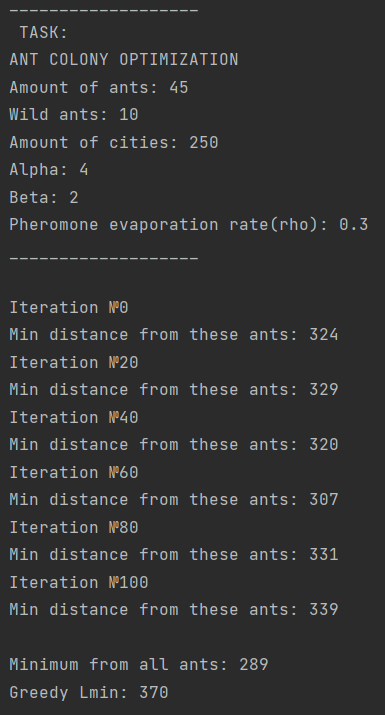


Рисунок 3.1 –

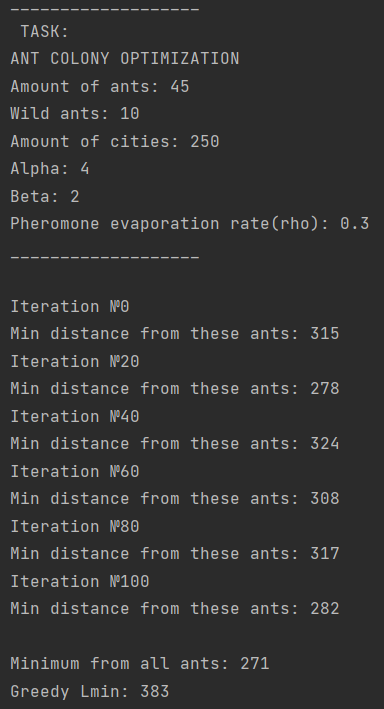


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

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

|  |  |
| --- | --- |
| Iteration | Lmin |
| 20 | 556 |
| 40 | 411 |
| 60 | 359 |
| 80 | 290 |
| 100 | 283 |
| 120 | 283 |
| 140 | 283 |
| 160 | 283 |
| 180 | 283 |
| 200 | 275 |
| 220 | 275 |
| 240 | 275 |
| 260 | 275 |
| 280 | 275 |
| 300 | 275 |
| 320 | 275 |
| 340 | 270 |
| 360 | 270 |
| 380 | 270 |
| 400 | 270 |
| 420 | 267 |
| 440 | 267 |
| 460 | 267 |
| 480 | 267 |
| 500 | 267 |
| 520 | 267 |
| 540 | 267 |
| 560 | 267 |
| 580 | 266 |
| 600 | 264 |
| 620 | 264 |
| 640 | 264 |
| 660 | 264 |
| 680 | 264 |
| 700 | 264 |
| 720 | 264 |
| 740 | 264 |
| 760 | 264 |
| 780 | 264 |
| 800 | 264 |
| 820 | 264 |
| 840 | 264 |
| 860 | 264 |
| 880 | 263 |
| 900 | 263 |
| 920 | 263 |
| 940 | 263 |
| 960 | 263 |
| 980 | 263 |
| 1000 | 263 |

### Графіки залежності розв'язку від числа ітерацій

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

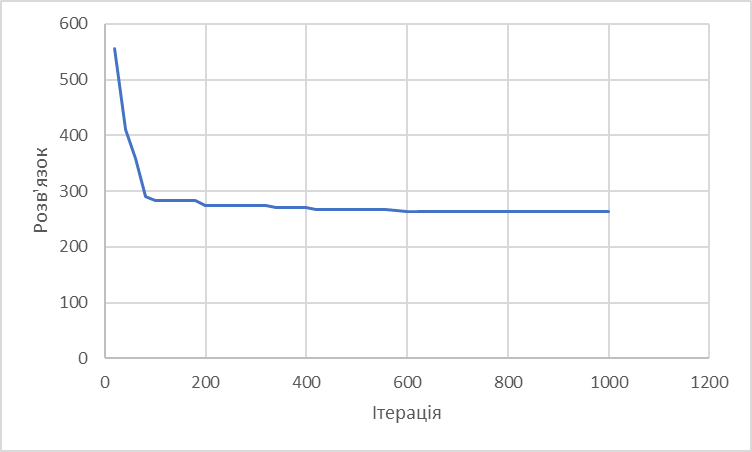


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи було вивчено основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою. Була розглянута задача комівояжера, яка має в собі ціль в знаходженні найкоротшого шляху від пункту А через всі міста і назад в пункт А. Вирішувалась вона через мурашиний алгоритм. Також була проаналізована відповідність між кількістю ітерацій і результатом розв’язку, що призвело до висновку – велика кількість ітерацій не сильно впливає на відповідь, ще на початку значення падає майже до мінімального, а потім незначимо змінюється.

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%.