**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)

*ІТ-04 Стрільчук Михайло Васильович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

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** |
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 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). |
| 9 | Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 12 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 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). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |

# Виконання

## Програмна реалізація алгоритму

https://github.com/Mishanych/Algorithms/tree/main/laba3

### Вихідний код

Program.cs

using System;

namespace Laba3

{

class Program

{

static void Main(string[] args)

{

StartProgram.Init();

Answer[] answers = new Answer[MainParams.COLONY\_LIFETIME];

bool flag = false;

//for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

//{

// for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

// {

// Console.Write(MainParams.Distances[i,j] + " ");

// }

// Console.WriteLine();

// Console.WriteLine();

//}

for (int i = 0; i < MainParams.COLONY\_LIFETIME; i++)

{

StartProgram.PlaceAnts();

answers[i] = new Answer();

MainParams.UpdatedHormoneConcentration = new double[MainParams.AMOUNT\_OF\_VERTICES, MainParams.AMOUNT\_OF\_VERTICES];

for (int j = 0; j < StartProgram.AllAnts.Length; j++)

{

var indexOfVertex = 0;

while (StartProgram.AllAnts[j].VisitedVertices.Count != MainParams.AMOUNT\_OF\_VERTICES)

{

var max = (double)int.MinValue;

for (int v = 0; v < MainParams.AMOUNT\_OF\_VERTICES; v++)

{

if (!StartProgram.AllAnts[j].VisitedVertices.Contains(v) && v != StartProgram.AllAnts[j].InitialVertex)

{

var probability = CountWayProbability(StartProgram.AllAnts[j], StartProgram.AllAnts[j].CurrentVertex, v);

if (probability > max)

{

max = probability;

indexOfVertex = v;

flag = true;

}

}

}

if (flag)

{

StartProgram.AllAnts[j].VisitedVertices.Add(indexOfVertex);

StartProgram.AllAnts[j].RouteLength += MainParams.Distances[StartProgram.AllAnts[j].CurrentVertex, indexOfVertex];

StartProgram.AllAnts[j].CurrentVertex = indexOfVertex;

flag = false;

}

}

PreUpdatePheromones(j);

}

for (int a = 0; a < StartProgram.AllAnts.Length; a++)

{

var currRouteLength = StartProgram.AllAnts[a].RouteLength;

if (currRouteLength < answers[i].RouteLength)

{

answers[i].RouteLength = currRouteLength;

answers[i].ShortestRoute = StartProgram.AllAnts[a].VisitedVertices;

}

}

UpdatePheromones();

PrintResult(answers[i], i);

//Console.WriteLine( "\n\n");

//PrintHormoneConcentration();

}

}

private static double CountWayProbability(Ant ant, int startVertex, int endVertex)

{

var probability = (Math.Pow(MainParams.HormoneConcentration[startVertex, endVertex], MainParams.ALPHA)) \* (Math.Pow(MainParams.Visibility[startVertex, endVertex], MainParams.BETA));

double sum = 0;

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)//cycle of bypassing vertices

{

if (!ant.VisitedVertices.Contains(i))

{

sum += (Math.Pow(MainParams.HormoneConcentration[startVertex, i], MainParams.ALPHA)) \* (Math.Pow(MainParams.Visibility[startVertex, i], MainParams.BETA));

}

}

probability = probability / sum;

return probability;

}

private static void PreUpdatePheromones(int indexOfAnt)

{

StartProgram.AllAnts[indexOfAnt].RouteLength += MainParams.Distances[StartProgram.AllAnts[indexOfAnt].CurrentVertex, StartProgram.AllAnts[indexOfAnt].InitialVertex];

for (int p = 0; p < StartProgram.AllAnts[indexOfAnt].VisitedVertices.Count - 1; p++)

{

if (StartProgram.AllAnts[indexOfAnt].IsElite)

{

MainParams.UpdatedHormoneConcentration[p, p + 1] = (MainParams.ShortestRouteLength / StartProgram.AllAnts[indexOfAnt].RouteLength) \* 2;

}

else

{

MainParams.UpdatedHormoneConcentration[p, p + 1] = MainParams.ShortestRouteLength / StartProgram.AllAnts[indexOfAnt].RouteLength;

}

}

MainParams.UpdatedHormoneConcentration[StartProgram.AllAnts[indexOfAnt].CurrentVertex, StartProgram.AllAnts[indexOfAnt].InitialVertex] = MainParams.ShortestRouteLength / StartProgram.AllAnts[indexOfAnt].RouteLength;

}

private static void UpdatePheromones()

{

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

{

for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

{

var temp = MainParams.HormoneConcentration[i, j];

temp = temp \* (1 - MainParams.RHO) + MainParams.UpdatedHormoneConcentration[i,j];

MainParams.HormoneConcentration[i, j] = temp;

}

}

}

private static void PrintResult(Answer answer, int iteration)

{

Console.WriteLine("The shortest route in " + iteration + " iteration: " + answer.RouteLength);

Console.WriteLine();

Console.WriteLine("The route consists of: ");

for (int p = 0; p < answer.ShortestRoute.Count; p++)

{

if (answer.ShortestRoute.Count - 1 == p)

{

Console.Write(answer.ShortestRoute[p]);

}

else

{

Console.Write(answer.ShortestRoute[p] + " -> ");

}

}

Console.WriteLine("\n\n");

}

private static void PrintHormoneConcentration()

{

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

{

for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

{

Console.Write(MainParams.HormoneConcentration[i, j] + " ");

}

Console.WriteLine();

Console.WriteLine();

}

}

}

}

MainParams.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba3

{

public static class MainParams

{

public const short ALPHA = 3;

public const short BETA = 2;

public const float RHO = 0.7f;

public const int ANTS = 45;

public const int ELITE\_ANTS = 10;

public const int AMOUNT\_OF\_VERTICES = 200;

public const int MIN\_DISTANCE = 1;

public const int MAX\_DISTANCE = 40;

public static int[,] Distances;

public static double[,] Visibility;

public static double[,] HormoneConcentration;

public static double[,] UpdatedHormoneConcentration;

public static float[] InitHormoneConcentrationOptions = new float[3] { 0.1f, 0.2f, 0.3f };

public static int ShortestRouteLength = 0;

public static int RouteLength = int.MaxValue;

public static List<int> ShortestRoute = new List<int>();

public const int COLONY\_LIFETIME = 500;

}

public class Answer

{

public List<int> ShortestRoute = new List<int>();

public double RouteLength = int.MaxValue;

}

}

StartProgram.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba3

{

public class StartProgram

{

public static Ant[] AllAnts;

public static void Init()

{

RandomizeDistances();

GreedyAlgorithm(MainParams.Distances);

SetupVisibility();

SetupHormoneConcentration();

//PlaceAnts();

}

private static void RandomizeDistances()

{

MainParams.Distances = new int[MainParams.AMOUNT\_OF\_VERTICES, MainParams.AMOUNT\_OF\_VERTICES];

Random rnd = new Random();

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

{

for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

{

if(i == j)

{

MainParams.Distances[i, j] = -1;

}

else

{

if (MainParams.Distances[i,j] == 0)

{

MainParams.Distances[i, j] = rnd.Next(MainParams.MIN\_DISTANCE, MainParams.MAX\_DISTANCE + 1);

MainParams.Distances[j, i] = MainParams.Distances[i, j];

}

}

}

}

}

private static void GreedyAlgorithm(int[,] distances)

{

int sum = 0;

int counter = 0;

int j = 0, i = 0;

int min = int.MaxValue;

List<int> visitedRouteList = new List<int>();

// Starting from the 0th indexed

// city i.e., the first city

visitedRouteList.Add(0);

int[] route = new int[distances.Length];

// Traverse the adjacency

// matrix tsp[,]

while (i < distances.GetLength(0) &&

j < distances.GetLength(1))

{

// Corner of the Matrix

if (counter >= distances.GetLength(0) - 1)

{

break;

}

// If this path is unvisited then

// and if the cost is less then

// update the cost

if (j != i &&

!(visitedRouteList.Contains(j)))

{

if (distances[i, j] < min)

{

min = distances[i, j];

route[counter] = j + 1;

}

}

j++;

// Check all paths from the

// ith indexed city

if (j == distances.GetLength(0))

{

sum += min;

min = int.MaxValue;

visitedRouteList.Add(route[counter] - 1);

j = 0;

i = route[counter] - 1;

counter++;

}

}

// Update the ending city in array

// from city which was last visited

i = route[counter - 1] - 1;

for (j = 0; j < distances.GetLength(0); j++)

{

if ((i != j) && distances[i, j] < min)

{

min = distances[i, j];

route[counter] = j + 1;

}

}

sum += min;//distances[visitedRouteList[visitedRouteList.Count - 1], 0];

MainParams.ShortestRouteLength = sum;

}

private static void SetupVisibility()

{

MainParams.Visibility = new double[MainParams.AMOUNT\_OF\_VERTICES, MainParams.AMOUNT\_OF\_VERTICES];

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

{

for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

{

if(i == j)

{

MainParams.Visibility[i, j] = 0;

}

else

{

MainParams.Visibility[i, j] = 1 / (double)MainParams.Distances[i, j];

}

}

}

}

private static void SetupHormoneConcentration()

{

MainParams.HormoneConcentration = new double[MainParams.AMOUNT\_OF\_VERTICES, MainParams.AMOUNT\_OF\_VERTICES];

for (int i = 0; i < MainParams.AMOUNT\_OF\_VERTICES; i++)

{

for (int j = 0; j < MainParams.AMOUNT\_OF\_VERTICES; j++)

{

if (i == j)

{

MainParams.HormoneConcentration[i, j] = 0;

}

else

{

Random rnd = new Random();

var index = rnd.Next(MainParams.InitHormoneConcentrationOptions.Length);

MainParams.HormoneConcentration[i, j] = MainParams.InitHormoneConcentrationOptions[index];

//MainParams.HormoneConcentration[i, j] = 0.01;

}

}

}

}

public static void PlaceAnts()

{

AllAnts = new Ant[MainParams.ANTS];

var rndNumbers = new List<int>();

Random rnd = new Random();

for (int i = 0; i < AllAnts.Length; i++)

{

var vertex = rnd.Next(0, MainParams.AMOUNT\_OF\_VERTICES);

while(rndNumbers.Contains(vertex))

{

vertex = rnd.Next(0, MainParams.AMOUNT\_OF\_VERTICES);

}

AllAnts[i] = new Ant();

AllAnts[i].CurrentVertex = vertex;

AllAnts[i].InitialVertex = vertex;

AllAnts[i].VisitedVertices.Add(vertex);

if(i < MainParams.ELITE\_ANTS)

{

AllAnts[i].IsElite = true;

}

}

}

}

}

Ant.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Laba3

{

public class Ant

{

public int CurrentVertex = 0;

public int InitialVertex = 0;

public bool IsElite = false;

public List<int> VisitedVertices = new List<int>();

public double RouteLength = 0;

}

}

### Приклади роботи

На рисунках 3.1.1 і 3.1.2 показані приклади роботи програми.

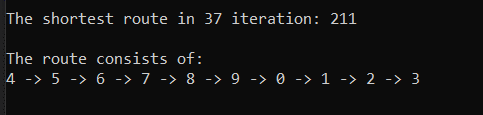


Рисунок 3.1.1

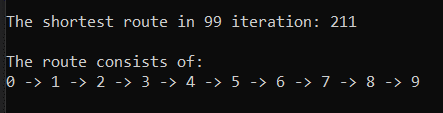


Рисунок 3.1.2

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

|  |  |
| --- | --- |
| **Кількість ітерацій** | **Значення цільової функції** |
| 5 | 4045 |
| 10 | 4104 |
| 20 | 4153 |
| 50 | 4183 |

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

### Графіки залежності розв'язку від числа ітерацій

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

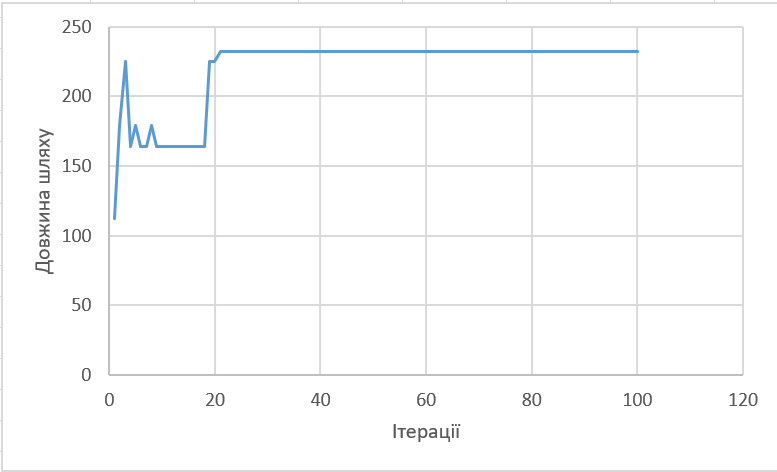


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи я ознайомився з основними підходами формалізації метаеврестичних алгоритмів і застосував один з них для вирішення типової задачі на практиці.

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%.