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: Григоренко Родіон Ярославович

Перевірив *Сопов О. О.*

(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

Варіант 5

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

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
1	Задача про рюкзак (місткість P=500, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб сумарна вага не перевищувала задану, а сумарна цінність була максимальною. Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.
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.

3 Розфарбовування графа (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число.

Застосування:

- розкладу для освітніх установ;
- розкладу в спорті;
- планування зустрічей, зборів, інтерв'ю;
- розклади транспорту, в тому числі - авіатранспорту;
- розкладу для комунальних служб;

4 Задача вершинного покриття (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа G = (V, E) - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S.

Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).

На вході: Граф G = (V, E).

Результат: множина $C \subseteq V$ - найменше вершинне покриття графа G.

Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;

- проектування інтегральних схем і конвеєрних ліній.

5 Задача про кліку (300 вершин, степінь вершини не більше 30, але не менше 2). Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній.

Задача про кліку існує у двох варіантах: у задачі розпізнавання потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в обчислювальному варіанті потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).

Застосування:

- біоінформатика;
- електротехніка;

6 Задача про найкоротший шлях (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях.

Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№ Алгоритми і досліджувані параметри

1 Генетичний алгоритм:

- оператор схрещування (мінімум 3);
- мутація (мінімум 2);
- оператор локального покращення (мінімум 2).

2 Мурашиний алгоритм:

- α;
- β;
- ρ;
- Lmin;
- кількість мурах M і їх типи (елітні, тощо…);
- маршрути з однієї чи різних вершин.

3 Бджолиний алгоритм:

- кількість ділянок;
- кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм

25 Задача комівояжера (асиметрична мережа) + Генетичний алгоритм

26 Задача комівояжера (симетрична мережа) + Генетичний алгоритм

27 Задача комівояжера (змішана мережа) + Генетичний алгоритм

28 Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм

29 Задача комівояжера (симетрична мережа) + Мурашиний алгоритм

30 Задача комівояжера (змішана мережа) + Мурашиний алгоритм

ВИКОНАННЯ

Псевдокод алгоритмів

-Генетичний алгоритм

```
public double GeneticAlgorithm()
ПОЧАТОК
    Random rnd = new Random();
    ПОКИ (int i = 0; i < iterations; i++)
        int a = rnd.Next(0, Population.Count);
        int b = rnd.Next(0, Population.Count);
        ЯКЩО (a == b && b < SizeOfPopulation - 2)
            b++;
        CrossBreeding(Population[a], Population[b]);
        NaturalSelection();
    return FindBestChromosome().Length;
```

КІНЕЦЬ

Вихідний код

Program.cs

```
using System;

namespace PALab5
{
    class Program
    {
        static void Main(string[] args)
        {
            Algorithm algorithm = new Algorithm();
            algorithm.Optimisation();
        }
    }
}
```

Chromosome.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PALab5
{
    class Chromosome
    {
        public Node[] Path = new Node[Algorithm.AmountOfCities];
        public double Length => GetLength();


        public Chromosome(Node[] path)
        {
            for (int i = 0; i < path.Length; i++)
            {
                Path[i] = path[i];
            }
        }
        public double GetLength()
        {
            double length = 0;
            for (int i = 0; i < Path.Length - 1; i++)
            {
```

```
            length += Path[i].Distances[Path[i + 1].Key];
        }

        length += Path[Path.Length - 1].Distances[Path[0].Key];

        return length;
    }

  }
}
```

Node.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PALab5
{
  class Node
  {
    public int Key;
    public double[] Distances = new
double[Algorithm.AmountOfCities];
```

```
        public double[] Pheromones = new
double[Algorithm.AmountOfCities];
    }
}
```

Algorithm.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PALab5
{
    class Algorithm
    {

        public List<Chromosome> Population = new
List<Chromosome>();
        public static int AmountOfCities = 150;
        private Node[] Nodes;
        public int SizeOfPopulation = 10;
        public double MutationProbability = 5;
        public int iterations = 1000;
```

```
Chromosome chromosome1;
Chromosome chromosome2;
Chromosome chromosome3;
public void InitMap()
{
    Nodes = new Node[AmountOfCities];
    Random rnd = new Random();
    for (int i = 0; i < AmountOfCities; i++)
    {
        Nodes[i] = new Node();
        Nodes[i].Key = i;
    }
    for (int i = 0; i < AmountOfCities; i++)
    {
        for (int j = 0; j < AmountOfCities; j++)
        {
            Nodes[i].Pheromones[j] = 0.5;
            if (Nodes[i].Key == j)
            {
                Nodes[i].Distances[j] = 0;
            }
            else if (Nodes[j].Distances[i] == 0)
            {
                double num = rnd.Next(5, 50);
                Nodes[i].Distances[j] = num;
```

```
                Nodes[j].Distances[i] = num;
            }
            else if (Nodes[j].Distances[i] != 0)
            {
                Nodes[i].Distances[j] = Nodes[j].Distances[i];
            }


        }
    }
}

public void InitPopulation()
{
    Population.Clear();
    Random rnd = new Random();
    for (int i = 0; i < SizeOfPopulation; i++)
    {
        List<Node> list = new List<Node>();
        foreach (var node in Nodes)
        {
            list.Add(node);
        }
        Node[] initialPath = new Node[AmountOfCities];
        for (int j = 0; j < AmountOfCities; j++)
```

```
            {
                initialPath[j] = list[rnd.Next(0, list.Count)];
                list.Remove(initialPath[j]);
            }
            Chromosome chromosome = new
Chromosome(initialPath);
            if (Repeats(chromosome) > 1) Console.WriteLine("WTF");
            //DisplayChromosome(chromosome);
            Population.Add(chromosome);
        }
    }

    public int Repeats(Chromosome chromosome)
    {
        int reps = 0;
        for (int i = 0; i < AmountOfCities; i++)
        {
            int tempreps = 0;
            for (int j = 0; j < AmountOfCities; j++)
            {
                if (chromosome.Path[i] == chromosome.Path[j])
                {
                    tempreps++;
                }
            }
```

```
            if (tempreps > reps)
            {
                reps = tempreps;
            }
        }
        return reps;
    }

    public void CrossBreeding(Chromosome parent1, Chromosome
parent2)
    {
        Random rnd = new Random();
        int border = rnd.Next(1, AmountOfCities - 1);

        Chromosome child1 = new Chromosome(parent1.Path);
        Chromosome child2 = new Chromosome(parent2.Path);

        //if (Repeats(parent1) > 1 || Repeats(parent2) > 1)
Console.WriteLine("AAAAAAAAAAAAAAAAGH");

        int reps1 = Repeats(child1);
        int reps2 = Repeats(child2);

        //Console.WriteLine(reps1.ToString() + "   " +
reps2.ToString());

        int counter = border;
```

```
chromosome1 = new Chromosome(parent1.Path);
chromosome2 = new Chromosome(parent2.Path);

for (int i = border; i < AmountOfCities; i++)
{

   if (!Contains(parent2.Path[i], child1.Path[0..border]))
   {
      child1.Path[counter] = parent2.Path[i];
      counter++;
      if (counter == AmountOfCities)
      {
         // Console.WriteLine("SHIT");
      }
   }
}
if (Repeats(child1) > 1)
{
}
//chromosome2 = new Chromosome(child1.Path);

foreach (var node in parent1.Path)
{
   if (!Contains(node, child1.Path))
   {
```

```
            child1.Path[counter] = node;
            counter++;
        }
    }

    chromosome3 = child1;



    if (Repeats(child1) > 1)
    {
        //Console.WriteLine("WHY");
    }

    counter = border;

    for (int i = border; i < AmountOfCities; i++)
    {

        if (!Contains(parent1.Path[i], child2.Path[0..border]))
        {
            child2.Path[counter] = parent1.Path[i];
            counter++;
        }
    }
    if (Repeats(child2) > 1)
```

```
{

}

foreach (var node in parent2.Path)
{
    if (!Contains(node, child2.Path))
    {
        child2.Path[counter] = node;
        counter++;
    }
}

chromosome3 = child2;

if (Repeats(child2) > 1)
{

}

reps1 = Repeats(child1);
reps2 = Repeats(child2);

child1 = Mutation(child1);
child2 = Mutation(child2);
```

```
            Population.Add(child1);
            Population.Add(child2);

        }


        public Chromosome Mutation(Chromosome chromosome)
        {

            Random rnd = new Random();
            if (rnd.Next(0, 100) < MutationProbability)
            {
                Chromosome mutated = new
Chromosome(chromosome.Path);
                int a = rnd.Next(0, AmountOfCities);
                int b = rnd.Next(0, AmountOfCities);
                Node temp = mutated.Path[a];
                mutated.Path[a] = mutated.Path[b];
                mutated.Path[b] = temp;
                if (mutated.Length < chromosome.Length)
                {
                    return mutated;
                }
            }
            return chromosome;
        }
```

```
public bool Contains(Node targetNode, Node[] nodes)
{
    foreach (var node in nodes)
    {
        if (targetNode == node)
        {
            return true;
        }
    }
    return false;
}

public void NaturalSelection()
{
    for (int i = 0; i < Population.Count - SizeOfPopulation; i++)
    {
        Chromosome chromosome = Population[0];
        for (int j = 1; j < Population.Count; j++)
        {
            if (Population[i].Length > chromosome.Length)
            {
                chromosome = Population[j];
            }
        }
        Population.Remove(chromosome);
```

```
    }

}


public Chromosome FindBestChromosome()
{
    Chromosome chromosome = Population[0];
    for (int i = 1; i < Population.Count; i++)
    {
        if (chromosome.Length > Population[i].Length)
        {
            chromosome = Population[i];
        }
    }
    return chromosome;
}

public void DisplayChromosome(Chromosome chromosome)
{
    foreach (var node in chromosome.Path)
    {
        Console.Write(node.Key.ToString() + " - ");
    }
    Console.WriteLine("\n\n");
}
```

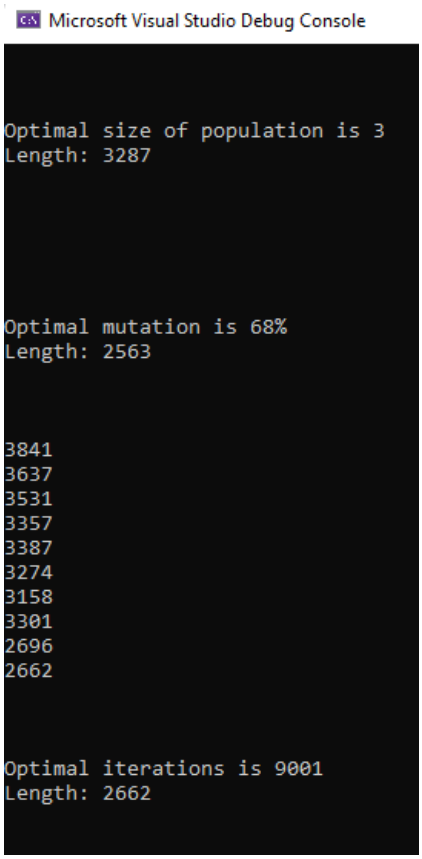
```
public double GeneticAlgorithm()
{
    Random rnd = new Random();
    for (int i = 0; i < iterations; i++)
    {
        int a = rnd.Next(0, Population.Count);
        int b = rnd.Next(0, Population.Count);
        if (a == b && b < SizeOfPopulation - 2) b++;
        CrossBreeding(Population[a], Population[b]);
        NaturalSelection();
    }

    /*DisplayChromosome(FindBestChromosome());
    Console.WriteLine("\n\n");
    /*Console.WriteLine(FindBestChromosome().Path.Length);
    Console.WriteLine("\n\n");
    Console.WriteLine(Repeats(FindBestChromosome()));
    Console.WriteLine(FindBestChromosome().Length);*/

    return FindBestChromosome().Length;
}

public void Optimisation()
{
    InitMap();
```

```
        double optimalSizeOfPopulation = 2;
        double optimalSizeOfPopulationLength = 100000;
        for (int i = 2; i < 100; i++)
        {
            SizeOfPopulation = i;
            InitPopulation();
            double l = GeneticAlgorithm();
            if (l < optimalSizeOfPopulationLength)
            {
                optimalSizeOfPopulationLength = l;
                optimalSizeOfPopulation = SizeOfPopulation;
            }

        }

        Console.WriteLine("\n\n");
        Console.WriteLine("Optimal size of population is " +
optimalSizeOfPopulation.ToString() + "\nLength: " +
optimalSizeOfPopulationLength.ToString());
        Console.WriteLine("\n\n");
        SizeOfPopulation = 10;
        InitPopulation();

        double optimalMutation = 0;
        double optimalMutationLength = 100000;
        for (int i = 0; i < 100; i++)
```

```
        {
            InitPopulation();
            MutationProbability = i;
            double l = GeneticAlgorithm();
            if (l < optimalMutationLength)
            {
                optimalMutationLength = l;
                optimalMutation = MutationProbability;
            }

        }

        Console.WriteLine("\n\n");
        Console.WriteLine("Optimal mutation is " +
optimalMutation.ToString() + "%\n" + "Length: " +
optimalMutationLength.ToString());
        Console.WriteLine("\n\n");
        MutationProbability = 5;

        double optimalIterations = 0;
        double optimalIterationsLength = 100000;
        for (int i = 1; i < 10000; i+= 1000)
        {
            InitPopulation();
            iterations = i;
            double l = GeneticAlgorithm();
```

```
            Console.WriteLine(l);
            if (l < optimalIterationsLength)
            {
                optimalIterationsLength = l;
                optimalIterations = iterations;
            }

        }

        Console.WriteLine("\n\n");
        Console.WriteLine("Optimal iterations is " +
optimalIterations.ToString() + "\nLength: " +
optimalIterationsLength.ToString());
        Console.WriteLine("\n\n");
        iterations = 1000;

       }


   }
}
```

Приклади роботи



Тестування алгоритму

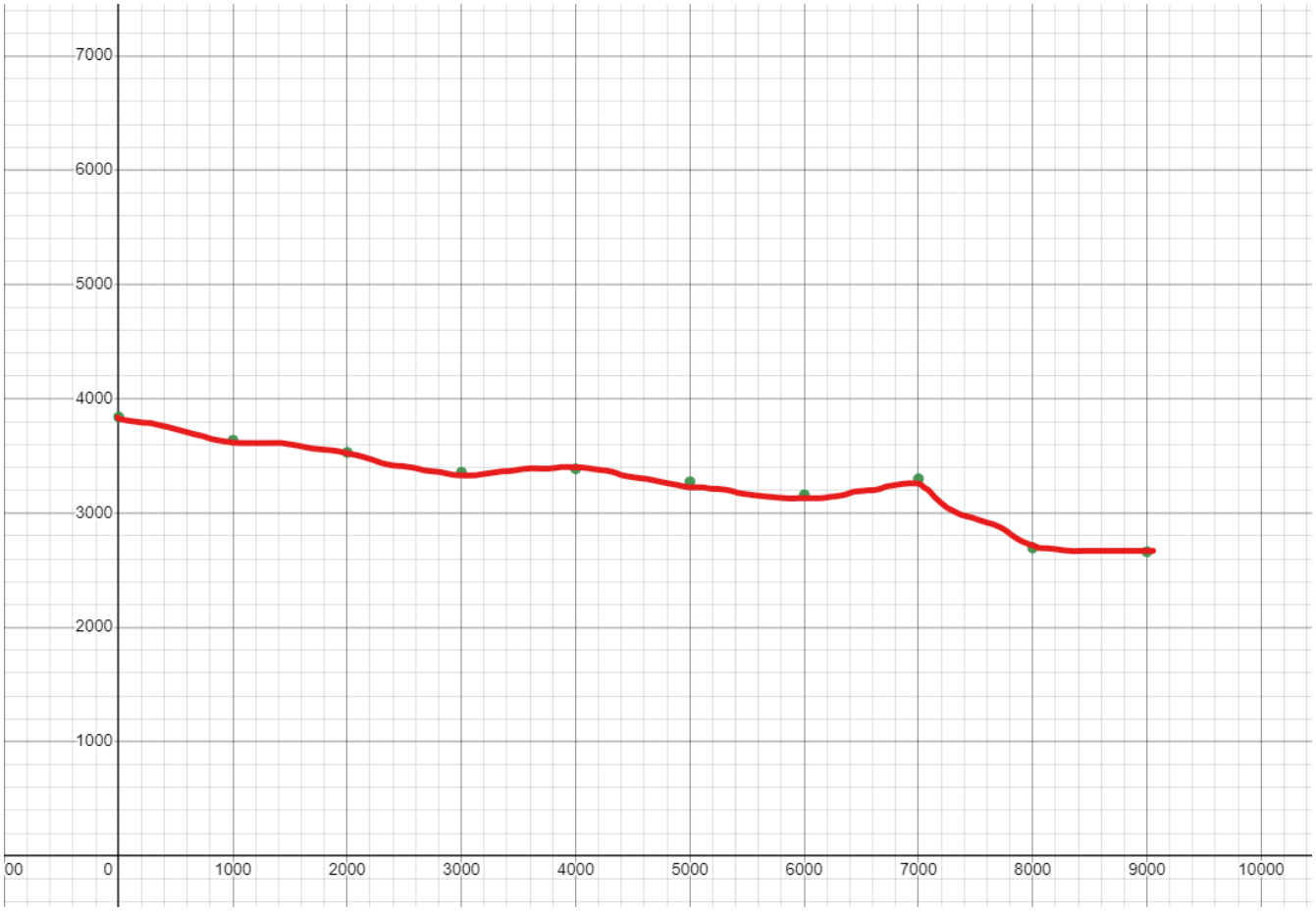
У таблиці наведена довжина шляху на кінцевій ітерації в залежності від кількості ітерацій

Кількість ітерацій	Довжина шляху
1	3841
1001	3637
2001	3531
3001	3357
4001	3387
5001	3274
6001	3158
7001	3301
8001	2696
9001	2662

Як бачимо чим більша кількість ітерацій тим краще результат роботи алгоритма.

Графіки залежності розв'язку від числа ітерацій

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



ВИСНОВОК

В рамках даної лабораторної роботи дослідив і навчився працювати з алгоритмами для вирішення NP-складних задач. Зокрема написав програмну реалізацію генетичного алгоритму для задачі комівояжера та його оптимізацію.