BILKENT UNIVERSITY

IE 400 Principles of Engineering Management

Spring 2017-2018

Project Assignment

Group Members:

Özgür Can Erdoğan-21300586

Barış Gündoğan-21300797

Arif Can Terzioğlu-21302061

Instructor: Prof. Oya Karaşan

Submission Date: May 3, 2018

Introduction

In our project, nearest neighbor algorithm was chosen as heuristic to solve four different type of TSP problems: Euclidean, Non-Euclidean, Symmetric and Asymmetric. The resultant paths, total travel times, errors of TSP problems, discussions accordingly and conclusion can be found below:

Also, Euclidian and Non-Euclidian samples have been generated on Matlab. Symmetric and Non-symmetric samples have been produced on C# environment. Nearest neighbor algorithm has been also implemented on C# environment.

Euclidean Solution:

Optimal Solution with Heuristic:

1-36-22-5-24-8-11-15-29-35-20-4-9-23-31-6-14-37-18-12-21-28-39-38-30-40-2-17-34-25-33-27-32-26-3-16-19-10-7-13-1

Total Travel Time= 586.794

Optimal Solution with Given Model:

1-22-5-24-8-11-15-29-35-20-4-9-23-14-6-31-37-18-12-16-21-39-28-10-7-19-13-38-30-40-34-2-17-25-33-32-27-26-3-36-1

Total Travel Time= 485.741

Error: $\left(\frac{\text{Original Model Solution} - \text{Heuristic Solution}}{\text{Original Model Solution}}\right) * 100$ = 17.2 %

Non-Euclidean Solution:

Optimal Solution with Heuristic:

1-36-3-26-33-32-27-2-17-25-34-40-38-39-28-19-10-7-21-16-37-12-31-6-14-15-29-35-20-4-9-23-19-13-30-22-5-24-8-11-1

Total cost= 942.973

Optimal Solution with Given Model:

1-36-3-26-27-32-33-25-30-38-34-40-2-17-39-13-7-10-19-12-37-18-21-28-16-31-14-6-23-9-4-20-35-29-15-11-8-24-5-22-1

Total cost= 773.115

Error: $\left(\frac{\text{Original Model Solution} - \text{Heuristic Solution}}{\text{Original Model Solution}}\right) * 100$ = 18.0 %

Discussion/Conclusion: According to results, we had 17.2 percent error with Euclidean instances and 18.0 percent error with Non-Euclidean instances. Even if the difference is relatively small, we conclude that nearest neighbor heuristic algorithm works better for TSP problem with Euclidean instances over Non-Euclidean instances with slight difference.

However, it may be related to the instances that were generated randomly, and still with different generations, we expect the same results in terms of functionality.

Symmetric Solution:

Optimal Solution with Heuristic:

1-19-24-38-5-13-36-34-39-8-37-7-18-32-20-22-15-21-10-17-4-27-11-14-29-25-26-35-28-2-40-6-30-16-33-3-31-23-9-12-1

Total cost= 344

Optimal Solution with Given Model:

1-38-37-8-9-18-7-20-32-24-31-35-28-29-25-39-34-11-14-12-17-10-21-15-22-6-40-5-13-36-30-16-33-3-26-23-27-4-2-19-1

Total cost= 208

Error: $\left(\frac{\text{Original Model Solution}-\text{Heuristic Solution}}{\text{Original Model Solution}}\right) * 100$ =39.5 %

Asymmetric Solution:

Optimal Solution with Heuristic:

1-26-8-25-6-9-12-29-21-5-3-13-34-19-32-10-2-36-39-20-16-40-22-38-15-31-14-30-11-28-37-27-33-35-4-23-24-18-7-17-1

Total cost= 400

Optimal Solution with Given Model:

1-26-2-13-34-29-25-33-28-21-5-18-7-35-8-22-11-20-16-40-24-17-6-31-14-30-39-4-9-12-37-27-32-10-19-23-3-15-36-38-1

Total cost=141

Error: $\left(\frac{\text{Original Model Solution}-\text{Heuristic Solution}}{\text{Original Model Solution}}\right) * 100$ = 64.8 %

Discussion/Conclusion: According to results, 39.5 percent error with Symmetric instances and 64.8 percent error with Asymmetric instances have gained. Error is very high for both of cases where the nearest neighbor heuristic's functionality is questionable. On the other hand, if we compare two different case of instances, it is clearly concluded that with Symmetric instances nearest neighbor heuristic algorithm works better over Asymmetric instances.

Best Solution

With randomly generated instances and the data collected, as a result of solution four type of TSP problems, Euclidean TSP problem were best solved with the nearest neighbor heuristic that we have used in this project.

Capability of Nearest Neighbor Algorithm

Some inconsistencies on the graphs can be observed such as symmetric and non-symmetric graph as the number of nodes are increasing. This situation might occur since we have new possibilities of new better weighted edges that leads to lower cost indirectly as we are add new nodes.

For example, let's assume that we have a resulting sub path that is A->B->C with total cost 20. Let's add one new node 'D' to graph. Cost of going to 'D' from A may be smaller than cost of going to 'B' from 'A'. As a result, we may get lower the total cost 20.

Consider a graph in a form of

```
 A
/|\
B-+-C
\|/
 D
```

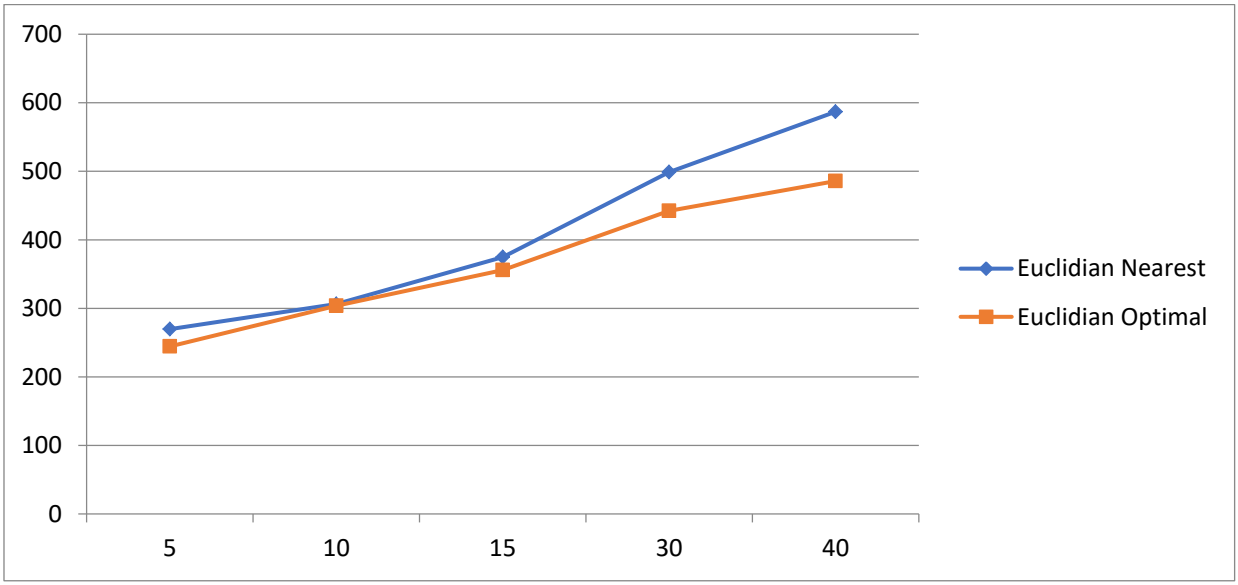
Say length of B-C is 10, length of A-D is 24 and thus length of A-B is 13. The optimal route is A-B-D-C-A, 52 units long. Algorithm would produce the path A-B-C-D-A, 60 units long.

As it is expected, nearest neighbour algorithm does not give the optimal solution all the time. In general case, as number of nodes is increasing, heuristic results are getting away from optimal results. Algorithm has a higher success rate at number of nodes that are 5 and 10 most of the case. Exact match with optimal solution has been observed only at non-Euclidian and non-symmetric case where number of nodes is 5.

Differences in symmetric and non-symmetric are higher compared to other two. It may be because there are many specially arranged city distributions which make the nearest neighbour algorithm give the worst route [1]. This may hold for both asymmetric and symmetric TSP.

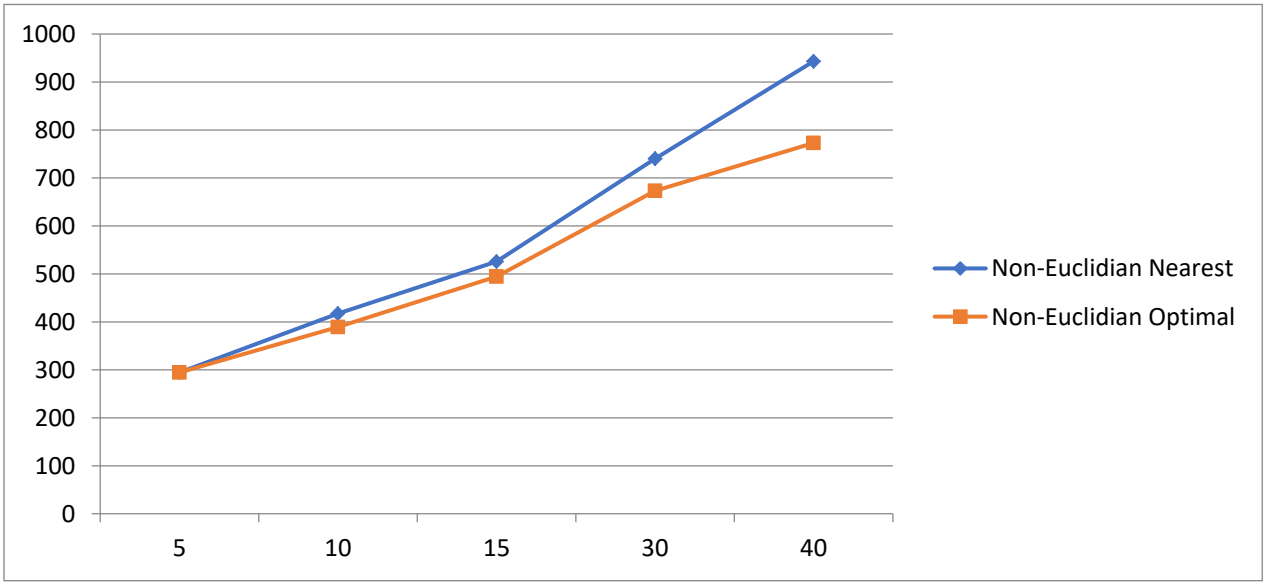
Results based on input sets can be seen below.

Euclidean Sample



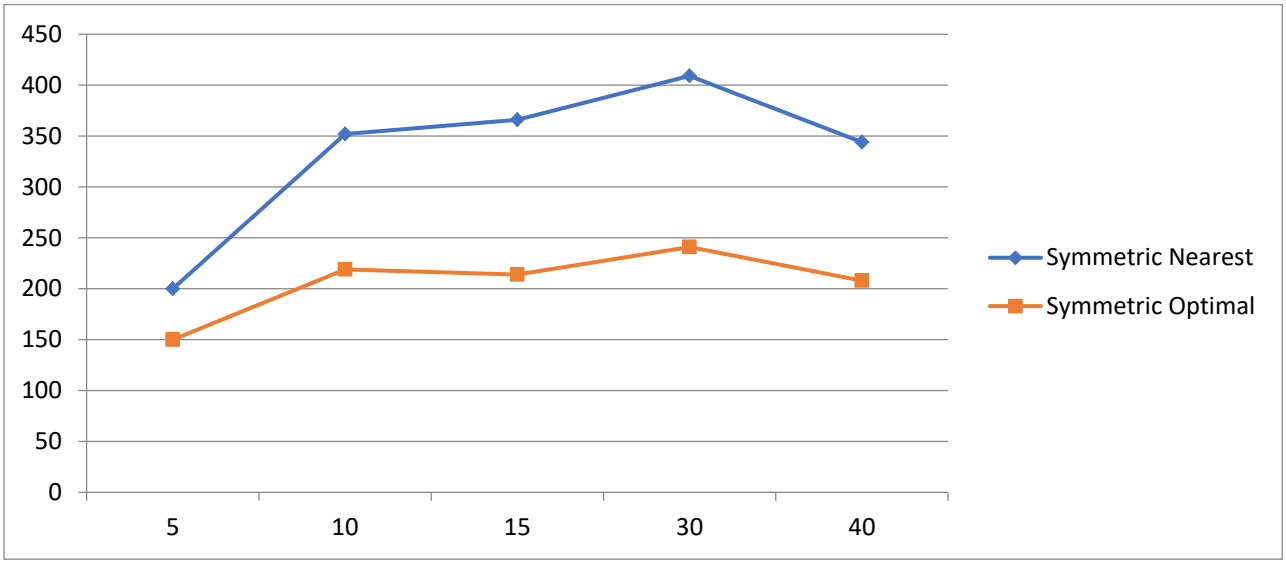
	Nearest	Optimal
5	269.704	244.44
10	306.3487	303.788
15	374.83047	355.967
30	498.866957	442.212
40	586.794	485.741

Non- Euclidian Sample



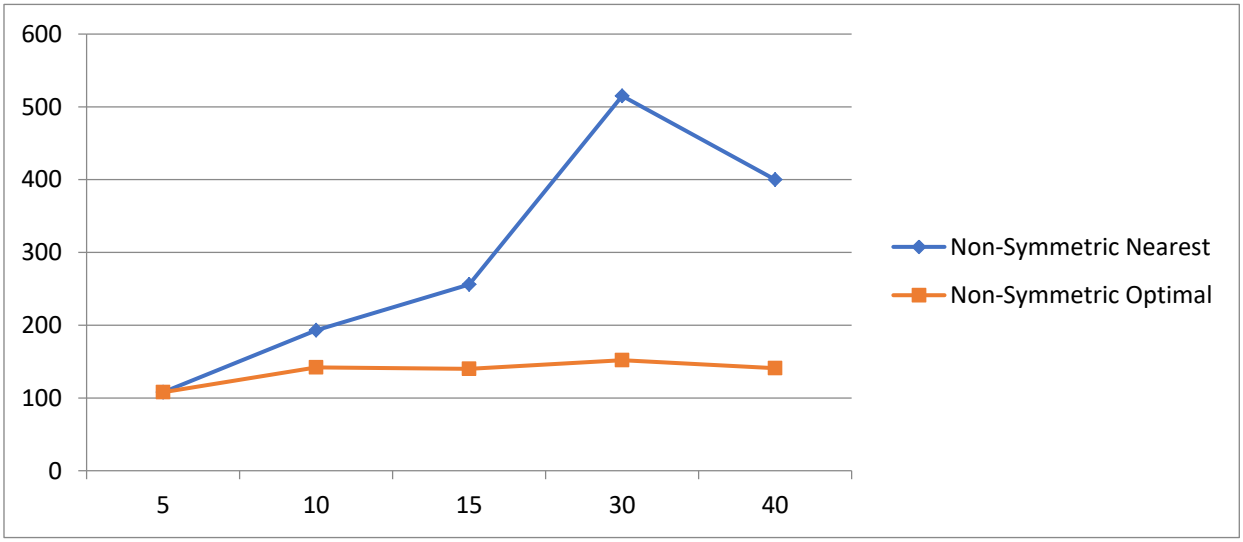
	Nearest	Optimal
5	294.67	294.67
10	417.2559	389.408
15	525.825	494.265
30	740.29076	673.264
40	942.97318	773.115

Symmetric Sample



	Nearest	Optimal
5	200	150
10	352	219
15	366	214
30	409	241
40	344	208

Non-Symmetric Sample



	Nearest	Optimal
5	108	108
10	193	142
15	256	140
30	515	152
40	400	141

Appendix A

Euclidian and Non-Euclidian Random Sample Generator

```
clc
clear

x=rand(1,40)*100;
y=rand(1,40)*100;
for i=1:40
    x(i)=floor(x(i));
end
for i=1:40
    y(i)=floor(y(i));
end
scatter(x,y)
z=[];
for i=1:length(x)
    for j=1:length(x)
            z(i,j)= sqrt(((x(i)-x(j))^2)+((y(i)-y(j))^2));
    end
end

w=[];
for i=1:length(x)
    for j=1:length(x)
            if (i==j)
                w(i,j)=z(i,j);
            else
                w(i,j)=z(i,j)+rand(1)*20
            end
    end
end
```

Appendix B

Symmetric and Non-Symmetric Random Sample Generator

```
using System;
using System.IO;

namespace ConsoleApp2
{
   class Program
   {
      static void Main(string[] args)
      {
         int[,] sample = new int[40,40];
         Random rnd = new Random();

         for(int i = 0; i<40;i++)
         {
            for(int j = 0; j<40;j++)
            {
               if(i!=j)
               {
                  int card = rnd.Next(100);
                  sample[i, j] = card;
                 // sample[j, i] = card;
               }
            }
         }

         form(sample);
      }

      public static void form(int [,] data)
      {

         using (StreamWriter outfile = new StreamWriter(@"C:\Users\Arif\Desktop\mycsv2.csv"))
         {
            for (int x = 0; x < 40; x++)
            {
               string content = "";
               for (int y = 0; y < 40; y++)
               {
                  content += data[x, y].ToString("0.00") + ";";
               }
               outfile.WriteLine(content);
            }
         }
      }

   }
}
```

Appendix C

Nearest Neighbour Algorithm

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

namespace ConsoleApp1
{
    class Program
    {
        static int numberOfNodes = 5;
        static double cost = 0;
        static void Main(string[] args)
        {

            Double[,] graph = new Double[numberOfNodes, numberOfNodes];

            //StreamReader streamreader = new
StreamReader(@"C:\Users\erdog\source\repos\ConsoleApp1\ConsoleApp1\Euclidian.txt");
            //char[] delimiter = new char[] { '\t' };
            using (StreamReader reader = new
StreamReader(@"C:\Users\Arif\Desktop\project\ConsoleApp1\Tab İnputs\symmetric.txt"))
            {
                int i = 0;
                int j = 0;
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    var delimiters = new char[] { '\t' };
                    var segments = line.Split(delimiters,
StringSplitOptions.RemoveEmptyEntries);

                    foreach (var segment in segments)
                    {

                        graph[i, j] = double.Parse(segment,
System.Globalization.CultureInfo.InvariantCulture);
                        j++;
                    }
                    i++;
                    j = 0;
                }
            }

            for(int i=0; i < numberOfNodes; i++)
            {
                for (int j = 0; j < numberOfNodes; j++)
                {
                    Console.Write(graph[i,j] + " ");
                }
                Console.Write("\n");
            }
            List<int> path = new List<int>();
            path.Add(0);
```

```
            EnYakınKomşuBaldanTatlıdır(graph, path);
            //cost += graph[path.Last(), 0];

            Console.WriteLine("Total Cost: " +cost);
            foreach(int node in path)
            {
                Console.Write(node+1 + "-");
            }

            Console.ReadLine();


        }
        public static void EnYakınKomşuBaldanTatlıdır (Double[,] graph, List<int>
path){

            Dictionary<int, Double> shortestOnes = new Dictionary<int, Double>();
            for (int i = 0; i < numberOfNodes; i++)
                shortestOnes.Add(i, 100000);

            int p = 0;
            do
            {
                int position = path.Last();
                int i = position;

                for (int j = 0; j < numberOfNodes; j++)
                {
                    if (graph[i, j] != 0 && shortestOnes[i] > graph[i, j] &&
!path.Contains(j))
                    {
                        shortestOnes[i] = graph[i, j];
                        position = j;
                        //graph[i, j] = 0;

                    }
                    //else
                    //{
                    //    graph[i, j] = 0;
                    //}
                }

                path.Add(position);
                cost += shortestOnes[i];
                //for (int k = 0; k < numberOfNodes; k++)
                //    graph[k, i] = 0;
                p++;

            }
            while (path.Last() != 0 && p<numberOfNodes-1);

            cost += graph[path.Last(), 0];
            path.Add(0);


        }
    }
}
```

References

[1]G. Gutin, A. Yeo and A. Zverovich, "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP", 2018. .