Evaluation of an Algorithm that Minimizes the Maximum Congestion Ratio of a Network in OSPF Routing

Submitted by

Shoieb Rahman

2011-1-60-014

Dept. Of Computer Science and Engineering

East West University

Supervised By

K. M. Imtiaz-Ud-Din

Senior Lecturer

Dept. Of Computer Science and Engineering

East West University

Declaration by Candidate

I hereby declare that the work presented in this project is, to the best of our knowledge and belief, original, except as acknowledged in the text and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Shoieb Rahman

_____

Letter of Acceptance

I hereby declare that this thesis is from the student's own work and effort, and all other sources of information used have been acknowledged. This thesis has been submitted with my approval.

SUPERVISOR: K. M. Imtiaz-Ud-Din

SIGNATURE: _____

DATE: _____

CHAIRPERSON: Dr Shamim H Ripon

SIGNATURE: _____

DATE: _____

ACKNOWLEDGEMENT

This project work would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, we would like to express our sincere gratitude to K. M. Imtiaz-Ud-Din Sir, my supervisor and mentor for this project work, for his valuable time helpingme to overcome all the obstacles.

Besides my supervisor, I would like to thank the rest of our thesis committee and all the members of Computer Science and Engineering Department for enormous support they have given me for four years.

Table of Contents

List of Table

List of Figure

Abstract

In Open Shortest Path First (OSPF) routing protocol, all the packets are transmitted over the shortest path. And these path is determined by the weight associated with each link in the network. According to OSPF the link with minimum weight will transmit most of the packet. For this, at a moment the link exceeds its maximum capacity, thus congestion occurs. This problem can be avoided by applying an optimal set of link weight. To solve this OSPF weight setting problem several iterative methods have been used.In this paper, we have evaluated an algorithm proposed by Kamrul and Oki [1] to solve this problem. And we found that the computational result leads us to minimize the maximum network congestion.

Chapter-1: INTRODUCTION

1.1 Introduction

Internet traffic is rapidly growing for the massive expansion of network [2]. The number of web user is increasing day by day due the huge number of web based application. People are becoming more dependent to the web service and if this keep going on, it is certain to get the network collapsed.

Here comes the management factor of the network. And managing the flows of the traffic can increase the service of the network. Between the various networking devices Router is the most commonly used device. It basically forwards the data from the source to destination. Different ways exist between a given source and end of the line pair, because of excess network between nodes. The basic activities of a router is to find the best path from the source to destination which is shortest path among them. And for this they maintain a table named routing table which is get updated after some certain time again and again which stores the shortest path by which the packet should be forwarded from source to destination.

The complexity of internet is increasing day by day, and the only reason for this is growing enormously. So it is separated into Autonomous Systems (AS) to reduce the complexity for this huge size of internet. An AS is nothing but a set of networks under the control of one single entity or organization with a common routing policy. For managing the routing inside of an AS there are Interior Gateway Protocols (IGP) and Exterior gateway Protocols (EGP) for managing routing in-between Ass [3]. Among the different Interior Gateway Protocol (IGP) OSPF is one of them, which uses the Dikjstra algorithm to find the shortest path. And the

performance measure parameter for this algorithm for fining the shortest path is the link weight. Where the sum of the OSPF weights on the links in the path gives the cost of that path. The path with the least cost is the shortest path. So, determination of link which is optimized means the determination of optimal routing based on shortest-path routing [1].

1.2 Problem statement

Congestion, in the context of networks, refers to a network state where a node or link carries so much data that it may decrease network service quality, resulting in queuing delay, frame or data packet loss and the blocking of new connections. In a congested network, response time slows with reduced network throughput.

This problem can be avoided in OSPF routing by setting an optimal link weight. Determining the optimal link weight means determining the optimal routing based on shortest path routing. This problem is found to be a part of NP-hard [4]. To solve this OSPF weight setting problem Tabu Search (TS) [5] algorithm is used by Kamrul and Oki [1]. Previously TS has been used by Fortz and Thorup [4] and they have used it to determine the cost and the maximum utilization by the cost function proposed by them.

1.3 Objectives

Our objective is toevaluate the proposed algorithm of Kamrul and Oki [1] to minimize the maximum congestion ratio of a network. And to determine whether the computational results indicated that the proposed algorithm could lead to avoid network congestion or not.

Chapter-2: LITERATURE REVIEW

2.1 Traffic Engineering

In order to optimize the resource utilization and increase the network performance traffic engineering sets the map for the traffic flows through the existing network topology. It helps to reduce the network congestion from the problem due to insufficient of resource allocation. In early 1990's, internet service providers mapped Internet traffic flows onto the physical network topology in an ad-hoc manner. The traffic routs followed the shortest path. Using the shortest paths attempts to conserve network resources, however since the transfer speed accessibility and movement attributes are not considered, this methodology regularly prompts over utilization.

2.2 Routing Protocols

A routing protocol indicates out how a packet is routed in a network from source to its destination. Its information is as per the following: here a directed graph is given, G = (V,E, c), which describes the system topology. In G, V represents all the nodes of the network, E is the situated of connections between the nodes, and c is a limit capacity matrix bends to whole numbers.

A few numerical amounts are connected with each one circular edge, nodes, capacity, load, utilization. The capacity, given as info, is a physical property of the edges. The load speaks to real traffic over a connection and is reliant on the routing protocol utilized and the activity request. Their dimensionless ratio is the utilization of the edge. The neighborhood cost at each one edge is a metric of optimality. It is a capacity of edge utilization and, perhaps, changes from edge to edge.

We expect that packets take after IP packets and have a time-to-live field (TTL) which details the remaining number of edges a parcel can cross before it is dropped. Every switch decrements the TTL field after sending it to a neighbor. In the event that the TTL lapses, the switch just drops the bundle.

The conventions we consider have the accompanying three-section structure.

i. Weights are assigned to arcs and changed infrequently (e.g., in response to long-term trends in network traffic and topology).
ii. Next, the routers periodically broadcast to each other the network's link state (which links are currently up) or when a link goes down. From the link state and the known weights, each router R computes and stores a local routing table that specifies, for each destination router t, to which neighbor(s) of R packets bound for t should be forwarded.
iii. Finally, when a packet arrives at R bound for t, R consults its routing table and the packet's TTL field and sends the packet to one of the active neighbors or drops the packet.

All protocols that we consider assume that positive weights have been assigned to the links of the network in some fashion. To avoid confusion we will use the terms hop count and depth and speak of shallow and deep paths when we talk about the unweighted graph. We use the terms weight and (less frequently) length to refer to the weighted graph, thus the lightest or shortest paths refer to paths of least total weight.

2.3 Overview of OSPF

In OSPF the routers have edge weights. Periodically the routers agree on the link state of the network via broadcast. Each router locally performs Dijkstra's shortest path algorithm; for

each destination, the router determines which of its neighbors lie along the lightest path to destination. This next-hop by destination information is stored in routing tables. Upon receiving a packet bound for destination, the router forwards the packet to the neighbor along the lightest path to destination. In case there is more than one neighbor with this property, the router splits the traffic bound for destination evenly among these next hops. This last property of OSPF is crucial—by this mechanism, OSPF can spread traffic along many edges to avoid congestion—and several heuristic weight-setting procedures try to exploit it. As specified, OSPF guarantees that packets reach their destinations with optimum latency. It does not show how to set weights or suggest an objective function to optimize. Fortz and Thorup have shown how to set up link costs and approximate a solution close to optimal for minimizing maximum link utilization; nevertheless, approximating the optimum weight setting within a constant factor, given the network topology, capacities, is an NP-hard problem [6].

2.4Tabu Search

Tabu search is an iterative procedure designed to solve optimization problems. It is based on selected concepts that untie the fields of artificial intelligence and optimization. It has been the applied to a large number of problems such as job scheduling, graph coloring, and networking planning. There are several studies that used Tabu search to find an optimal OSPF link weight set [3] [10] [11].

Tabu search is the guided investigation of the space of permissible solutions, and keeps a record for all arrangement assessed along the way. The investigation begins from a starting solution. At the point when a stop basis is fulfilled, the calculation gives back where it’s due gone by solution. To move starting with one solution then onto the next, Tabu search

investigates the area of the last solution went to. It produces a neighbor solution by applying a change, which is known as a move, on the current solution. The set of all permissible move particularly characterizes the area of the current solution. At every emphasis of the Tabu search calculation, all solution in the area are assessed and the best chosen as the new present solution.

2.5 Related Work

The attention of even traffic part in OSPF weight setting was first done by Fortz and Thorup [4]. They additionally settled that this issue is NP-hard. They at first proposed a nearby inquiry heuristic furthermore connected Tabu Search iterative heuristic [3] [6] to enhance the aftereffects of neighborhood pursuit. They tried their heuristics on At&t spine system and on engineered systems.

Ericsson et al. [7] proposed a Genetic Algorithm (GA) and utilized the set of test issues considered in [4]. A half and half GA was additionally proposed by them [9] which makes utilization of the element most limited way calculation to re-compute briefest ways after the modification of connection weights.

Sridharan et al. [8] created an alternate heuristic for a somewhat distinctive variant of the issue, in which the flow is part among a subset of the friendly connections on the most limited ways to the goal IP.

Chapter-3: METHODOLOGY

3.1 Definitions

G(V,E) represents the graph of the network where V is the set of links and *E* is the set of link. $V \in V$, where *v = 1, 2,……., N*, which represents an individual node and *e*, where *e = 1, 2, ……., L*, which represents bidirectional individual link. Here *L* represents the number of the link and *N* represents the number of nodes. C_e is the capacity of $e \in E$. The traffic $T = \{d_{pq}\}$ is the traffic matrix and *d* is the demand of traffic from the source *p* to destination *q*. The ration of congestion of the network r is indicates the maximum utilization ratios of all the network link. And r is defined by,

$$r = \max_{e \in E} \frac{u_e}{c_e} \quad \text{……. (1)}$$

Here, 0 <*r*< 1. The maximum value of the traffic $\frac{1-r}{r}\ d_{pq}$ which is added to the existing $d_{pq,}$ regarding any source and destination pair *p*, *q*. So that any the value of the traffic passing through any *e* doesn't cross C_e. The total traffic becomes $\frac{1}{r}\ d_{pq}$ after adding $\frac{1-r}{r}\ d_{pq}$ to the $d_{pq.}$ For that the updated congestion has become the upper limit $R^* = 1$. And now the maximization of the extra value of traffic $\frac{1-r}{r}\ d_{pq}$ is similar to minimizing [12].

$W = \{w_e\}$ is the link weight matrix of the network *G*, where w_e is the weight of link e. W_{cand} is the set of candidate W for which we are calculating the worst case congestion. *r(w)* is the function to which returns the congestion ration defined in Eq. (1) for *G* according to OSPF

based shortest path routing using the link weight in *W*. *R(w)* refers to the worst-case congestion ratio in *W*. *R(W)* is defined by,

$$R(w) = \max_{W \in W\,cand} r(w) \quad \text{..... (2)}$$

Now our target is to find the most appropriate set of link weight W^* for network *G* that minimizes *R(W)* defined in Eq. (2). W^* is defined by

$$W* ¿ \arg\min_{W \in W\,cand} R(w) ❑ \quad \text{..... (3)}$$

Where W_{cand} all possible link weight set candidates. The network congestion ratio is achieved by using W^* is *R(W)* that represents the upper bound of congestion in a network.

3.2 Algorithm

Step 1: A weight matrix (W) and Traffic matrix (T) is set by randomly generating value of each link in the network. At this stage this is our initial best solution (WM*).

Step 2: The maximum congestion is evaluated by the Eq. (1).

Step 3: If the maximum congestion ratio is less than the previous congestion ratio then we set this W as our best solution W*. Otherwise we go to our next step.

Step 4: Then we find out the maximum congested link in the network topology.

Step 5: Change the weight of the link by increasing 1. At least one route passing through this link is changed then which decreases the congestion of that link.

Step 6: Then again we go to step 2 until the stopping criterion is satisfied. And here our stopping criterion is a pre-determined iteration number I_{max}.

3.3Procedure of evaluation

At first we have taken 3 different network topologies as shown in Figure 1, Figure 2, and Figure3 having 6 nodes 10 bidirectional links, 6 nodes 11 bidirectional links and 12 nodes 18 bidirectional links respectively. We have used these network topologies to measure the performance of the algorithm. And here our performance measure is the congestion ratio. Initially the W_{cand} and T is a matrix generated by random value in the range of (1, 10). Link capacity C for each link in this given networks is a constant integer value set to 100 and the stopping criterion, maximum iteration number I_{max} is set to 10.

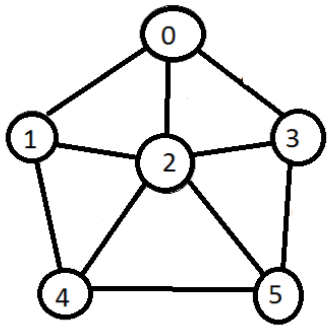

Figure 1: Network topology 1

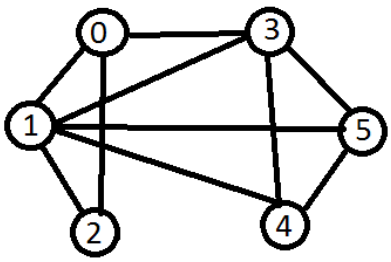

Figure 2: Network topology 2

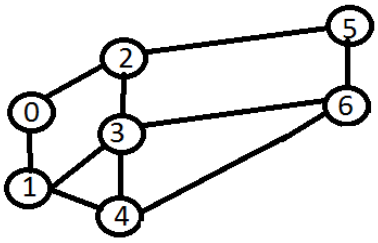

Figure 3: Network topology 3

We started our initial iteration with the randomly generated W_{cand} and T. And this is our initial best solution $W^*$and the congestion ratio R is the upper limit of R^*. Then we calculated the congestion ratio r using Eq. (1) for each link in the network. Using Eq. (2) we find out the maximum congestion ratio$R$ and mark the link. Then using Eq. (3) we find out the W*. After that increased the max congested link weight by adding 1. The pseudo code for this procedure is given below,

```
1:    Begin
2:    randomly generated W_cand and T
3:        while not reached the stopping criterion
4:        do
5:            calculate r
6:            if upper R>rthen
7:                replace W* with W
```

8: **end if**

9: update W by increasing max *r* link weight by adding 1

10: **end while**

11: **end**

Chapter-4: RESULTS

4.1 Network topology 1

For network topology 1 the congestion ratio of each link is given in Table 1,

Iteration	Congestion ratio
1	0.55
2	0.41
3	0.41
4	0.41
5	0.41
6	0.37
7	0.38
8	0.41
9	0.31
10	0.38

Table 1: Maximum Congestion ratio of network topology 1

4.2 Network topology 2

For network topology 2 the congestion ratio of each link is given in Table 2,

Iteration	Congestion ratio
1	0.53
2	0.49
3	0.49
4	0.49
5	0.34
6	0.30
7	0.31
8	0.31
9	0.31
10	0.30

Table 2: Maximum Congestion ratio of network topology 2

4.3 Network topology 3

For network topology 3 the congestion ratio of each link is given in Table 3,

Iteration	Congestion ratio
1	0.80
2	0.70
3	0.70
4	0.50
5	0.69
6	0.50
7	0.69
8	0.50
9	0.69
10	0.51

Table 3: Maximum Congestion ratio of network topology 3

4.4 Discussion

From the three example network topology the minimized maximum congestion ratios are shown in Table 4,

Network topology	Max Utilization
1	0.31
2	0.30
3	0.50

Table 4: Minimized Congestion ratio of three network topology

Chapter-5: Conclusion and Future work

5.1 Conclusion

In this paper we evaluated the proposed method of Kamrul and Oki [1] which has been used to solve the Open Shortest Path First(OSPF) weight settings problem, getting a set of link weight which gives minimum congestion for worst case. We used three example network topology. In which we applied the method. Firstly we calculated the network congestion ratio *r* using Eq. (1), then calculated the maximum congestion ratio *R(w)* using Eq. (2). Finally obtained the optimal link weight using Eq. (3). And from the computational result we found that it leads us to minimize minimum network congestion.

5.2 Future work

In future we can try different methods to solve this problem. Compare the results to see the effectiveness of this method among the other, determine that whether it works well than

other methods or not. And also we want to evaluate it in real life network using real data for example the network topology of different Internet Service Provider (ISP).

Reference:

[1] I. M. Kamrul and E. Oki, "Optimization of OSPF link weight to minimize worst-case network congestion against single-link failure," June 2011, IEEE International Conference on Communications, Page: 1-5.

[2] K.G. Coffman and A.M. Odlyzko. Internet growth: Is there a moore's law for data traffic? Handbook of Massive Data Sets, pages 47–93, 2001

[3] Bernard Fortz, J. Rexford, and MikkelThorup. Traffic engineering with traditional ip routing protocols. IEEE Communicatoins Magazine, pages 118–124, 2002.

[4] Bernard Fortz and MikkelThorup. Increasing internet capacity using local search. Technical Report IS-MG, 2000.

[5] Sadiq M. Sait and Habib Youssef. Iterative Computer Algorithms and their Application to Engineering. IEEE Computer Society Press, December 1999.

[6] Bernard Fortz and MikkelThorup. Internet traffic engineering by optimizing ospf weights. IEEE INFOCOM, 2000.

[7] M Resende Ericsson and P Pardalos. A genetic algorithm for the weight setting problem in ospf routing. Combinatorial Optimisation conference, 2002.

[8] A. Sridharan, R. Gurin, and C. Diot. Achieving near-optimal traffic engineering solutions for current ospf/is-is networks. Sprint ATL Technical Report TR02-ATL-022037, Sprint Labs.

[9] L.S. Buriol, M.G.C. Resende, C.C. Ribeiro, and M. Thorup. A hybrid genetic algorithm for the weight setting problem in ospf/is-is routing. AT&T Labs Research Technical Report, TD-5NTN5G, 2005.

[10] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. IEEE Journal on Selected Areas in Com- munications, 20(4):756–767, 2002.

[11] A. Nucci, N. Taft, “IGP Link Weight Assignment for Optimal Tire-1 Backbones,” IEEE/ACM Transaction on Networking, vol. 15. No. 4, pp.789-802, Aug. 2007.

[12] E. Oki and A. Iwaki, "F-TPR: Fine two-phase IP routing scheme over shortest paths for hose model," IEEE Commun. Letters, vol. 13, no. 4, pp. 277-279, Apr. 2009.

Appendix A

The Source code.

Program.cs

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
usingMicrosoft.Office.Interop.Excel;
usingSystem.Diagnostics;

namespace ProjectV2
{
class Program
   {
static void Main(string[] args)
      {
maxUtilization _maxUtilization = new maxUtilization();
double R;
         R = _maxUtilization.max_utilization();
          Process.Start(@"C:\Users\Shoieb\Desktop\Project\ProjectV2\ProjectV2\bin\Debug\t
          est.txt);
      }
   }
}
```

weightDetermination.cs

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
```

```
usingSystem.Threading.Tasks;
namespace ProjectV2
{
classweightDetermination
   {
static Random rand = new Random();
publicint[,] adjMatrix = new int[100, 100];
publicint[,] traficMatrix = new int[100, 100];
public double[,] loadMatrix = new double[100, 100];
publicint[] predecessor = new int[150];
publicint[] distance = new int[150];
publicbool[] mark = new bool[150];
publicint source;
publicintnumOfVertices;
publicintcurrentNode;
publicint[,] tempAdjArr = new int[100, 100];
publicint[,] tempWeightMatrix = new int[100, 100];
public double[,] tempLoadMatrix = new double[100, 100];
public double[] Load = new double[100];
public double capacity = 100;
public double[] temp = new double[100];
public double r=0;
publicint[] nodeTrace = new int[100];
public void read()
      {
Console.WriteLine("Enter the number of vertices of the graph(should be > 0)\n");
numOfVertices = Convert.ToInt32(Console.ReadLine());

while (numOfVertices<= 0)
         {
Console.WriteLine("Enter the number of vertices of the graph(should be > 0)");
```

```
numOfVertices = Convert.ToInt32(Console.ReadLine());
        }
Console.WriteLine("Enter the adjacency matrix for the graph");
Console.WriteLine("To enter infinity enter:9999");
for (inti = 0; i<numOfVertices; i++)
        {
Console.WriteLine("Enter the (+ve)weights for the row " + i);
for (int j = 0; j <numOfVertices; j++)
            {
adjMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
while (adjMatrix[i, j] < 0)
                {
Console.WriteLine("Weights should be +ve. Enter the weight again");
adjMatrix[i, j] = Convert.ToInt32(Console.ReadLine());
                }
            }
        }
    }
publicweightDetermination()
    {
for (int a = 0; a <numOfVertices; a++)
        {
for (int b = 0; b <numOfVertices; b++)
            {
if (a == b)
                {
traficMatrix[a, b] = 0;
                }
else
                {
traficMatrix[a, b] = rand.Next(1, 10);
```

```
                }
            }
        }
    }
public void resetValue()
    {
loadMatrix = new double[100, 100];
predecessor = new int[150];
distance = new int[150];
mark = new bool[150];
source = 0;
currentNode = 0;
        Load = new double[100];
capacity = 100;
temp = new double[100];
        r = 0;
    }
public void initialize()
    {
for (inti = 0; i<numOfVertices; i++)
        {
mark[i] = false;
predecessor[i] = -1;
distance[i] = 9999;
        }
distance[source] = 0;
    }
publicintgetClosestUnmarkedNode()
    {
intminDistance = 9999;
intclosestUnmarkedNode = 0;
```

```
for (inti = 0; i<numOfVertices; i++)
        {
if ((!mark[i]) && (minDistance>= distance[i]))
            {
minDistance = distance[i];
closestUnmarkedNode = i;
            }
        }
returnclosestUnmarkedNode;
    }
public void calculateDistance(int source, int[,] Arr)
    {
initialize();
intclosestUnmarkedNode;
int count = 0;
while (count <numOfVertices)
        {
closestUnmarkedNode = getClosestUnmarkedNode();
mark[closestUnmarkedNode] = true;
for (inti = 0; i<numOfVertices; i++)
            {
if ((!mark[i]) && (Arr[closestUnmarkedNode, i] > 0))
                {
if (distance[i] > distance[closestUnmarkedNode] +    Arr[closestUnmarkedNode,i])
                    {
distance[i] = distance[closestUnmarkedNode] + Arr[closestUnmarkedNode,i];
predecessor[i] = closestUnmarkedNode;
                    }
                }
            }
count++;
```

```
            }
        }
public void output()
        {
for (inti = 0; i<numOfVertices; i++)
            {
currentNode = i;
if (i == source)
                {
loadMatrix[source, i] += traficMatrix[source, i];
Console.Write((source) + "->" + source);
                }
else
                {
printPath(i);
                }
Console.Write("=" + distance[i]);
Console.WriteLine();
            }
Console.WriteLine();
        }
public void printPath(int node)
        {
if (node == source)
Console.Write(node + "->");
else if (predecessor[node] == -1)
Console.Write("No path from" + source + "to " + node);
else
            {
printPath(predecessor[node]);
loadMatrix[predecessor[node], node] += traficMatrix[source, currentNode];
```

```
Console.Write(node + "->");
        }
    }
public void Dijkstra ( intsrc , int[,] w_mat )
    {
calculateDistance(src, w_mat);
output();
    }
public double LoadCalculation(int[,] mat)
    {
for (int a = 0; a <numOfVertices; a++)
        {
for (int b = 0; b <numOfVertices; b++)
            {
tempAdjArr[a, b] = adjMatrix[a, b];
tempWeightMatrix[a, b] = mat[a, b];
            }
        }
for (int a = 0; a <numOfVertices; a++)
        {
for (int b = 0; b <numOfVertices; b++)
            {
if (tempAdjArr[a, b] == 1)
                {
        tempLoadMatrix[a, b] = tempLoadMatrix[b, a] = (loadMatrix[a, b] +
        loadMatrix[b,a])/capacity;
tempAdjArr[a, b] = tempAdjArr[b, a] = 0;
                }
            }
        }
for (inti = 0; i<numOfVertices; i++)
```

```
        {
for (int j =0;j<numOfVertices;j++)
            {
if (tempLoadMatrix[i,j]>r)
                {
                    r=tempLoadMatrix[i,j];
                }
            }
        }
for (inti = 0; i<numOfVertices; i++)
        {
for (int j = 0; j <numOfVertices; j++)
            {
if (tempLoadMatrix[i, j] == r)
                {
tempWeightMatrix[i, j] += 1;
                }
            }
        }
return r;
      }
   }
}
```

maxUtilization.cs

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Text;
usingSystem.Threading.Tasks;
using System.IO;
namespace ProjectV2
```

```
{
classmaxUtilization
   {
static Random _r = new Random();
weightDetermination _wD = new weightDetermination();

public double bigR = 1;
public double temp_bigR;
public double u;
publicint test = 0;
publicint[,] randomWeight = new int[100, 100];
publicint[,] ini_randomWeight = new int[100, 100];
publicint[,] array = new int[100, 100];
publicint[,] adj_temp = new int[100, 100];
publicinti = 1;
public double[,] tempLoad = new double[100, 100];
public double[,] tempUtilization = new double[100, 100];

public double max_utilization()
      {
adjArray_copy();
while(test < 10)
         {
if (test == 0)
            {
for (int a = 0; a < _wD.numOfVertices; a++)
               {
for (int b = 0; b < _wD.numOfVertices; b++)
                  {
if (a != b &&adj_temp[a,b] == 1)
                     {
```

```
ini_randomWeight[a, b] = ini_randomWeight[b, a] = _r.Next(1, 10);
adj_temp[a, b] = adj_temp [b, a] = 0;
                    }
                }
            }
for (int n = 0; n < _wD.numOfVertices; n++)
            {
                _wD.source = n;
                _wD.Dijkstra(_wD.source, ini_randomWeight);
            }
            u = _wD.LoadCalculation(ini_randomWeight);
for (int a = 0; a < _wD.numOfVertices; a++)
            {
for (int b = 0; b < _wD.numOfVertices; b++)
                {
randomWeight[a, b] = _wD.tempWeightMatrix[a, b];
                }
            }
if (bigR> u && u != 0.0)
            {
bigR = u;
for (int a = 0; a < _wD.numOfVertices; a++)
                {
for (int b = 0; b < _wD.numOfVertices; b++)
                    {
tempLoad[a, b] = _wD.loadMatrix[a, b];
array[a, b] = randomWeight[a, b];
tempUtilization[a, b] = _wD.tempLoadMatrix[a, b];
                    }
                }
            }
```

```
WriteFile(_wD.traficMatrix);
WriteFile(ini_randomWeight, _wD.loadMatrix, _wD.tempLoadMatrix, u, test);
            }
else
            {
                _wD.resetValue();
for (int a = 0; a < _wD.numOfVertices; a++)
                {
for (int b = 0; b < _wD.numOfVertices; b++)
                    {
randomWeight[a, b] = _wD.tempWeightMatrix[a, b];
                    }
                }
for (int n = 0; n < _wD.numOfVertices; n++)
                {
                    _wD.source = n;
                    _wD.Dijkstra(_wD.source, randomWeight);
                }
                u = _wD.LoadCalculation(randomWeight);
if (bigR> u && u != 0.0)
                {
bigR = u;
for (int a = 0; a < _wD.numOfVertices; a++)
                    {
for (int b = 0; b < _wD.numOfVertices; b++)
                        {
tempLoad[a, b] = _wD.loadMatrix[a, b];
array[a, b] = randomWeight[a, b];
tempUtilization[a,b] = _wD.tempLoadMatrix[a,b];
                        }
                    }
```

```
                }
WriteFile(randomWeight, _wD.loadMatrix, _wD.tempLoadMatrix, u, test);
            }
test++;
        }
WriteFile(array, tempLoad, tempUtilization, bigR);
temp_bigR = bigR;
bigR = 1;
returntemp_bigR;
    }
public void WriteFile (int[,] val)
    {
using (StreamWriter writer = new StreamWriter(@"test.txt", true))
        {
writer.WriteLine("Trafic Matrix:");
for (inti = 0; i< _wD.numOfVertices; i++)
            {
for (int j = 0; j < _wD.numOfVertices; j++)
                {
writer.Write(val[i, j] + " ");
                }
writer.WriteLine();
            }
writer.Close();
        }
    }
public void WriteFile(int[,] val,double[,] load_val,double[,] u_val,doubleut, int it)
    {
using (StreamWriter writer = new StreamWriter(@"test.txt", true))
        {
writer.WriteLine("Iteration: " + it);
```

```
writer.WriteLine("Weight Matrix:");
for (inti=0;i<_wD.numOfVertices;i++)
            {
for (int j=0;j<_wD.numOfVertices;j++)
                {
writer.Write(val[i,j]+" ");
                }
writer.WriteLine();
            }
adjArray_copy();
for (int n = 0; n < _wD.numOfVertices; n++)
            {
for (int m = 0; m < _wD.numOfVertices; m++)
                {
if (adj_temp[n, m] == 1)
                    {
writer.WriteLine("Utilization " + n + "-" + m + ": " + u_val[n, m]);
u_val[m, n] = 0.0;
adj_temp[n, m] = adj_temp[m, n] = 0;
                    }
                }
            }
writer.WriteLine("Congestion ratio: " + ut);
writer.WriteLine();
writer.Close();
        }
    }
public void WriteFile(int[,] val, double[,] load_val, double[,] u_val,doubleut)
    {
using (StreamWriter writer = new StreamWriter(@"test.txt", true))
        {
```

```
writer.WriteLine("Minimized Condition: ");
writer.WriteLine("Weight Matrix:");
for (inti = 0; i< 6; i++)
            {
for (int j = 0; j < 6; j++)
                {
writer.Write(val[i, j] + " ");
                }
writer.WriteLine();
            }
adjArray_copy();
for (int n = 0; n < 6; n++)
            {
for (int m = 0; m < 6; m++)
                {
if (adj_temp[n, m] == 1)
                    {
writer.WriteLine("Utilization " + n + "-" + m + ": " + u_val[n, m]);
u_val[m, n] = 0.0;
adj_temp[n, m] = adj_temp[m, n] = 0;
                    }
                }
            }
writer.WriteLine("Congestion ratio: " + ut);
writer.WriteLine();
writer.Close();
        }
    }

public void adjArray_copy ()
    {
```

```
for (int a = 0; a < _wD.numOfVertices; a++)
        {
for (int b = 0; b < _wD.numOfVertices; b++)
            {
adj_temp[a, b] = _wD.adjMatrix[a, b];
            }
        }
    }
  }
}
```