DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

MESTRADO INTEGRADO EM ENG. DE COMPUTADORES E TELEMÁTICA

ANO 2014/2015

DESEMPENHO E DIMENSIONAMENTO DE REDES

ASSIGNMENT GUIDE NO. 3

TRAFFIC ENGINEERING OF PACKET SWITCHED NETWORKS

Assignment duration:	**8 May – 5 June**
Report delivery deadline:	**12 June**

NOTE: The report should include: (i) the responses to all questions of this guide and (ii) the code of all developed MATLAB codes with well explained comments.

Consider an MPLS (Multi-Protocol Label Switching) network of an ISP (Internet Service Provider) with the topology presented in Figure 1. Consider all connections with a bandwidth capacity of 1 Gbps. In Annex I, the *R* matrix defines, in MATLAB format, the network topology shown in Figure 1 where element *R*(*i*,*j*) is 1 when the connection between nodes *i* and *j* exists.

Consider that, in all network connections, the propagation delay is approximately the light speed (3×10^8 meters per second). In Annex I, the *L* matrix indicates, in MATLAB format, the connection lengths (in kilometers) where element *L*(*i*,*j*) indicates the length of the connection from node *i* to node *j*.

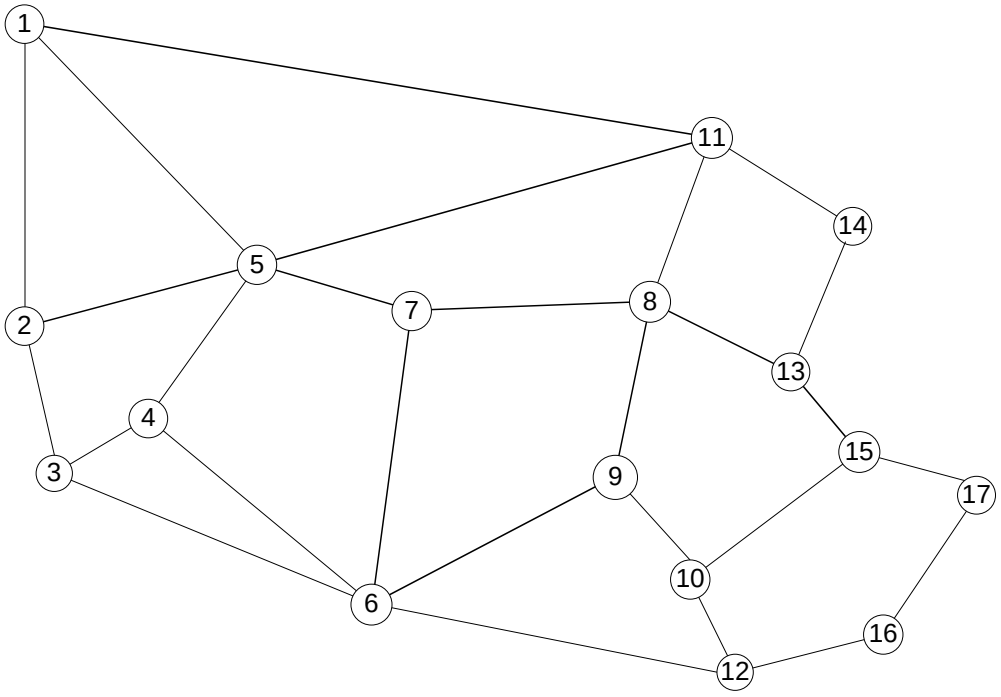

Figure 1: Topology of an ISP Network

Consider that the network must support the estimated traffic matrix presented in Figure 2 (the values are in Mbps). In Annex I, the *T* matrix defines, in MATLAB format, the traffic matrix presented in Figure 2 where element *T*(*i*,*j*) defines the average traffic bandwidth sent by node *i* to node *j*.

Assume that in all traffic flows, the packet arrivals are Poisson processes and the packet sizes are exponentially distributed with an average size of 1000 Bytes.

Consider that the ISP requires each traffic flow to be routed through a single LSP (Label Switched Path). The LSPs between the same nodes are required to be symmetrical (*i.e.*, the network connections of an LSP from node *i* to node *j* must be the same network connections of the LSP from node *j* to node *i*).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	27	78	18	14	60	14	17	13	13	11	15	15	55	17	14	12
2	60	0	80	38	44	290	79	62	65	51	78	70	51	280	42	64	54
3	82	82	0	11	18	70	18	19	25	19	15	48	17	84	18	19	15
4	18	51	18	0	12	41	19	16	15	17	41	16	12	42	15	14	19
5	10	42	14	14	0	63	15	12	13	17	48	15	17	61	17	10	16
6	69	320	71	73	41	0	52	62	55	54	71	59	56	210	54	94	83
7	18	43	15	15	17	72	0	10	16	17	16	19	75	73	13	12	18
8	18	62	19	15	13	50	14	0	16	12	15	18	16	76	13	14	13
9	13	44	22	16	19	51	17	13	0	15	18	12	18	74	15	18	16
10	17	70	14	13	15	61	18	14	14	0	12	13	15	56	14	15	12
11	12	48	13	41	44	54	13	14	16	19	0	13	14	63	12	15	11
12	14	81	47	13	13	79	19	11	11	14	16	0	13	44	13	37	13
13	14	43	12	16	12	47	79	15	13	19	13	18	0	50	15	17	12
14	54	240	65	74	56	250	73	85	76	42	42	44	61	0	54	76	53
15	16	49	14	13	14	49	11	19	11	14	13	16	15	63	0	15	14
16	14	63	13	15	15	59	13	15	14	12	12	17	12	75	19	0	15
17	18	45	14	14	18	73	16	17	13	12	14	17	12	59	14	12	0

Figure 2: Traffic Matrix

For the implementation of this assignment, a MATLAB function is provided (file `ShortestPathSym.m`) that, for a given pair of nodes, determines the routing path that minimizes the sum of the connection costs in both directions between them. The function sintax is:

```
function [route]= ShortestPathSym(costs,node1,node2)
```

Input parameters:

`costs` – a square matrix of 17×17 elements where element `costs(i,j)` defines the cost of the connection from node *i* to node *j*;
`node1` – the first end node;
`node2` – the second end node.

NOTES:
- the cost values must be non-negative;
- the function ignores the values `costs(i,j)` for pairs of nodes such that there is no connection in the ISP network (as defined in Figure 1).

Output parameter:

`route` – a line array of 17 elements with the sequence of nodes in the routing path from `node1` up to `node2` (the remaining elements are zero).

Error codes:

When the output parameter `route` is a single element, its content indicates the following error in the input parameters:
`-1` : the matrix `costs` is not a square matrix of 17×17 elements
`-2` : the `node1` and/or `node2` are either equal or invalid
`-3` : at least one cost value is negative

a) Assume that the routing of each flow is done on the shortest path given by the sum of the lengths of the connections. Consider this solution as solution A. Implement a MATLAB function to determine the average and the maximum connection load of solution A (the load of a connection is the total bandwidth supported by the connection divided by its bandwidth capacity). Note that each connection is bi-directional and, therefore, has two load values, one on each of its directions.

b) For solution A, add to the previous MATLAB function an appropriate code to determine, based on the Kleinrock approximation: (i) the network average round-trip delay, (ii) the flow (identified by its end nodes) with the highest average round-trip delay and (iii) the value of the highest average round-trip delay.

c) For solution A, add to the previous MATLAB function an appropriate code to present graphically in decreasing order: (i) the average packet round-trip delay of each flow and (ii) the load of each direction of each connection.

d) Compute a different solution in the following way. Start by considering the network empty. Then, for each flow, compute as its routing path the one that minimizes the sum of the connection loads that resulted from the previous selected routing paths. Consider this solution as solution B. Implement a MATLAB function to determine the average and the maximum connection load of solution B.

e) Repeat b) and c) for solution B. Analyzing the results, show that solution B is better than solution A both in terms of maximum connection loads and round-trip delay parameters. Explain why the method used to compute solution B has produced a better solution than the method used to compute solution A.

f) Compute another solution in the following way. Start by considering the network empty. Then, for each flow, select as its routing path the one that has the minimum average round-trip delay resulted from the previous selected routing paths (use the M/M/1 assumption for the delay of each connection). Consider this solution as solution C. Determine the average and the maximum connection load of solution C.

g) Repeat b) and c) for solution C. Analyzing the results, show that solution C is better than solution A. Between solutions B and C, draw conclusions on which solution is better and in what cases.

h) Develop a multi-start local search optimization algorithm for n cycles, where n is an input parameter (see Annex II), to find a solution with the lowest network average round-trip delay. At the end, the algorithm outputs the highest average round-trip delay, the network average round-trip delay and the maximum connection load of the best found solution. Run your algorithm for n = 3, 10 and 30 cycles. Analyze the values of the solutions founds in the 3 cases. What do you conclude?

i) Adapt the previous developed algorithm to find a solution with the lowest maximum connection load. Run your algorithm for n = 3, 10 and 30 cycles. Analyze the values of the solutions founds in the 3 cases. What do you conclude?

j) Between the best solution found in h) and the best solution found in i), which one is, in your opinion, the solution that the ISP should adopt on its network? Explain your opinion.

ANNEX I – PROBLEM DATA MATRICES

```
R= [0  1  0  0  1  0  0  0  0  0  1  0  0  0  0  0  0
    1  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0  0
    0  1  0  1  0  1  0  0  0  0  0  0  0  0  0  0  0
    0  0  1  0  1  1  0  0  0  0  0  0  0  0  0  0  0
    1  1  0  1  0  0  1  0  0  0  1  0  0  0  0  0  0
    0  0  1  1  0  0  1  0  1  0  0  1  0  0  0  0  0
    0  0  0  0  1  1  0  1  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  1  0  1  0  1  0  1  0  0  0  0
    0  0  0  0  0  1  0  1  0  1  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0  1  0  0  1  0  0  1  0  0
    1  0  0  0  1  0  0  1  0  0  0  0  0  1  0  0  0
    0  0  0  0  0  1  0  0  0  1  0  0  0  0  0  1  0
    0  0  0  0  0  0  0  1  0  0  0  0  0  1  1  0  0
    0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  0  0
    0  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0  1
    0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  1
    0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  0];
```

```
L= [0   179 0   0   207 0   0   0   0   0   422 0   0   0   0   0   0
    179 0   91  0   155 0   0   0   0   0   0   0   0   0   0   0   0
    0   91  0   60  0   206 0   0   0   0   0   0   0   0   0   0   0
    0   0   60  0   113 176 0   0   0   0   0   0   0   0   0   0   0
    207 155 0   113 0   0   95  0   0   0   281 0   0   0   0   0   0
    0   0   206 176 0   0   180 0   166 0   0   227 0   0   0   0   0
    0   0   0   0   95  180 0   140 0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   140 0   101 0   113 0   102 0   0   0   0
    0   0   0   0   0   166 0   101 0   76  0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   76  0   0   49  0   0   127 0   0
    422 0   0   0   281 0   0   113 0   0   0   0   0   103 0   0   0
    0   0   0   0   0   227 0   0   0   49  0   0   0   0   0   91  0
    0   0   0   0   0   0   0   102 0   0   0   0   0   102 59  0   0
    0   0   0   0   0   0   0   0   0   0   103 0   102 0   0   0   0
    0   0   0   0   0   0   0   0   0   127 0   0   59  0   0   0   70
    0   0   0   0   0   0   0   0   0   0   0   91  0   0   0   0   103
    0   0   0   0   0   0   0   0   0   0   0   0   0   0   70  103 0];
```

```
T=  [0  27  78  18  14  60  14  17  13  13  11  15  15  55  17  14  12
     60 0   80  38  44  290 79  62  65  51  78  70  51  280 42  64  54
     82 82  0   11  18  70  18  19  25  19  15  48  17  84  18  19  15
     18 51  18  0   12  41  19  16  15  17  41  16  12  42  15  14  19
     10 42  14  14  0   63  15  12  13  17  48  15  17  61  17  10  16
     69 320 71  73  41  0   52  62  55  54  71  59  56  210 54  94  83
     18 43  15  15  17  72  0   10  16  17  16  19  75  73  13  12  18
     18 62  19  15  13  50  14  0   16  12  15  18  16  76  13  14  13
     13 44  22  16  19  51  17  13  0   15  18  12  18  74  15  18  16
     17 70  14  13  15  61  18  14  14  0   12  13  15  56  14  15  12
     12 48  13  41  44  54  13  14  16  19  0   13  14  63  12  15  11
     14 81  47  13  13  79  19  11  11  14  16  0   13  44  13  37  13
     14 43  12  16  12  47  79  15  13  19  13  18  0   50  15  17  12
     54 240 65  74  56  250 73  85  76  42  42  44  61  0   54  76  53
     16 49  14  13  14  49  11  19  11  14  13  16  15  63  0   15  14
     14 63  13  15  15  59  13  15  14  12  12  17  12  75  19  0   15
     18 45  14  14  18  73  16  17  13  12  14  17  12  59  14  12  0];
```

ANNEX II –MULTI -START LOCAL SEARCH

A multi-start local search algorithm combines a randomized greedy algorithm with a local search algorithm. These two components are first introduced and, then, the final algorithm is described.

II.1 – Greedy Algorithm

In a greedy algorithm, a solution is computed by selecting each solution component one at a time.

In the traffic engineering problem addressed in this guide, the routing solution is a routing path that must be selected for each origin-destination pair of nodes. Therefore, in a greedy algorithm, we select the routing path for the first node pair, then, we select the routing path for the second node pair taking into account the path selected for the first node pair, and so on…

The methods used to compute solution A, in **a)**, solution B, in **d)**, and solution C, in **f)**, are examples of greedy algorithms.

II.2 – Greedy Randomized Algorithm

In a greedy randomized algorithm, like in the greedy algorithm, a solution is computed by selecting each solution component one at a time. Nevertheless, the aim is to add some randomness to the algorithm in order to avoid that different runs of the same algorithm generate the same solution.

Note that in the methods used to compute solution B and solution C, the routing path selected for each node pair depends on the routing paths selected for the previous node pairs. Therefore, different orders produce different solutions. An easy way to obtain a greedy randomized algorithm is to use a random order for the sequence of node pairs that is used to select the paths.

A simple MATLAB code to generate a random order for the network of this guide is:

```
for i=1:16
    for j=(i+1):17
       ord= [ord; i j];
    end
end
npares= size(ord,1);
b= randperm(npares);
for i= 1:npares
    ordem(i,:)= ord(b(i),:);
end
```

At the end, the matrix `ordem` defines a random order of all origin-destination node pair: it has two columns (defining origin-destination pairs) and a number of rows equal to the total number of origin-destination pairs.

II.3 – Local Search Algorithm

In a local search algorithm, we start by an initial solution and try to move to a better solution by making local changes. The moves are repeated until no possible local change produces any better solution.

In more detail (see Figure II.1), the initial solution *x* is a given previously computed solution (generated, for example, by a greedy algorithm). Then, a local search algorithm is an iterative process where, on each iteration, a set of neighbor solutions *V*(*z*) are computed and if the best neighbor solution *y* is better than the current solution *x*, the neighbor solution becomes the current solution (in Figure II.1, *F*(*x*) is the value of the objective function of solution *x*). The iterative process stops when the best neighbor solution is not better than the current solution and the current solution *x* is the resulting local minimum solution.

```
x = initial solution
repeat
        z = x
        for y ∈ V(z) do:
                if F(y) better than F(x) then  x = y
until z == x
```

Figure II.1: Local Search

Remember once again that, in the traffic engineering problem addressed in this guide, the routing solution defines a routing path for each origin-destination pair of nodes. A possible neighbor set is the set of solutions where one routing path is changed by another. Therefore, there is a neighbor solution for each origin-destination pair of nodes. The neighbor solution for a particular node pair *p* is computed by considering the same routing paths for all other node pairs and, then, to select the routing path of *p* using either the criterion used for solution B or the criterion used for node C (the one most convenient for your optimization goal).

II.4 – Multi-Start Local Search Algorithm

A multi-start local search algorithm is a combination of a greedy randomized algorithm and a local search algorithm (see Figure II.2). The greedy randomized algorithm is used to generate different initial solutions *x* and the local search is used to improve each initial solution and find its closest local minimum solution *z*. If the local minimum solution *z* is better than the global best solution computed in all previous iterations, it becomes the best global solution x_{best} (in Figure II.2, *F*(*z*) is the value of the objective function of solution *z*). One possible stopping criterion is to run a predefined number of iterations *n*.

```
F_best = +∞
repeat
        x = GreedyRandomized()
        z = LocalSearch(x)
        if F(z) better than F_best then  x_best = z  e  F_best = F(z)
until Stopping Criteria is met
```

Figure II.2: Multi-Start Local Search

A MATLAB function code that implements a multi-start local search algorithm for the minimization of the network average round-trip delay is:

```
function [F_MaxDelay F_AverageDelay F_MaxLoad]= Optimize(Iterations)
    Matrizes;
    GlobalBest= Inf;
    for iter=1:Iterations
        routes= BuildSolution();
        [CurMaxDelay CurAverageDelay CurMaxLoad]= Evaluate(routes);
        repeat= true;
        while repeat
            LocalBest= Inf;
            for i=1:size(routes,1)
                routesAux= BuildNeighbor(routes,i);
                [MaxDelay AverageDelay MaxLoad]= Evaluate(routesAux);
                if AverageDelay < LocalBest
                    LocalBest= AverageDelay;
                    LocalBestRoutes= routesAux;
                end
            end
            if LocalBest < CurAverageDelay
                CurAverageDelay= LocalBest;
                routes= LocalBestRoutes;
            else
                repeat= false;
            end
        end
        if LocalBest < GlobalBest
            GlobalBest= LocalBest;
            [F_MaxDelay F_AverageDelay F_MaxLoad]= Evaluate(routes);
            GlobalBestRoutes= routes;
        end
    end
end

function routes= BuildSolution()
    %Computes an initial solution with a random order of all node
    %pairs. Outputs a matrix with the routing path of each node pair.
end

function [MD AD ML]= Evaluate(routes)
    %Uses the Kleinrock approximation to compute the highest average
    %round-trip delay (MD), the global average round-trip delay (AD)
    %and the maximum connection load (ML) of the solution given by the
    %routing paths defined in the input matrix 'routes'
end

function routesAux= BuildNeighbor(routes,i)
    %Outputs in matrix 'routesAux' the neighbor solution of 'routes'
    %by changing the routing path of the ith flow.
end
```

In this code, you just need to implement the functions: (i) `BuildSolution()` to compute the initial solutions, (ii) `Evaluate(routes)` to compute the performance parameters of a solution and (iii) `BuildNeighbor(routes,i)` to compute a neighbor solution.