Desempenho e Dimensionamento de Redes - 2018/2019

Engenharia de Computadores e Telemática

5º Guião Prático - Relatório

TRAFFIC ENGINEERING OF TELECOMMUNICATION NETWORKS

P4G2
Diego Hernandez nº 77013
Ricardo Pousa nº 80328

Minimizing Worst Link Load:

K	Worst Link Load	Time elapsed
1	0.891	0.003 s
2	0.684	1.064 s
3	0.648	2.289 s
4	0.641	3.138 s
5	0.641	3.765 s
10	0.641	6.901 s
15	0.641	10.04 s
20	0.641	17.92 s
25	0.641	17.80 s
30	0.641	21.58 s

Observations:

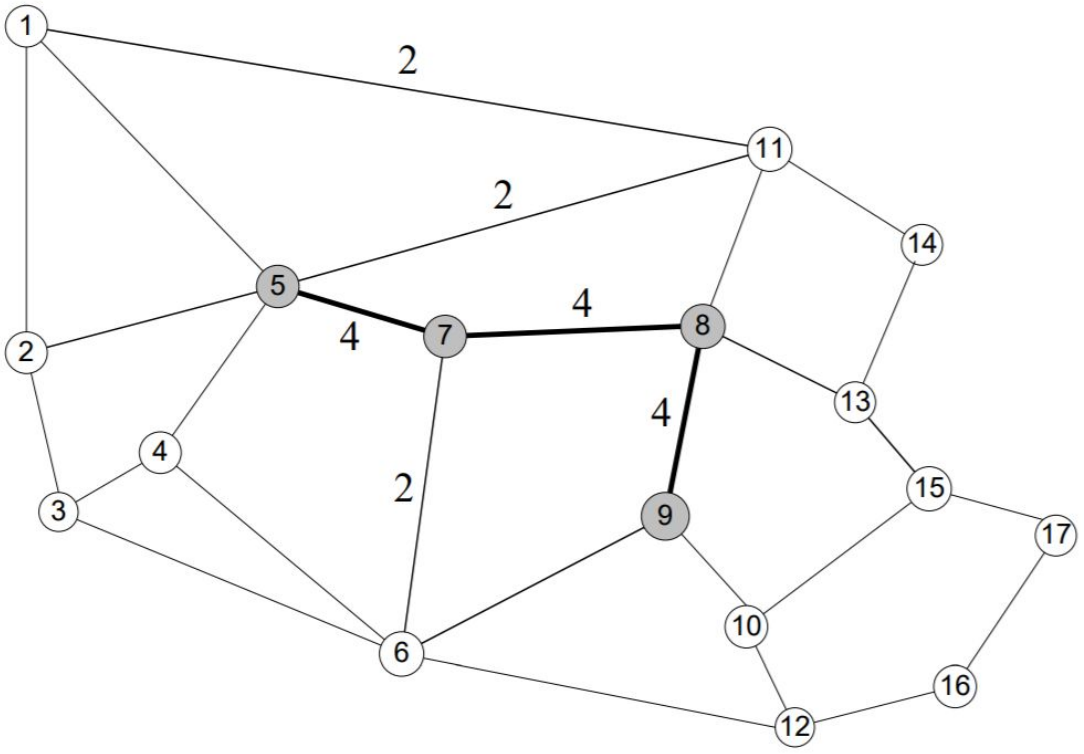
Taking into account the worst link load for each iteration, we can see (even though not asked) that if we increase the possible paths from which we can select a better solution, from 1 to 5 we notice a decrease on the worst link load, after that point there is no noticeable improvement (if at all) from increasing the amount of possible paths from 5 to 30 even though the amount of time it takes to arrive at a solution increases constantly accompanying the increase of paths.

Having this said we found that the sweet spot to calculate the best solution in which the worst link load is minimized, is if take into account 5 paths only. (Regardless of we already having arrived at the best solution for 4 paths, 5 paths mark would give us a somewhat bigger margin to work with while minimizing the worst link load).

Prob1.m: Code used to produce the aforementioned results:

```
clear all;
Matrices;
nNodes= size(L,1);
%k= 5, 10, 15, 20, 25 and 30
nPaths= 5; % nPaths = k
f= 0;
for i=1:nNodes
   for j= 1:nNodes
      if T(i,j)>0
```

```
        f= f+1;
        flowDemand(f) = T(i,j);
        [shortestPaths{f}, tc] = kShortestPath(L, i, j, nPaths);
        if isempty(tc)
          fprintf('Error: no connectivity\n');
        end
      end
    end
end
nFlows= length(flowDemand);
solution= ones(1,nFlows);
worstlinkload= maxLoad(solution,shortestPaths,flowDemand,R);
improved = true;
count = 0;
best_solution = ones(1,nFlows);
best_worstlinkload = worstlinkload;
tic
while improved
   count = count + 1;
   [count best_worstlinkload];
   for f = 1:nFlows
      for p = 1:length(shortestPaths{f})
         if solution(f) ~= p
           tmp_solution = solution;
           tmp_solution(f) = p;
           tmp_worstlinkload = maxLoad(tmp_solution, shortestPaths, flowDemand, R);
           if(best_worstlinkload > tmp_worstlinkload)
             best_worstlinkload = tmp_worstlinkload;
             best_solution = tmp_solution;
           end
         end
      end
   end
   if(worstlinkload > best_worstlinkload)
     worstlinkload = best_worstlinkload;
     solution = best_solution;
   else
     improved = false;
   end
end
fprintf("worst link load: %d  \n",worstlinkload)
fprintf("toc: %f\n",toc)
```



Energy consumption of each link, being that when not marked it's 1, and the bandwidth is 1Gbps, the bolder links have a 10 Gbps capacity. This will help contextualize and visualize the following results.

Minimizing Consumption :

Criteria	Worst Link Load Limit	Actual Worst Link Load	Energy Consumption	Removed Links
Worst Link Load	0.7	0.669	33	4-5 1-11 3-4 10-12 13-15
	0.75	0.669	33	4-5 1-11 3-4 10-12 13-15
	0.8	0.779	27	4-5 1-11 3-4 10-12 13-15 7-8 8-11 1-2
	0.85	0.779	27	4-5 1-11 3-4 10-12 13-15 7-8 8-11 1-2

Ratio	0.7	0.687	29	1-11 5-11 8-9 1-5 15-17
	0.75	0.745	24	8-9 5-7 1-11 1-2 10-12 10-15 8-11 3-4
	0.8	0.795	23	8-9 5-7 1-11 1-2 6-12 10-15 8-11 3-4
	0.85	0.795	23	8-9 5-7 1-11 1-2 6-12 10-15 8-11 3-4

Observations:

In this exercise we arrived at a somewhat expected results when the stopping criteria was the worst link load , at each stepping point of load the actual worst link load was better than its' counterpart of ratio optimisation. But the contrary is the case when the optimisation is done for the ratio and it's given more importance to reducing the energy consumption.

The fact that the ratio in this case is the worst link load divided by the energy consumed, it's easy to see how a small part of the results each step is influenced by a decrease in link load, but an even greater part can be attributed to the decrease in energy spent, giving us a better solution for each step when it comes to energy spent. And the plain and simple minimization of only the link load would, in its turn, wield better results for link load.

Prob2.m: Code used to produce the aforementioned results:

```
clear all;
Matrices;
nPaths= 5;

%gets all bi directional links  and the number of links.
[row,col] = find(L ~= Inf);
Links = [row col];
Links = Links(Links(:,1)<Links(:,2),:);
numLinks=size(Links,1);

%worst link loads values
w = [0.7 0.75 0.80 0.85];
w_index = 4;

%removedd link storage
worst_link_load = w(w_index)
removed_links = [];
```

```
while( worst_link_load <= w(w_index))
   best_calculated_worst_link_load = Inf;
   best_energy_calculated_worst_link_load = Inf;
   linkIndex = -1;
   %determinar o best worst link load ao retirar o link
   for index = 1:numLinks %percorro todos os links disponiveis na rede
      %removes the link at the network
      [Temp_L, Temp_C, Temp_R] = remove_link(L, R, C, Links(index,1), Links(index,2));
      %calculates the best worst link load of network
      calculated_worst_link_load = compute_worst_link_load(T, Temp_L, Temp_R,nPaths);

      energy_calculated_worst_link_load = calculated_worst_link_load./C(Links(index,1),Links(index,2));
      %Nota: utilizar uma das duas condicoes!
      if(energy_calculated_worst_link_load < best_energy_calculated_worst_link_load &&
calculated_worst_link_load<=w(w_index))  %!!  used when optimizing ratio

          if(calculated_worst_link_load < best_calculated_worst_link_load &&
calculated_worst_link_load < w(w_index)) % !! used when optimizing for worst link load
        best_energy_calculated_worst_link_load = energy_calculated_worst_link_load;
        best_calculated_worst_link_load =  calculated_worst_link_load;
        linkIndex = index;
      end
   end
   if(best_calculated_worst_link_load > w(w_index))
     break;
   end
   worst_link_load = best_calculated_worst_link_load;
   %removes permanently link
   tmpLink = Links(linkIndex, :);
   tmpLink;
   %stores links from removed links matrix
   removed_links = [removed_links; tmpLink];
   Links(linkIndex,:) = [];
   numLinks=size(Links,1);
   [L, C, R] = remove_link(L, R, C, tmpLink(1,1), tmpLink(1,2));
   fprintf("Worst link load: %d\n",worst_link_load);
end
fprintf("final worst link load: %d\n",worst_link_load);
fprintf("final energy consumption: %d\n",sum(sum(C))/2);
removed_links


%removes desired link of the network
function [Temp_L,Temp_C, Temp_R] = remove_link(L,R,C,i,j)
  Temp_L = L;
  Temp_R = R;
```

```
  Temp_C = C;
  Temp_C(i,j) = 0;
  Temp_C(j,i) = 0;
  Temp_R(i,j) = 0;
  Temp_R(j,i) = 0;
  Temp_L(i,j)= inf;
  Temp_L(j,i)= inf;
end

%computes worst link load of the network
function worstlinkload = compute_worst_link_load(T,L,R, nPaths)
  f= 0;
  nNodes= size(L,1);
  for i=1:nNodes
    for j= 1:nNodes
      if T(i,j)>0
        f= f+1;
        flowDemand(f) = T(i,j);
        [shortestPaths{f}, tc] = kShortestPath(L, i, j, nPaths);
        if isempty(tc)
          worstlinkload = Inf;
          return
          %fprintf('Error: no connectivity\n');
        end
      end
    end
  end

  nFlows= length(flowDemand);
  solution= ones(1,nFlows);
  worstlinkload= maxLoad(solution,shortestPaths,flowDemand,R);
  improved = true;
  count = 0;
  best_solution = ones(1,nFlows);
  best_worstlinkload = worstlinkload;
  while improved
    count = count + 1;
    %[count best_worstlinkload]
    for f = 1:nFlows
      for p = 1:length(shortestPaths{f})
        if solution(f) ~= p
          tmp_solution = solution;
          tmp_solution(f) = p;
          tmp_worstlinkload = maxLoad(tmp_solution, shortestPaths, flowDemand, R);
          if(best_worstlinkload > tmp_worstlinkload)
            best_worstlinkload = tmp_worstlinkload;
```

```
                best_solution = tmp_solution;
              end
            end
          end
        end

        if(worstlinkload > best_worstlinkload)
          worstlinkload = best_worstlinkload;
          solution = best_solution;
        else
          improved = false;
        end
      end
    end
```