INSTITUTO SUPERIOR TÉCNICO

MEEC

Capacitated Routing Code (*routing_v2*)

Author:
Alexandre Freitas, 90005

Professor:
João José Pires

October 2023

How to use *routing_ v2*:

- Define the adjacency matrix of the network, *matrix*.
- Define a traffic matrix, *traffic*. If no traffic matrix is defined a default traffic matrix for a full-mesh logical topology (one unit of traffic from one node to every other) is automatically created.
- Define the maximum capacity of the links (maximum number of traffic units of each link), *MAX_ LINK_ CAP*. By default a very large value is defined, so the routing is made as in an uncapacitated routing problem.
- Define the type of order ("shortest", "longest" or "largest") to sort the paths, *sorting_ order*.
- Run the script.

The results (number of nodes, number of links, average node degree, network diameter, average number of hops per demand, minimum, maximum and average link lengths, the paths/routes chosen in the order they were routed, the number of paths, the minimum, maximum and average path length, the loads in every link, the average load per link, the amount of blocked traffic, the paths in which the traffic was blocked and the blocking probability) are printed in the terminal.

Capacitated Routing Example:

Figure 1 represents the network used to exemplify the use of the the *routing_ v2.py* code. A logical full-mesh topology was considered and the sorting strategy was shortest-first. The maximum link capacity considered was 5 units of traffic.

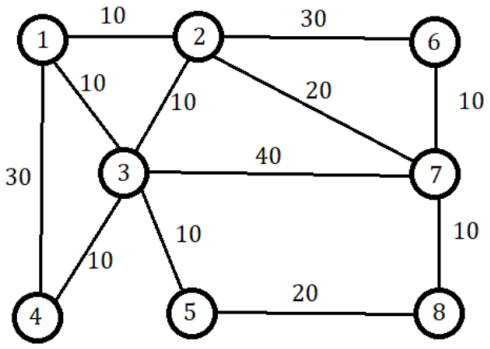

Figure 1: Test Network - Physical Topology with the link lengths.

The first 46 traffic demands (the ones with distances 10, 20, 30 and four with distance 40) are routed through the following paths: [1, 2], [2, 1], [1, 3], [3, 1], [2, 3], [3, 2], [3, 4], [4, 3], [3, 5], [5, 3], [6, 7], [7, 6], [7, 8], [8, 7], [1, 3, 4], [4, 3, 1], [1, 3, 5], [5, 3, 1], [2, 3, 4], [4, 3, 2], [2, 3, 5], [5, 3, 2], [2, 7], [7, 2], [4, 3, 5], [5, 3, 4], [5, 8], [8, 5], [6, 7, 8], [8, 7, 6], [1, 2, 7], [7, 2, 1], [2, 6], [6, 2], [2, 7, 8], [8, 7, 2], [3, 2, 7], [7, 2, 3], [3, 5, 8], [8, 5, 3], [5, 8, 7],[7, 8, 5], [1, 2, 6], [6, 2, 1], [3, 2, 6] and [6, 2, 3].

The residual capacity of the network after the routing of these demands is represented in figure 2.

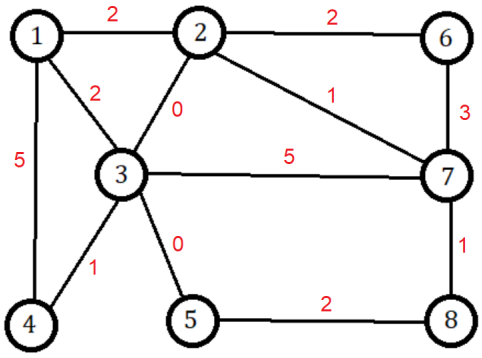

Figure 2: Test Network - Residual Capacity represented bidirectionally (1).

The next traffic demand to route is 4->7 with the path [4,3,2,7], but since the link (3,2) has a residual capacity of zero, an alternative routing path is determined: [4,3,7]. The computation of this path now excludes the saturated link (3,2). The same process is done for the traffic demand 7->4.

Afterwards the traffic demand 4->8 is routed. Normally this would be done through the shortest path [4,3,5,8], but, since the link (4,3) has now (after routing 4->7) residual capacity zero, an alternative path is computed: [4,1,2,7,8]. The same occurs for the symmetric demand 8->4.

After the routing of these paths, the residual capacity of the network is now represented in figure 3.

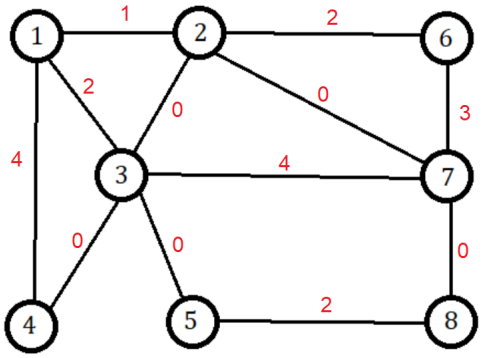

Figure 3: Test Network - Residual Capacity represented bidirectionally (2).

The next traffic demand to route is 5->6. This would normally be routed through the path [5, 8, 7, 6], but since the residual capacity of link (8,7) is zero, an alternative path needs to be determined. However, since any alternative path to this one would necessarily include link (5,3) and this link's residual capacity is also zero, it is impossible to route the traffic demand 5->6. This traffic is then blocked. The same thing happens to the symmetric demand 6->5.

The next traffic demands to route are 1->8 and 8->1. Given the residual capacities of the network, it is also impossible to establish a path between these nodes, and so this traffic is also blocked.

Finally, the last traffic demands 4->6 and 6->4 are routed through paths [4, 1, 2, 6] and [6, 2, 1, 4]. The loads in each of the links at the end of the routing process are represented in figure 4. The total amount of blocked traffic is 4.

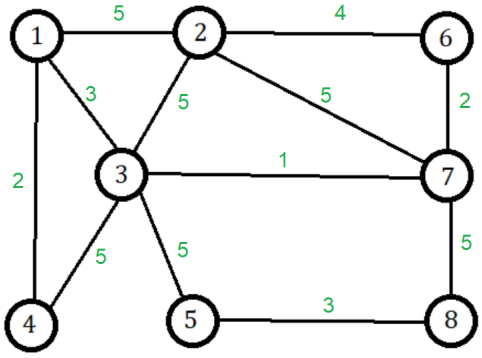

Figure 4: Test Network - Loads in each link.

These results are in accordance with the results obtained from *routing_v2.py*.

The loads in each of the links are represented in the *load_matrix*. The element in position i, j of this matrix will have, after routing, the total load in the link between nodes i and j.

The *distance_matrix* will have the path length (distance) between the nodes; that is, the element in position i, j of this matrix will have, after routing, the length of the chosen path with source in node i and destination in node j. In the capacitated routing, the values in *distance_matrix* may or may not be the same as the original distances used to sort the paths, depending on whether there was the need to choose an alternative path or not.

Other matrix used is *matrix_cp*, which is a copy of *matrix* (the adjacency matrix of the network) to which changes are made in the capacitated routing process (the removing of links by setting their value to 99999).

Ordered Paths:

The ordered paths printed as results are presented in the order they were routed, and follow the following structure:

Path: [7, 3, 4] , Dist: 50 , Og.Dist: 40 , traffic = 1 (Og. Path: [[7, 2, 3, 4]])

"Path" is the path chosen for a given traffic demand, "distance" is the length of that path, "original distance" is the distance of the original shortest path (the distance used to define the sorting order) and "original path" is the shortest path chosen before the routing. "Path" and "original path" (as well as "distance" and "original distance") will only be different if a recalculation of the path is necessary in the case of a residual capacity in the original path being zero. This representation allows to see the original shortest path as well as the alternative path chosen given the capacity limitation on the links.