POLITECNICO DI TORINO

Operational Research

Labs Report

Professor:
Emilio Leonardi

Daniele Manerba

GROUP 15:
Lorenzo Bellone s258340

Giulia Ciaramella s265764

Matteo Zhang s267461

A.Y.2018-2019

Contents

Chapter 1

Lab 1: The Logical Topology Design Problem (LTD)

1.1 Introduction

The Logical Topology Design problem (or, LTD for short) is a network design problem that consists in defining an optimal network configuration that fulfills specified traffic demands under technical and economical constraints.

During the configuration of the network topology, given a node-to-node traffic, the goal is to find the best logical topology that minimizes the cost by finding the best compromise between light-path cost and switching cost at the IP, SONET/sdH or ATM layers.

The LTD problem can be formulated as an integer-linear program, in particular, as an NP-Hard problem.

1.2 Problem formalization

Given a fixed number of nodes and the node degree, i.e. the maximum number of links per node, the problem formalization consists in the mathematical formulation of:

- the parameters;
- the input;
- the variables;
- the constraints;
- the objective function;

The **parameters** are already mentioned above i.e. :

- N, the number of nodes in the network
- Δ, the number of links per node consisting in δ^{Rx} *and* δ^{Tx} (respectively, the number of receiving links and number of transmitting links)

The **input** consists of the traffic from the source node S to the destination node D defined as:

$$[t^{sd}]$$

Note that in this problem, any node can be the source or the destination of a different traffic flow.

The **variables** are defined as:

- f_{ij}^{sd}, the amount of traffic going from one specific source to one specific destination, passing through the link connecting nodes i-j;
- f_{ij}, the amount of total traffic flowing on the lightpath from node i to node j (transit nodes) for any source and destination;
- $b_{ij} = \{0, 1\}$, the **topology indicator**; it indicates if the flow passes on the link i-j. $b_{ij} = 1$ if the link exists, 0 otherwise.

The **constraints** are:

- $f_{ij}^{sd} \leq b_{ij} t^{sd} \quad \forall i, j, s, d$, the **feasibility constraint**, this forces the flow to be less or equal than the total traffic s-d flowing through arc i-j if the link i-j exists, and the flow to be 0 if the link i-j does not exist in the logical topology.
- $\sum_j b_{ij} \leq \delta_i^{Tx} \quad \forall i$, **outgoing degree constraint**, each node has a maximum of outgoing links equal to δ_i^{Tx}
- $\sum_j b_{ji} \leq \delta_i^{Rx} \quad \forall i$, **incoming degree constraint** each node has a maximum of incoming links equal to δ_i^{Rx}
- $f_{ij} = \sum_{sd} f_{ij}^{sd} \quad \forall i, j$, the constraints that relates the total flow with the partial flow on the link i-j;
- $\sum_j f_{Sj}^{sd} = t^{sd}$, the total amount of flow outgoing from the source S must be equale to the traffic demand (or 1 in the normalized case)
- $\sum_j f_{jD}^{sd} = t^{sd}$ (or 1 in the normalized case)
- $\sum_j f_{ij}^{sd} = \sum_j f_{ji}^{sd} \quad with \quad i \neq s, d$, **transition equation** (or conservation equation): the total amount of traffic incoming in a node is the same as the total amount of traffic outgoing from the same node.
- $f_{ij} \geq 0, \quad f_{ij}^{sd} \geq 0$ **range** constraints

The **objective function** is defined as:

$$min \ \max f_{ij} \tag{1.1}$$

hence, the goal is to optimize the network topology by minimizing the max flow passing from any link i-j.

Note that, since the task is to work with LP (linear programming) problems, the objective function is not properly written, indeed, the maximization function is not a linear one. In order to by-pass this problem, the big M trick was used: a new variable F was defined and, instead of minimizing the max flow ($\max_{ij} f_{ij}$), the new objective function became

$$min \ F$$

A pair of new constraints on F were needed:

$$F \geq f_{ij}$$

$$F \geq 0$$

Given the formulation, two different flow cases were considered: the LTD problem with allowed flow splitting and the other in which the splitting was not allowed.

1.2.1 Problem formulation: splitting vs not splitting flow

The formulation of the two problems remains exactly the same except for a constraint: in the case in which the flow splitting is allowed, the flow variable f_{ij}^{sd} can assume real values; in the other case, in which the splitting is not allowed, the same variable can assume just binary values. The choice of one problem with respect to the other, influences several characteristics of the traffic flow and network solution (e.g. the computational time, the gap between the optimal solution and the obtained solution...) as it will be explained later on.

1.3 Task 3

The LTD problem was modeled for each of the previous formulations (allowed and not-allowed flow splitting) through the usage of an optimization software: Xpress-IVE [1]. The exploited programming language readable from the optimizer was Mosel [1].

1.3.1 Task 3-a

The first task of this laboratory required to solve the LTD problem by setting the number of nodes equal to 12 and the number of transmitters and receivers (Δ) for each node equal to 4. The initial traffic matrix t^{sd} was considered uniform, that is, the traffic sent from any source to any destination was expressed by a random variable in a certain range. In this case the uniform traffic matrix was defined as $t^{sd} = Uniform[6, 12]$.
Moreover, the maximum computational time was set to three minutes.
Under these conditions, the results of the first task are represented in Figure 1.1.

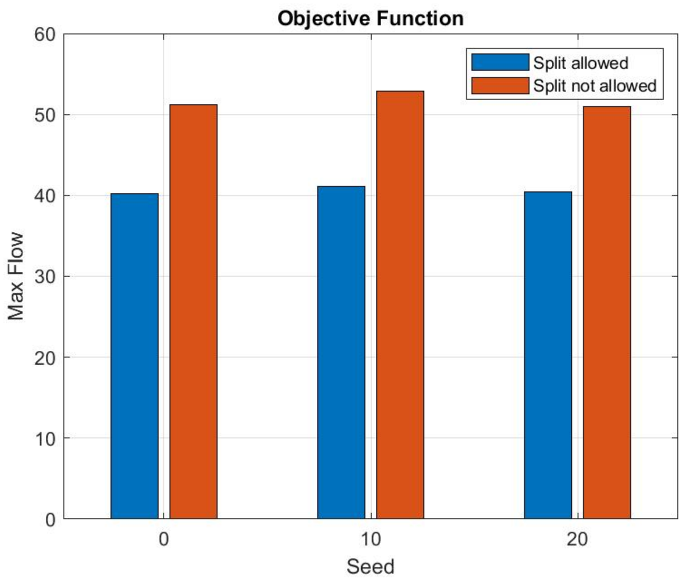


Figure 1.1: Objective function of the LTD problem computed with 3 different seeds

Three different traffic matrices were considered by choosing different seeds in the model. The behaviour of the objective function is always the same: the solution is larger when the flow splitting is not allowed. Since an extra integrality constraint is added, the complexity of the problem increases when the flow splitting is not allowed and the best solution becomes larger.

1.3.2 Task 3-b

The goal of the second task was to observe how the gap and the computational time varied according to the number of nodes. Both these values are automatically computed when the model runs on the Xpress optimizer, which evaluates the best bound of the solution and computes the gap between the obtained solution and the best bound. Only the case of allowed flow splitting was considered. The number of transmitters and receivers (Δ) was kept to 4 while the number of nodes (N) took the following values: 8, 16 and 24. As in the previous task, different traffic matrices were generated according to a uniform distribution between 6 and 12.
In Figure 1.2 the average, minimum and maximum gap between the obtained solutions

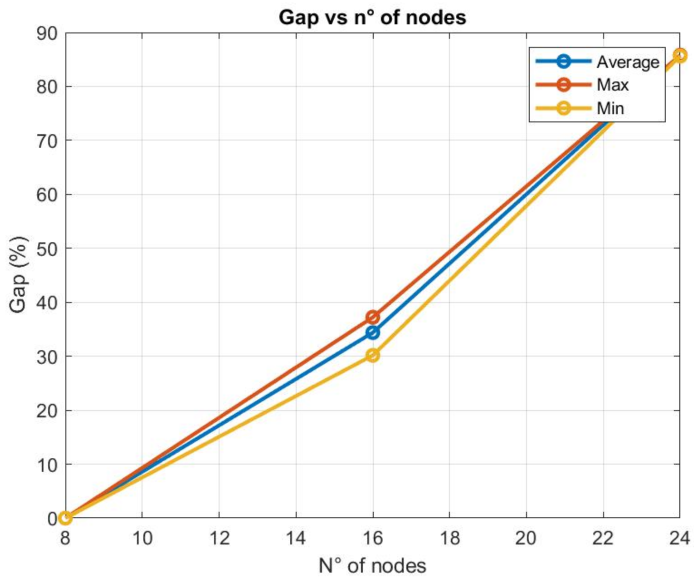


Figure 1.2: Average, minimum and maximum gap computed for the three different sizes of the network.

and the best bound are evaluated, considering three different traffic matrices at a time for each network size (number of nodes).
When N=8, the performances of the model are very accurate since the gaps are zero, and hence, the obtained solution is the optimal one. Also the computational times, represented in Figure 1.3, are very close to 0, which means that, with this configuration, the model finds an optimal solution in a reasonable amount of time.
Things change by doubling the size of the network (N = 16). With this new configuration, the model is not able anymore to give the best performance. The average gap increases up to the 34.38 % and the computational time required reaches the maximum established (three minutes). In the last case, with N = 24, the results are even worse. The average gap increases up to 85.69 % while the computational time remains the maximum established. From these results it is possible to deal with the time complexity of an NP-hard problem. As long as the parameters of the network are kept quite small, the solution is computed in reasonable amount of time. But, once the number of nodes are doubled, the time required to find the optimal solution dramatically increases.

In real cases, the number of parameters inside the model might be much higher than the numbers used in this lab. This is why, most of the times, a heuristic solution is preferred to the optimal one.

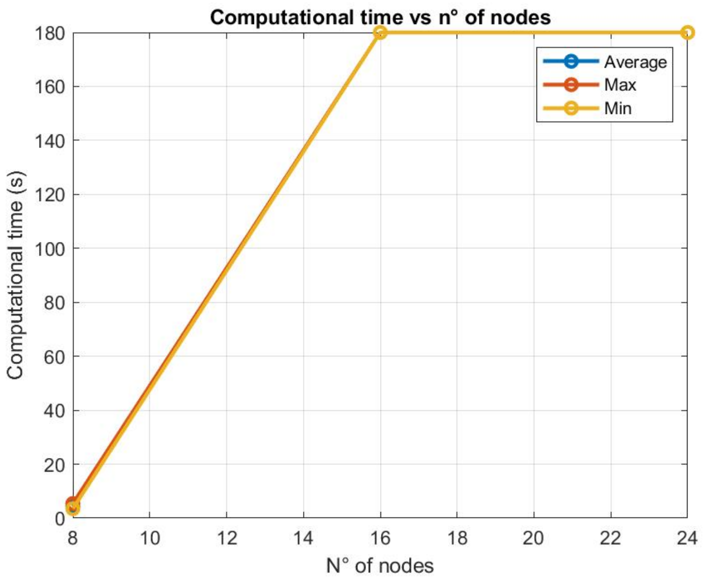


Figure 1.3: Average, minimum and maximum computational time computed for the three different sizes of the network.

As it can be seen in Figure 1.3, only when the number of nodes is equal to 8 the solution is found in few seconds; in the other cases, the algorithm is stopped by the time-out, and hence, it does not reach the optimal solution.

1.3.3 Task 3-c

The third task was slightly different from the second one. Similarly to the previous one, different traffic matrices were generated according to a uniform distribution between 6 and 12, but this time the number of nodes was kept to 10 while the number of transmitters and receivers (Δ) took the following values: 1, 2 and 4. According to this values, the **gap** and the **computational time** (their average, minimum and maximum values) were computed and plotted in the figures below (fig. 1.4 and fig.1.5).

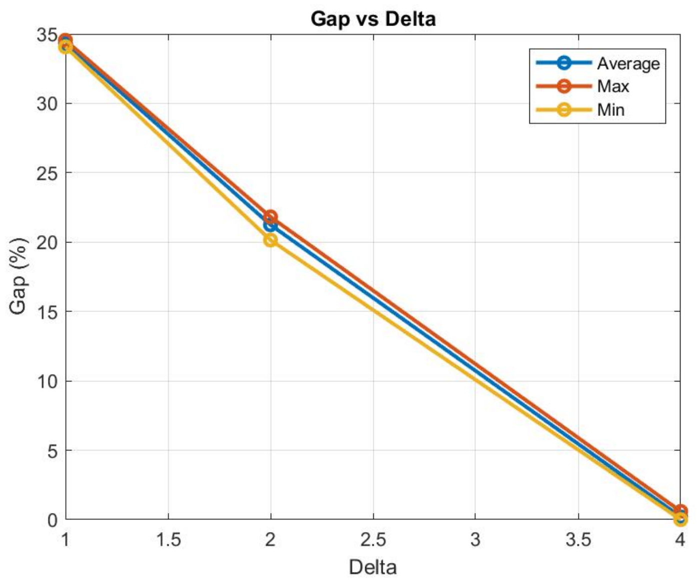


Figure 1.4: Average, minimum and maximum gap computed for the three different numbers of transmitters and receivers (Δ).

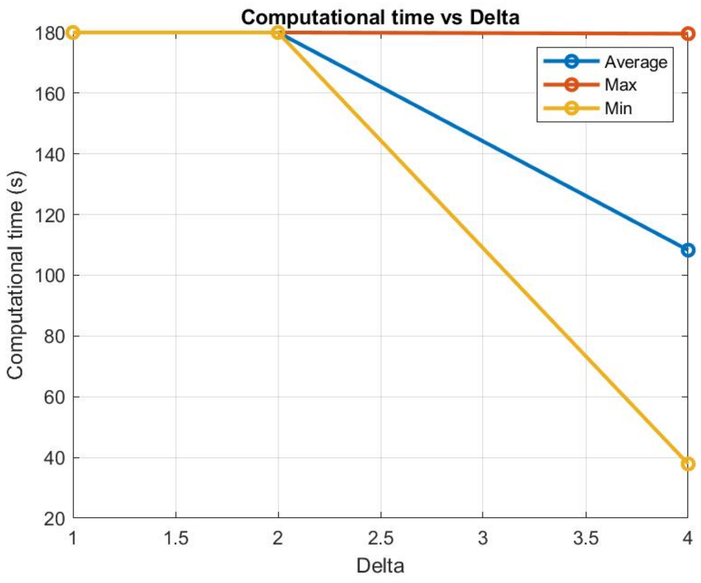


Figure 1.5: Average, minimum and maximum computational time computed for the three different numbers of transmitters and receivers (Δ).

The behaviour of the model is exactly the opposite with respect to the previous case (1.3.2). The quality of the solution increases as soon as the maximum number of transmitters and receivers for each node increases. When $\Delta = 1$ the average gap is 34.33 % and the computational time is the maximum established (also in this case equal to 3 minutes). When Δ is increased to two, the gap becomes lower (around 21.25 %) but the solution is still not optimal. The best performance is achieved when the number of transmitters and receivers for each node is set to four. The computational time starts to become less than the maximum established and the gap is very close to zero (around 0.19 %). This behaviour can be explained with the fact that, as the Δ increases, the constraints on the maximum number of links per node are more "relaxed", hence, the complexity of the model decreases.

1.4 Task 4

In the Task 4, a restriction on the topology was made. The restriction had the following parameters:

- Fixed square grid topology (also called Manhattan topology);
- number of nodes $N = 16$;
- number of transmitters and receivers per nodes $\Delta = 4$;
- traffic matrix uniformly distributed $t^{sd} = Uniform[1, 10]$;
- splitting allowed;

In order to obtain the square grid topology, a specific matrix of indicators (b_{ij}) was manually set up. Hence, the first different approach with respect to previous tasks, is that b_{ij} is no more a variable but it takes precise values defined a priori.

The initialization of such a matrix can be performed in different ways. A 4×4 square grid was created like the one shown in figure fig. 1.6. The horizontal links are built by connecting together one-hop distant nodes in the same row, while, for the the rightmost nodes, links with the leftmost side are created (indeed, each node must respect the node degree constraint). The same reasoning stands for the vertical links. Each time that a link is created between i and j, b_{ij} is set to 1.

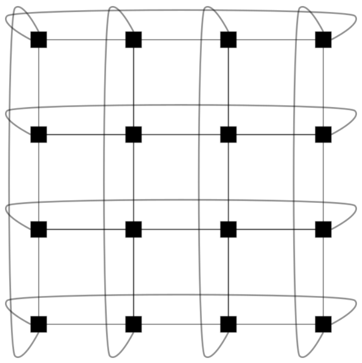


Figure 1.6: Grid topology of a network of 16 nodes which are represented by black squares. Every node have exactly four receivers and four transmitters links. Note that the links are bidirectional, hence only one link for both directions is draw.

seed	0	10	20
max flow	48.74	45.95	45.28
gap (%)	0	0	0
time (s)	0.9	0.8	0.8

Table 1.1: Results obtained using $t^{sd} = Uniform[1 - 10]$

seed	0	10	20
max flow	72.16	73,30	72,85
gap (%)	0	0	0
time (s)	0.9	0.8	0.8

Table 1.2: Results obtained using $t^{sd} = Uniform[6 - 12]$

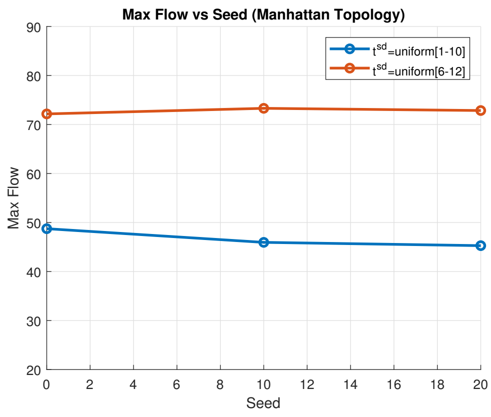


Figure 1.7: Maximum flow with Manhattan topology

The results of this model are reported in tables table 1.1, table 1.2 and fig. 1.7.

From these results, it can be noticed that the value of the seed does not affect the average behaviour of the solution.

More importantly, it can be seen that finally the results are very positive; the grid topology has some advantages in terms of computational time (less than 1 second with respect to three minutes seen in section 1.3) and in terms of gap, indeed the optimal solution is always reached (the gap is always 0).

As already mentioned, this can be explained by the characteristic of the topology, which is fixed by building the matrix b_{ij}. The complexity of the problem decreases because the model has not to create the topology, hence there are fewer variables.

1.5 Aggregate flow

In this task, it was asked to consider aggregate flow in the case in which flow splitting is allowed. The expression "aggregate flow" means that a set of flows is treated as if different flows are all part of a single flow.

In this case, the flow variable f_{ij}^{sd} considered before, becomes f_{ij}^{S}. Indeed, f_{ij}^{sd} is usually referred as dis-aggregated flow formulation [2] because each pair of s-d can be thought as a commodity. In the aggregated-flow problem, since the commodity is identified by the single sources, the formulation is simplified because it takes into account fewer variables and constraints.

1.5.1 Aggregate flow formulation

The formulation of the problem is similar to the one made in the previous task but easier.
Inputs:

- $t^s = \sum_d t^{sd}$, instead of using matrix $[t^{sd}]$, in order to aggregate flow it is convenient to deal only with the traffic generated at each source node, independently from the destination.

Variables:

- f_{ij}^s, portion of t^s flowing on the lightpath from node i to node j .
- $f_{ij} = \sum_s f_{ij}^s$, total traffic flowing on the lightpath from node i to node j.
- $b_{ij} = \{0, 1\}$, the topology indicator (same as before).
- $f_{max} = max\{f_{i,j}\} \quad \forall i, j$, the maximum amount of traffic flowing on any lightpath included in the logical topology.

All the variables take real values, except for the topology indicator that is binary (as specified above).
Objective function:

$$min f_{max}$$

Constraints:

- $\sum_j b_{ij} \leq \delta_i^{Tx}$, **outgoing degree constraint**
- $\sum_j b_{ji} \leq \delta_i^{Rx}$, **incoming degree constraint**
- $f_{ij} = \sum_s f_{ij}^s \quad \forall i, j$, total flow on the lightpath
- $\sum_j f_{sj}^s = t^s \quad \forall s$, the total amount of traffic outgoing from the source s must be t^s.
- $\sum_j f_{ij}^s - \sum_j f_{ji}^s = -t^{si} \quad \forall s, i \quad with \quad i \neq s$, (otherwise it would be equal to t^s); this is known as **transition equation**: the total amount of traffic incoming in a node is the same as the total amount of traffic outgoing from the same node.

- $f_{ij} \geq 0, \quad f_{ij}^s \geq 0, \quad f_{max} \geq 0, \quad f_{ij}^s \leq b_{ij} t^s \quad \forall i, j, s$ **range constraints**
- $f_{max} \geq f_{ij} \quad \forall i, j$

1.6 Unbalanced traffic

The last task of Lab 1 required to repeat Task 3-b, considering this time an unbalanced traffic matrix, that is, a network in which some nodes exchange a large amount of traffic while a small amount is exchanged by the others. In this particular case, some nodes exchange a random traffic uniformly distributed between 15 and 30, while some others a random traffic uniformly distributed between 1 and 3. An example of unbalanced traffic matrix, considering a network with 4 nodes, is reported below.

$$T_{sd} = \begin{bmatrix} 0 & 21.764 & 2.094 & 1.593 \\ 26.170 & 0 & 1.378 & 2.373 \\ 1.367 & 1.737 & 0 & 24.384 \\ 2.560 & 1.162 & 28.940 & 0 \end{bmatrix}$$

As in previous cases, the diagonal entries of the traffic matrix must be zero (no traffic exchanged between a node and itself). The first and fourth quadrant include the high-traffic nodes while a low traffic is exchanged between nodes in the second and third quadrant.
Exactly as in task 3-b, the goal is to observe how the gap and the computational time vary according to the number of nodes. The number of transmitters and receivers per node(Δ) is set to four and the number of nodes takes the following values: 8, 16 and 24. Only the case of flow splitting was considered.
In Figure1.8 the average, minimum and maximum gap between the obtained solution and the best bound are evaluated considering three different seeds for each network size.

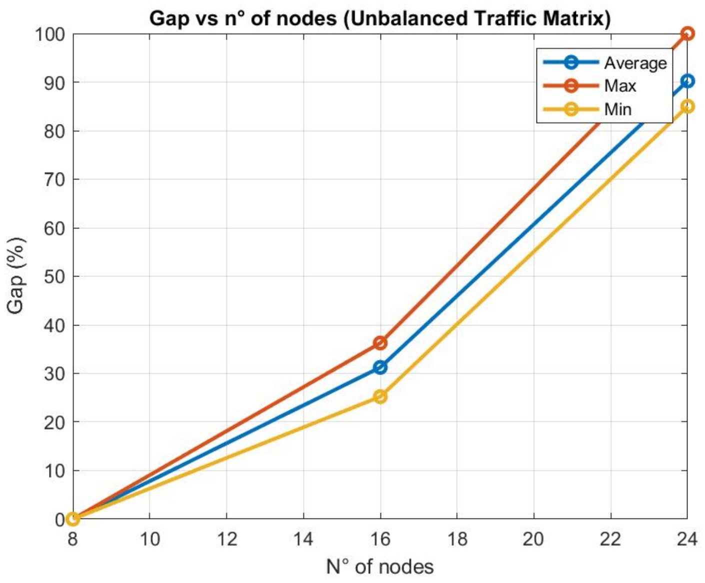


Figure 1.8: Average, minimum and maximum gap computed for the three different sizes of the network considering an unbalanced traffic mat.

From the plots in Figures 1.8 and 1.9 it appears that the resolution of a model in which the traffic matrix is unbalanced is more complex with respect to a model in which, instead, the traffic matrix has a uniform distribution. When N = 8, the model

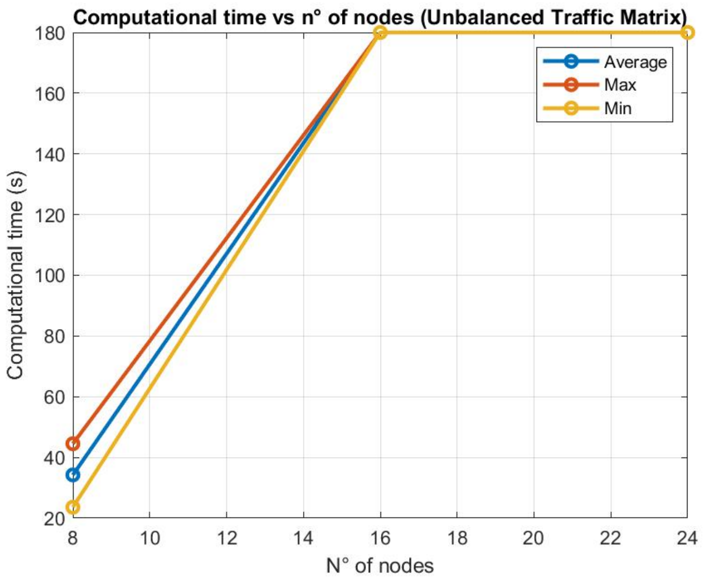


Figure 1.9: Average, minimum and maximum computational time computed for the three different sizes of the network considering an unbalanced traffic matrix.

is, on average, optimally solved in 30-40 seconds, differently from Figure 1.3 where the computational time was of the order of few seconds. This is the first evidence of the complexity behind the model proposed in this task. As soon as the network size increases, the model is solved in the maximum time established (3 minutes). As a consequence, the gap increases, assuming similar values to the ones showed in Figure 1.3. It is important to highlight the specific situation where N = 24 in which the gap assumes value 100% (Figure 1.8). This means that, according to the seed, the model could even be unable to find one single approximated solution. The numerical results are reported in Table 1.3.

Nodes	Seed	Max Flow	Gap (%)	Computational Time (s)
8	0	24.44	0	34.6
8	10	22.71	0	44.5
8	20	23.55	0	23.6
16	0	78.81	32.26	180
16	10	81.71	36.27	180
16	20	76.98	25.21	180
24	0	/	100	180
24	10	140.02	85.67	180
24	20	139.29	85.24	180

Table 1.3: Results obtained from the model with an unbalanced traffic matrix.

1.7 Last comments

The aim is just to summarize what it has already been said previously. What increases the complexity of a network is the number of nodes, the connectivity degree per node and the topology. More in particular, the more nodes are in the networks, the more the system is complex to be managed and, in a given time, the gap between the optimal solution and the solution gained by the compiler is higher; the less is the degree per node, the more traffic passes from those links, and, again, the higher is the gap (moreover, this could correspond to a situation in which the network is not completely connected); last but not the least, the complexity of a network can be drastically reduced by using a fixed topology such as the grid one.

Chapter 2

Lab 2: Greedy heuristic Algorithm Implementation

2.1 Introduction

In this lab session, it was asked to formalize a greedy heuristic algorithm in order to design a network topology. The general idea of a greedy algorithm, is to solve the designing problem and to output a solution which is not the optimal one, but it is good enough in order to be later processed and optimized with other algorithms. Hence, what it is obtained in this lab session, is a **sub-optimal** logical topology design.

2.2 Greedy algorithm: SCOM

The greedy algorithm chosen is "Source Copy Multicast Algorithm" (SCOM). It is a simple heuristic algorithm that allows to create a "virtual" sub-optimal logical topology given only the traffic matrix $[t^{sd}]$ and the maximum number of transmitters and receivers per node Δ.
The algorithm aims to create a virtual topology by adding lightpaths between nodes one by one. At the beginning there are N nodes without any link formed yet.
The **inputs**:

- Traffic matrix $[t^{sd}]$.
- Maximum number of transmitters per node: δ^{tx}.
- Maximum number of receivers per node: δ^{rx}.
- Current in and out degree for any node i: $deg^{T_x(i)}$ and $deg^{R_x(i)}$.

Step by step, the links are added following the pseudocode below.

Algorithm 1 SCOM algorithm

```
$b_{ij} = 0$
$deg^{T_x(i)} = 0 \quad \forall i, j$
$deg^{R_x(i)} = 0 \quad \forall i, j$

**while** $max([t^{sd}]) > 0$ **do**
    $s*, d* = arg\,max_{sd}[t^{sd}]$
    $[t^{s*d*}] = -[t^{s*d*}]$
    **if** $deg(s*) < \delta^{tx}$ & $deg(d*) < \delta^{rx}$ **then**
        $deg(s*) + +;$
        $deg(d*) + +;$
        $b_{s*d*} = 1;$
    **end if**
**end while**
```

At the beginning, no link is present and all the nodes are isolated, hence, the nodes degree is zero as the topology indicator for each link.
From this point, the algorithm starts. At each iteration in the while loop, the algorithm searches for the two nodes that exchange the maximum amount of traffic, among the elements of the traffic matrix. That traffic value is then made negative, in order to get a different maximum at the next iteration. If the out-degree of the source and the in-degree of the sink are smaller than Δ, the node has not yet saturated, hence, the new link can be created, and the respective entry in the link matrix b_{ij} can be updated to 1. A link can not be created if the constraints do not allow it. The algorithm ends when all the entries of the traffic matrix are negative.

2.2.1 Routing Algorithm

Once the virtual topology is created, the traffic must be sent over the network in an efficient way. This is done by the routing algorithm. The implemented routing algorithm is based on the **shortest path**.

The inputs are:

- traffic matrix $[t^{sd}]$
- virtual logical topology of the network
- flow matrix f_{ij}, with all the entries equal to 0

Algorithm 2 Shortest Path

```
$f_{ij} = 0 \quad \forall i, j$ in Nodes
for s = 1:number of Nodes do
    for d = 1:number of Nodes do
        path = ShortestPath(s, d);
        for every link ij in path do
            $f_{ij} += [t^{sd}]$
        end for
    end for
end for
result = $max(f_{ij})$;
```

At the end of this algorithm, the traffic at the most loaded link i-j will be assigned as the maximum flow, hence, $max(f_{ij})$.

2.2.2 Testing phase

In order to test the greedy algorithm, two different cases were considered: the case in which, the traffic is uniform and the case in which the traffic is unbalanced. For both of them, the SCOM algorithm outputting the virtual topology is tested against an algorithm that outputs a random topology with the same number of arcs.

The comparison between the two is based on the maximum flow value, achieved once the routing part returns the solution. The value of the flow-max is estimated considering not just one value but several ones and picking the average number.

Uniform traffic scenario

The first test is done by taking into account uniform traffic, modelled as a uniform random variable in the range [0:4]. Different values of number of nodes and node degree were considered and the corresponding maximum flow was plotted. This is shown in the Figures below (fig. 2.1, 2.2, 2.3).

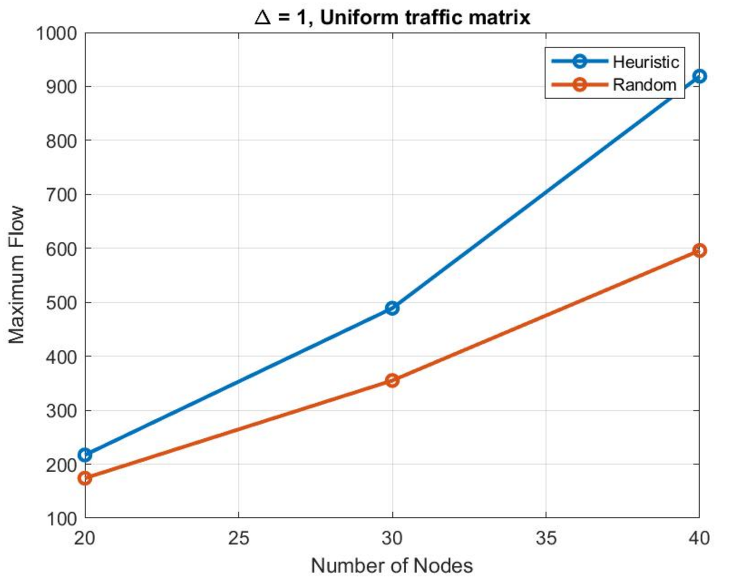


Figure 2.1: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), evaluated with node degree equal to 1 for different network size: N=20,30,40.

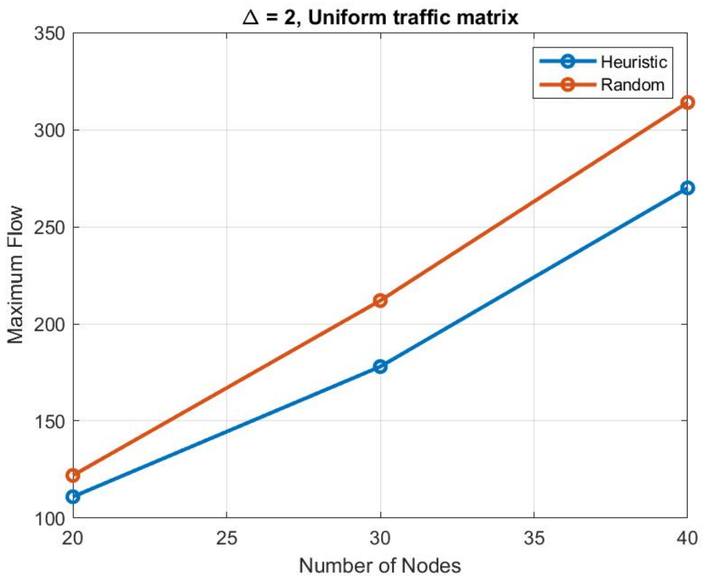


Figure 2.2: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), evaluated with node degree equal to 2 for different network size: N=20,30,40.

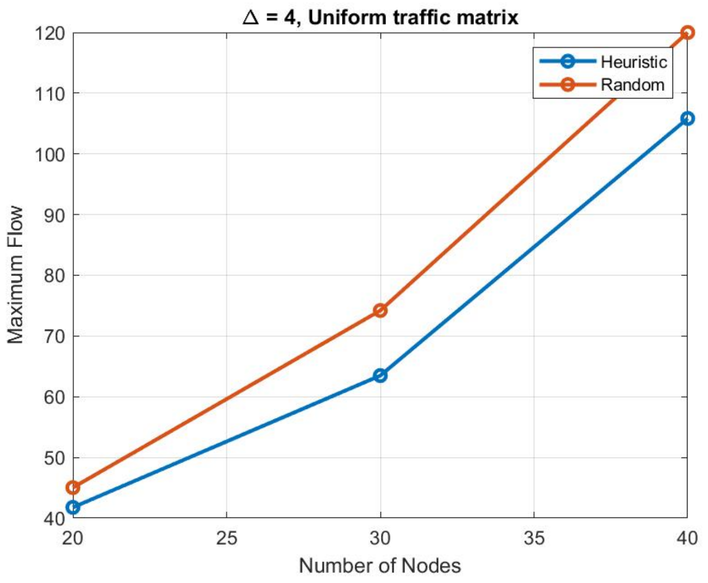
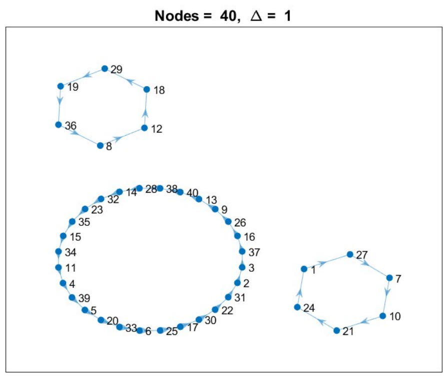
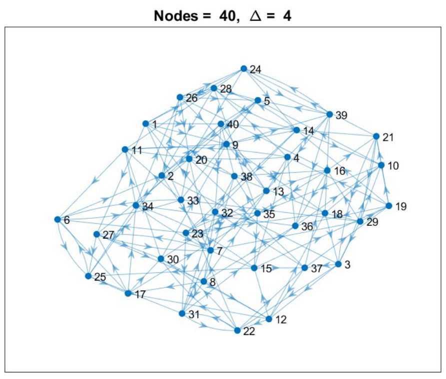


Figure 2.3: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), evaluated with node degree equal to 4 for different network size: N=20,30,40.

As it can be noticed, the topology created through SCOM (called greedy-topology) behaves always better than the one created by the randomly-generated topology. This is true for every network size for node degree 2 and 4. Indeed, in figures 2.2 and 2.3, the maximum flow given by the greedy-topology is always lower than the one generated by the random-topology. This means that the the greedy-topology ensures improved performances.
However, this does not stand for the network with node degree 1. By looking at figure 2.1, this time, the random-topology gives the best results of maximum flow. This must not be confuse with the fact that the random-topology is the best one. What happens in this case, is that with $\Delta = 1$ the class of topology obtained is a ring, however, not just a ring is obtained but more than 1, meaning that the network is disconnected (Figure 2.4). Hence, the results are not reliable.
In the end, it can be notice that, the more the network size increases, the bigger is the flow max, because the traffic is more intense; the bigger is the node degree, the better is the value of flow max. This actually makes sense, indeed, the more links are present, the more the traffic is split, so that the load on each link is lower.



(a)



(b)

Figure 2.4: Topology created with N=40 nodes and: node degree 1 in a and node degree 4 in b. Figure a shows that for such node degree, the topology is the ring, actually multiple rings, in which some smaller rings are created but they are not connected with the biggest one, resulting isolated. This depends on the type of used algorithm. The node isolation phenomena is even worst with less node in the network.

Unbalanced traffic

The same experiment is repeated in the case of unbalanced traffic: 90% of the network has low traffic and the remaining 10% has high traffic. The low traffic is modelled as a uniform random variable in the range [0, 3], while the high traffic is modelled in the range [5,15]. In the same way of the uniform traffic, different combinations of number of nodes and node degree were considered. The maximum flow for both topologies (random and greedy one) was again computed and plotted in Figures below (fig. 2.5, 2.6, 2.7) to make comparisons about performances.

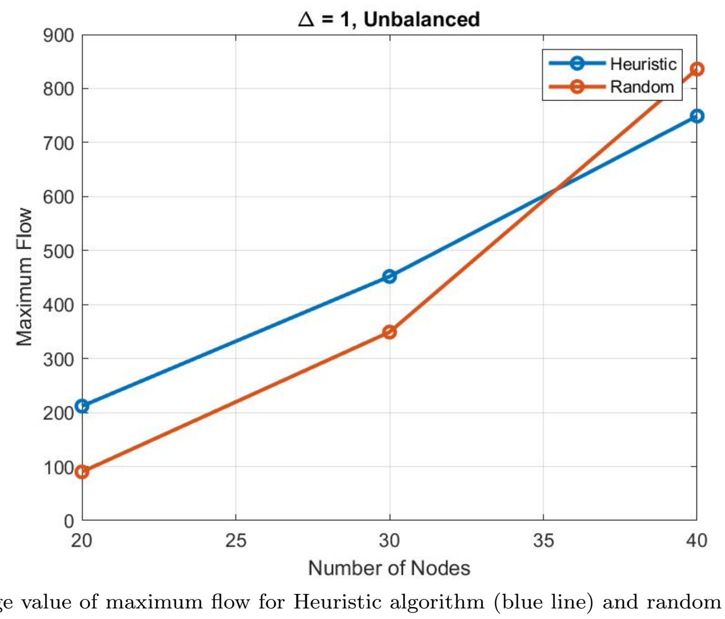


Figure 2.5: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), with unbalanced traffic, evaluated with node degree equal to 1 for different network size: N=20,30,40.

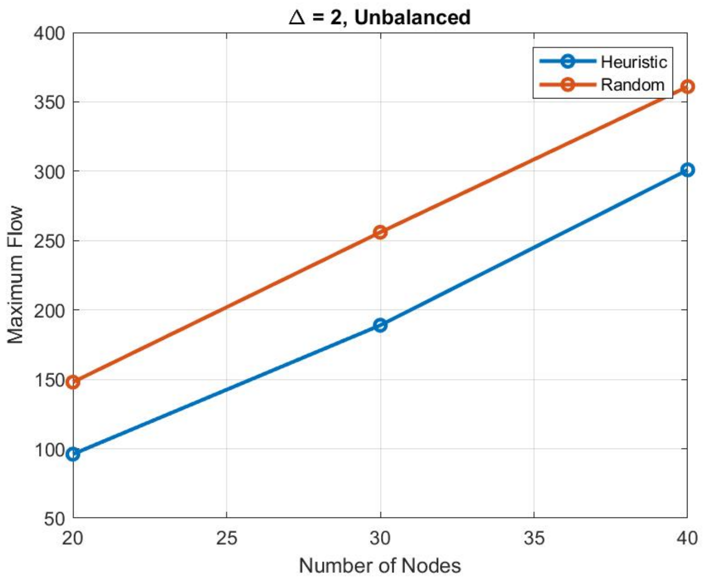


Figure 2.6: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), with unbalanced traffic, evaluated with node degree equal to 2 for different network size: N=20,30,40.

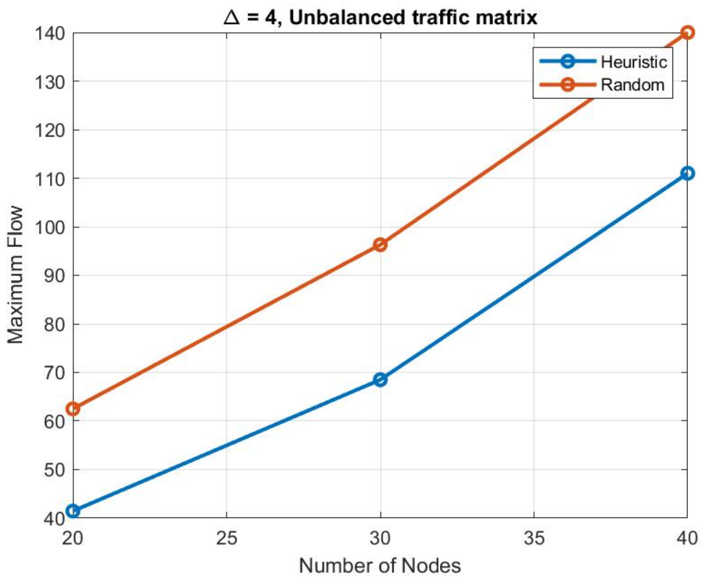


Figure 2.7: Average value of maximum flow for Heuristic algorithm (blue line) and random algorithm (red line), with unbalanced traffic, evaluated with node degree equal to 4 for different network size: N=20,30,40.

The behaviour of the two topologies considering their maximum flow is exactly the same as before: the greedy-heuristic gives the best performance for node degree 2 and 4 but not for node degree 1. Again, this means that this value is not reliable because the network is disconnected.

Also in this case, it can be noticed an increasing trend following the increasing network size and the fact that, the higher is the node degree, the lower is the max-flow value.
Moreover, comparing figure 2.2 with 2.6 (the uniform traffic and the unbalanced traffic with delta=2) and 2.3 with 2.7 (the uniform traffic and the unbalanced traffic with delta=4) the results show that, for small network size (N=20), the greedy-topology with uniform traffic has the same value of max flow as the unbalanced one; for greater nodes number however, even if for a small gap, the value of maximum flow is lower for the case of uniform traffic.

2.3 Manhattan Topology

The third task required to implement a greedy heuristic algorithm on a bidirectional Manhattan topology. This type of topology was already explained in section 1.4 of the previous lab with the only difference of an additional requirement which is: a smart positioning of nodes. In this kind of topology the nodes with higher traffic should be "smartly" placed close to each other in order to minimize the objective function. Basically, this is the same concept exploited in SCOM with an additional constraint due to the strict configuration of the topology.

The inputs of this algorithm are:

- Number of total nodes ($Nodes$).
- Traffic matrix $[t^{sd}]$.
- A Manhattan bidirectional topology.
- A list containing the nodes IDs that are already been placed in the topology, called *names*. At the beginning the list *names* must be empty.

The smart positioning algorithm can be divided into four sub-algorithms, since it has to deal with four possible scenarios.

Algorithm 3 Smart Positioning of the Nodes (First scenario)

```
while $max([t^{sd}] > 0$ do
  s*, d* = $arg\,max_{sd}[t^{sd}]$;
  if s* not in names & d* not in names then
    $a = randint(1, Nodes)$;
    if $names(a) == null$ then
      $names(a) = s*$;
    end if
    while names(b) <> Null do
      b = closest_neighbour(a);
    end while
    names(b) = d*;
    $[t^{s*d*}] = -[t^{s*d*}]$;
  end if
  ...
  ...
end while
```

In the first case, the two current nodes exchanging the largest amount of traffic have not been located in the topology yet, so their IDs do not appear in the list *names*. When this happens, a random topology node (with index a) is picked and, if it was never picked before, it is assigned to node s. It means that node s will be appended to list *names* at index a. Concerning node d, it should be as close as possible to node s. First of all, the neighbours of a must be sorted, from the nearest to the furthest. If the index of the closest one is already in list *names*, then it is necessary to look for the next one, and so on. When an unassigned topology node is found, its index will be associated with node d.

Algorithm 4 Smart Positioning of the Nodes (Second/Third scenario)

```
while $max[t^{sd}] > 0$ do
    ...
    ...
    if s* is in names & d* not in names then
        b = closest_neighbour(index(names, s));
        while names(b) <> Null do
            b = closest_neighbour(index(names(s)));
        end while
        names(b) = d*;
        $[t^{s^*d^*}] = -[t^{s^*d^*}]$;
    end if
    ...
    ...
end while
```

Algorithm 4 represents two similar scenarios. In fact, it can happen either that node s* is in list *names* and d* is not or the exact opposite. In both scenarios the algorithm has always to do the same thing: starting from the allocated node in the topology (s* or d*), it has to place the other one as close as possible to it by looking into its sorted neighbours list, exactly as the last part of Algorithm 3.
The last scenario is the one in which both the nodes have already been placed in the topology. In this case the algorithm can do nothing.

Algorithm 5 Smart Positioning of the Nodes (Fourth scenario)

```
while $max[t^{sd}] > 0$ do
    ...
    ...
    if s* is in names & d* is in names then
        $[t^{s^*d^*}] = -[t^{s^*d^*}]$;
    end if
end while
```

2.3.1 Results and comments

The Algorithm is run for different number of nodes obtaining fig. 2.8 which shows only a small gap between the Smart Positioning and Random generated Manhattan, further improvement of the algorithm can be done using other heuristic methods or changing the routing algorithm. The shortest path algorithm does not take into account the overlapping of several flows on the same link.

The alternative to the shortest path is to choose the less congested path but it might increase the consumption of the Power. This is what the Green Networking aims to optimize.

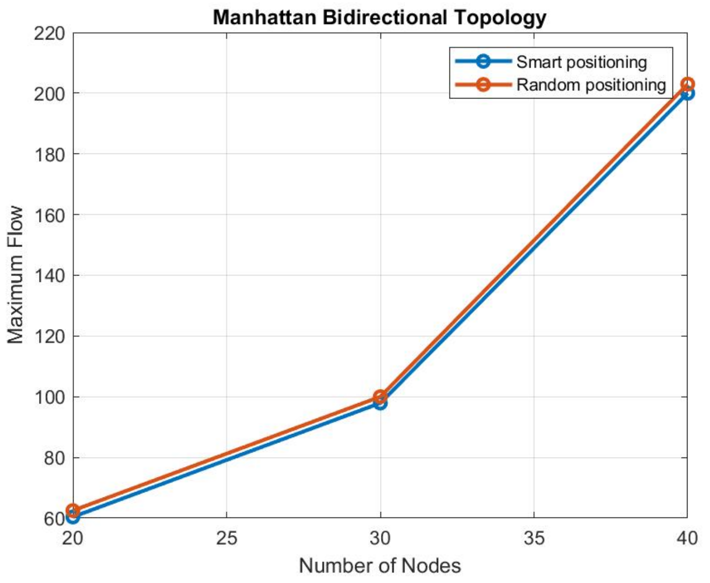


Figure 2.8: Smart Positioning of nodes

2.4 Local Search

In order to improve the performances achieved with the greedy-heuristic algorithm explained in the previous task, a local search can be implemented. Local search means to explore a space of feasible solutions that are close to the current one, in order to find a more optimal output of the problem. The space of neighbour solutions can be discovered by applying a perturbation to the current topology.
In the following algorithm, the perturbation includes to switch the position of two random nodes and to observe how the maximum flow is affected by it. If the new solution is better than the current one, then the topology is directly changed and the next swaps of nodes are applied on the new topology. These steps are repeated for a certain number of iterations.

The input of this algorithm are:

- Current topology obtained with the greedy-heuristic algorithm.
- The total number of nodes, *Nodes*.

Algorithm 6 Local Search

```
optimal_solution = sum([$t^{sd}$]);
for $N = 1 : iterations$ do
    $f_{ij} = zeros(Nodes, Nodes)$
    {
    ...
    Shortest Path (algorithm 2)
    ...
    }
    if result < optimal_solution then
        optimal_solution = result;
    else
        swap(node2, node1);
    end if
    node1 = randint(1, Nodes);
    node2 = randint(1, Nodes);
    swap(node1, node2);
end for
```

At the beginning, the optimal solution is set as the sum of the traffic exchanged by any node (worst possible case). In the for loop, the flow matrix is reset in order to perform the Shortest path algorithm (2). After that, if the new solution is better than the optimal one, the current topology is kept and the optimal solution becomes the current one. The next neighbour solution is found by switching two random nodes in the topology. At the next iteration, if the new solution is worse than the optimal one, the two nodes previously switched are put back in the earlier positions.

2.4.1 Results from Local Search

A local search in this case allows to increase the gap with the maximum flow obtained through a random Manhattan topology. In fact, while with a greedy algorithm the difference between the two flows is around 2, as shown in fig. 2.8 , thanks to the local search the improvement is much larger.

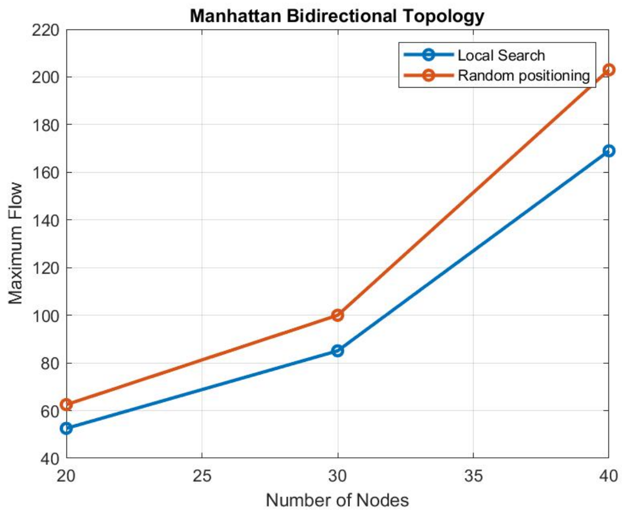


Figure 2.9: Average value of maximum flows in a Manhattan topology.

In the figure Fig. 2.9, the blue line represents the results obtained when the nodes are smartly placed and the route is made through a local search algorithm. The red line represents the results obtained when the nodes are randomly positioned and the route is made through the shortest path algorithm.

From a comparison between figure 2.8 and figure 2.9, it is clear that in the last case the results are much better. Not only the maximum flow obtained with a local search is always smaller than the solution generated from a random positioning of the nodes, but the difference between the two solutions is much greater. While the average difference between the two maximum flows in the previous task was around 2, now it is increased up to 20.

As already said before, a greedy algorithm allows to find a feasible and good solution, but not the optimal one. In order to determine a better solution, a local search algorithm has to be implemented, starting from the result derived from the greedy algorithm. By looking at the space of neighbours solutions, it is possible to achieve better performances, but the drawback is that the number of iterations needed might be extremely large.

Chapter 3

Lab 3: Green Networking Problem

3.1 Introduction

The green networking problem arises to solve the problem of power consumption in the network domain. Energy is becoming the main issue for the future and its careless consumption is causing dramatic climate changes. Moreover, the consumptions might double in the next decade, that is why it is very important to find efficient ways to take care of this problem in all the sectors, ICT included.

The goal is thus to reduce the power consumption of a network by reducing the power consumption of the devices composing it, such as routers (by reducing CPU, memory..) and links (by designing more efficient transmission technique and lower power transmission).

The reduction of the whole network power turns out to be a design and management problem. The main idea to cope with this problem is to use the TTO (Turn Things Off) solution, by turning in **sleeping modes** some portions of the network for a certain period. The main issues at this point become to guarantee the connectivity of the network, the continuity of the service and the updating of the routing, together with the distribution of state informations.

3.2 Problem Formulation

What was asked in this lab was to make the formulation of the green network problem considering the optimal design problem for an energy efficient network.
The problem was formulated by taking into account that the power dissipated by a device is the sum of two terms:

- a load independent term
- a term linearly dependent on the device load

Indeed, what happens in general is that, even if a device is not working in a certain moment, it consumes energy; when instead it has traffic to deliver, the consumption of energy linearly depends on the load. This means that the higher is the load, the higher is the power consumption. The load independent term is related to the set-up cost which is fixed, the other term instead is related to the marginal cost, hence the cost per unit of power.

The formulation is the following.
Input:

- $[t^{sd}]$, traffic matrix;
- b_{ij}, topology indicator;
- Pr_i, power consumed by router i (fixed cost);
- P_{Lij}, power consumed by the link i-j (fixed cost);
- Qr_i, cost of router i per unit of load;
- Q_{Lij}, cost of link i-j per unit of load;
- C_{ij}, capacity of link i-j;
- α, security coefficient (to avoid over provisioning);
- Δ degree connectivity value;

What is different with respect to the previous input formulation is the presence of the cost (in terms of power consumption), the capacity of the links and the α term. This one is needed in order to avoid the congestion of the link; say, if the capacity of a link is 10 Gb/s, and $\alpha = 0.7$ just the 70% of the total capacity is actually available. In this lab, it was set $\alpha = 0.7$, $\Delta = 4$, $C_{ij} = M \cdot N$ (where M is the half of the maximum traffic exchanged and N is the number of nodes in the network), $Pr_i = 1000W$, $P_{Lij} = 600W$, $Qr_i = Q_{Lij} = 0.087$ (5°of slope).
Variables:

- $x_i = \{0, 1\}$ router state indicator, it is 1 if router i is active, 0 otherwise;
- $y_{ij} = \{0, 1\}$ link state indicator, it is 1 if link ij exists ($b_{ij} = 1$) and it is active;
- f_{ij}^{sd} as usual, portion of the traffic from s to d passing through link ij;
- f_{ij} load on arc i-j (total amount of traffic passing through arc ij);
- F_i load of router i;

What is new here is the router load variable, the link state variable and the router state variable, needed in order to be able to switch them off or on.
Constraints:

- $\sum_j f_{ij}^{sd} - \sum_j f_{ji}^{sd} = \begin{cases} t^{sd} & \text{if } i = s \\ -t^{sd} & \text{if } i = d \\ 0 & \text{if } i \neq s, d \end{cases} \quad \forall s, d, i$ flow conservation constraints;
- $f_{ij} = \sum_{sd} f_{ij}^{sd} \quad \forall i, j$
- $f_{ij}^{sd} \leq t^{sd} y_{ij} \quad \forall i, j$, the flow must be 0 when a link if a link is turned off;
- $y_{ij} \leq b_{ij}$, the arc i-j cannot be 1 if the link does not exists in the topology;
- $f_{ij} \leq \alpha \cdot c_{ij} \cdot y_{ij} \quad \forall i, j$, the flow cannot exceed the αpercentage of the link capacity;
- $F_i = \sum_j f_{ij} + \sum_j f_{ji} \quad \forall i$ the router load constraint;
- $f_{ij} \geq 0 \quad \forall i, j$
- $f_{ij}^{sd} \geq 0 \quad \forall i, j, s, d$
- $\sum_j y_{ij} + \sum_j y_{ji} \leq 2 \cdot \delta \cdot x_i \quad \forall i$, if the router i is off, all the links outgoing and incoming must be turned off.

Objective function:

$$min \ \sum_i P_{Ni} \cdot x_i + \sum_i Qr_i \cdot F_i + \sum_{ij} P_{Lij} \cdot y_{ij} + \sum_{ij} Q_{Lij} \cdot f_{ij} = min \ P_{TOT}$$

As shown in the objective function, the set up cost for each node or link is multiplied for the indicator of that element; the load dependent cost instead, is multiplied for the cost produced by the particular router (F_i) or link (f_{ij}).
The formulation was tested with values of costs (Power terms) similar to the real cases and for different time slot in a day. Indeed, the network traffic depends on the particular time of the day. The Figure 3.1 shows the amount of traffic in Gb/s exchanged in three different time slot: morning, afternoon and evening.

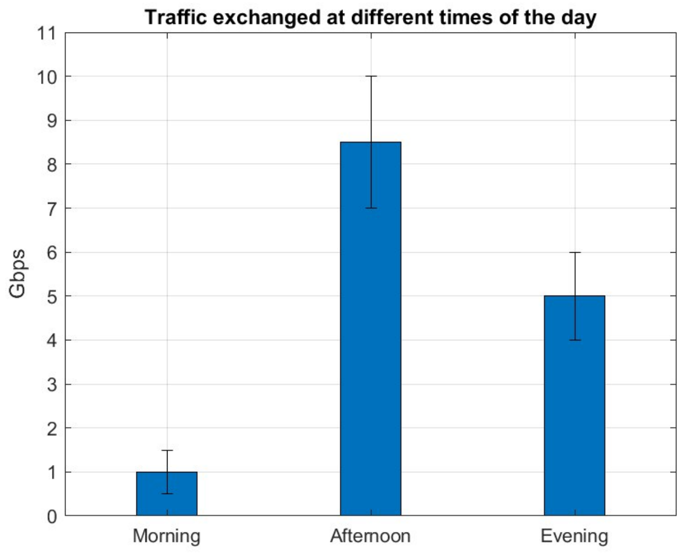


Figure 3.1: Amount of traffic in Gb/s exchanged in three different time slot: morning, afternoon and evening. The most o the traffic is exchanged in the afternoon with a value included in the range 7-10Gbps of traffic, range 0.5-1.5Gbps for the morning and range 4-6Gbps for the evening traffic demand.

The formulation was solved for different physical topologies, the Manhattan topology which is regular and a Realistic topology, which is irregular. The results are discussed in the following section.

3.3 Results with Manhattan topology

The Manhattan topology is exactly the same type of topology already seen in figure 1.6. Each node is placed in a grid structure and is connected with two neighbours along every direction (horizontal and vertical). This structure is well know to be regular, since each node has a precise position and all of them have the same topological properties. Moreover, a Manhattan topology is highly fault tolerant since many routes are available between any pair of nodes.
The Green Networking problem was applied on this structure to observe how many nodes and links would have been turned off. Three different situations were tested:

1. A 16 nodes Manhattan topology in which every node can be a source or a destination.

2. A 16 nodes Manhattan topology in which some of the nodes can be neither sources nor destinations (this type of nodes will be called "core" nodes).

3. A 20 nodes Manhattan topology in which some of the nodes can be neither sources nor destinations .

Furthermore, the network was tested under various conditions of traffic (represented in figure 3.1), in order to observe how the several elements of the topology were turned off to save as much energy as possible.

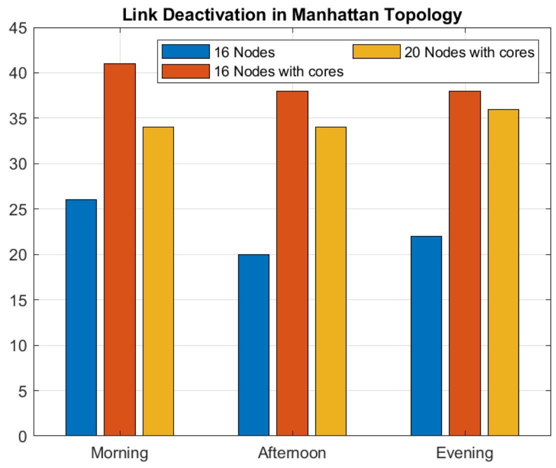


Figure 3.2: Number of deactivated links in the three types of Manhattan topology according to the time of the day.

Figure 3.2 shows how many links are deactivated according to the type of Manhattan topology and to the time of the day. The blue bar represents the 16 nodes Manhattan topology in which every node can be a source or a sink, the red bar is the 16 nodes topology with core nodes and the yellow bar is the 20 nodes topology with core nodes. As already said, the core nodes are vertexes of the topology whose only purpose is to route traffic. In both cases, the core nodes were the central four nodes of the grid. All three topologies display a similar behaviour: when the traffic exchanged between nodes is quite low, the number of deactivated links increases to save more energy. The networks prefer to route the traffic along longer paths instead of keeping active a higher number of links. When the traffic starts to increase, some of the links must be reactivated because of the capacity constraint. Between afternoon and evening there is not a big difference, but the links start to be deactivated again since the traffic becomes lower with respect to the afternoon traffic but still higher than the morning traffic.

When all the nodes of the structure can generate or receive traffic, the number of deactivated links is significantly lower. This happens because the flow constraints must be respected. If a node is a source or a sink, it can not be turned off. Given this property, when the topology has no core nodes, the number of deactivated nodes is always 0. For the other two cases instead, the number of deactivated nodes is constantly 4. This happens because in a regular grid topology, the paths between any pair of nodes are quite numerous. Hence, the network prefers to keep that four nodes off and to route the traffic along sub-optimal paths. Generally, the energy saved turning off a router is much larger than the power-saving obtained by a less loaded link.

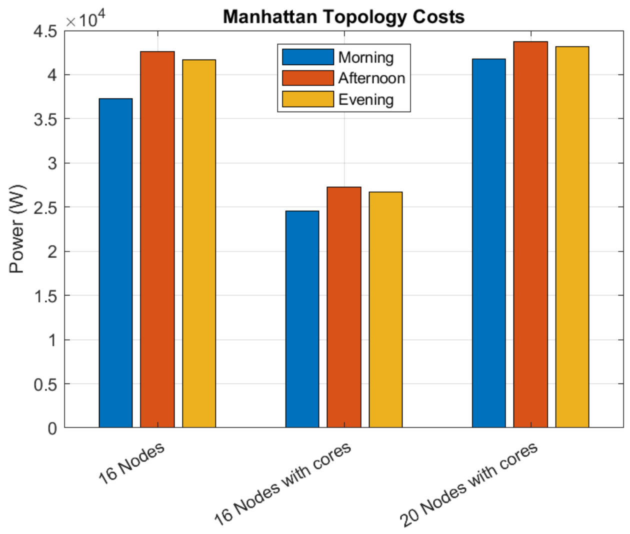


Figure 3.3: Power consumptions for Manhattan topology

Figure 3.3 represents the power consumption associated to each type of grid structure, according to the time of the day. The most power-effective topology, as expected, is the one with 16 nodes where core nodes are available. The power consumed by the other two grids is quite similar because in the topology with 16 nodes and no only-routing nodes, no router can be turned off. In the topology with 20 nodes and core nodes available, the number of links and nodes is higher, but four nodes are constantly turned off. This leads to a certain balance between the two topologies in terms of number of active nodes and links.

3.4 Results with a Realistic topology

Other than a regular topology, as seen in section 3.3, a more realistic topology, composed of "blue" and "black" nodes, is simulated. The fig. 3.4 shows that this topology is very close to a real scenario in which blue nodes generate traffic, then black nodes just rout them without generation of any additional traffic. This condition gives the possibility of "turning things off". The traffic load is divided in three time categories: morning with low traffic, afternoon with peak traffic and evening with a traffic smaller than the peak.

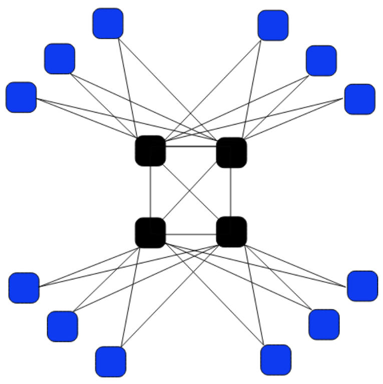
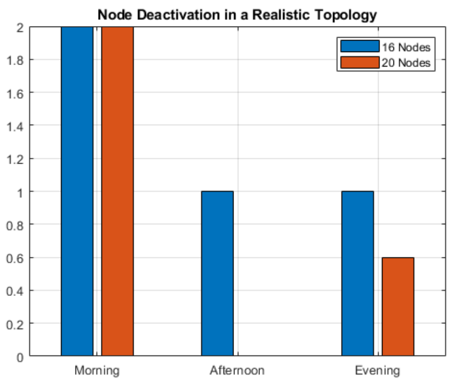


Figure 3.4: One blue node has 2 links with black nodes in order to have a redundant connection, thus making TTO possible

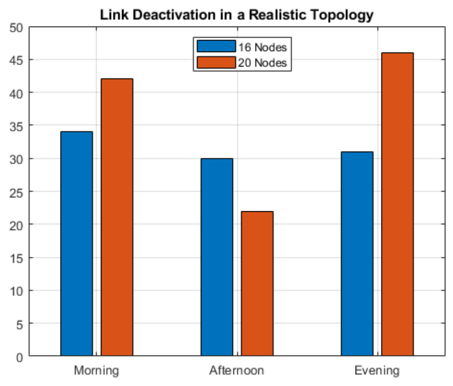
In fig. 3.4 there are 16 nodes. For the simulation of the same topology with N=20, other 4 external routers must be added to the topology. In order to save energy the objective function will turn off as many nodes and links as possible. By knowing the number of total links and nodes, and the number of active ones, it is possible to calculate the deactivated nodes with a simple subtraction.

The results are reported in fig. 3.5. They are an average obtained from different random seeds. The number of deactivated nodes (fig. 3.5a) is very low both for 16 nodes and for 20 nodes topology because only four nodes can be actually turned-off (which are the black ones and there are only 4 of them). In contrast fig. 3.5b displays a higher deactivation rate, around 25% and 60%, because the flow can be routed on different links in order to save energy. In both fig. 3.5a, fig. 3.5b the activation rate is higher during the afternoon. For example, at the peak traffic nearly all the routers are turned-on.

Finally the Power consumption fig. 3.6 is as expected, as it was already anticipated in fig. 3.5



(a) Node Deactivation



(b) Link Deactivation

Figure 3.5: The topology with Nodes = 16 has 60 links while for Nodes = 20 there are 76 total links

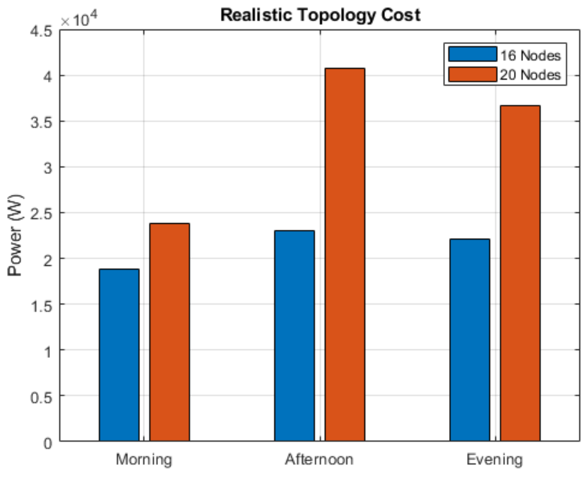


Figure 3.6: Power consumption comparison

3.5 Final comments

Comparing a regular topology such as the Manhattan (section 3.3) and a realistic topology (section 3.4), even if the network size represented in fig. 3.4 was not big enough (compared to an ISP network), some final conclusions can be drawn. The comparison can be done since the number of nodes is the same in both topologies as well as the average node degree. The overall power consumption in the grid topology, both with sixteen nodes and twenty nodes, is around 10% - 20% higher with respect to the realistic one, considering the Manhattan topology with core routers available. The results are very similar when the traffic exchanged is particularly high (during the afternoon periods). In a Manhattan topology there are many routes available between any pair of nodes. Hence, when the traffic increases, the core routers are kept "off". Instead, in the realistic topology, the black nodes (fig. 3.4) are all forced to be "on" since there are less routes available and the capacity constraints can not be satisfied. As already said in section 3.4, the core routers have a higher static power consumption. For this reason, the power consumed by the realistic topology sharply increases when all the black nodes are forced to be "on". This makes the results between the realistic and the Manhattan topology very similar in the case of a high traffic exchanged into the network.

Bibliography

[1] Fico xpress optimization website.

[2] R. Ramaswami and K. N. Sivarajan. Design of logical topologies for wavelength-routed optical networks. *IEEE J.Sel. A. Commun.*, 14(5):840–851, June 1996.