Final report on the Operational Research laboratories

Luca Gioacchini (s257076), Riccardo Sappa (s263241),
Daniele Russo (s266492), Francesco Guaiana (s267534)

A.A.2018/19

Contents

1 LTD Problem

The Logical Topology Design problem aims at determining a network topology in order to route a certain amount of flow within the network nodes by ensuring a minimum (or maximum) cost, such as the maximum traffic flow passing through a link.

1.1 LP Model: Full Mesh Topology. Flow Splitting Disabled

Linear Programming is a method that can find the optimum solution of a problem, such as the LTD one, by providing a set of inputs and variables. Their relationship is expressed by a model based on the problem characteristics.

In the case of the LTD problem, applied on a full mesh topology network with N nodes, the LP model is based on the traffic flow balance and nodes' connectivity constraints. Since the flow splitting is disabled, all the amount of traffic flowing from s to d is routed through only one path defined as a set of nodes.

Each node is characterized by a number of possible inputs (δ_I) and a number of possible outputs (δ_O). By assuming $\Delta = \delta_I = \delta_O$, this parameter can be modified analyzing the performances of optimization algorithm for the LTD problem.

1.1.1 Inputs/Outputs

$$\mathbf{T} \in \mathbb{R}^{NxN} \tag{1}$$

$$\mathbf{B} \in \mathbb{N}^{NxN} \qquad \text{with } b_{i,j} \in \{0,1\} \qquad \forall i,j \tag{2}$$

where $\mathbf{T}$ is the input traffic matrix. The $t_{s,d}$ entry contains the traffic flowing from node s, to node d. It is a hollow matrix[1]. $\mathbf{B}$ is the output indicator matrix, it contains information about the topology, since if the lightpath linking node i with node j is used, then $b_{i,j} = 1$, otherwise it is 0.

1.1.2 Variables

$$\mathbf{K} \in \mathbb{N}^{NxNxNxN} \qquad \text{with } k_{i,j}^{s,d} \in \{0,1\} \qquad \forall i,j,s,d \tag{3}$$

$$F_{max} \in \mathbb{R}^{+} \tag{4}$$

$$\mathbf{F} \in \mathbb{R}^{NxN} \qquad \text{with } f_{i,j} \in \mathbb{R}^{+} \qquad \forall i,j \tag{5}$$

$$\mathbf{FL} \in \mathbb{N}^{NxNxNxN} \qquad \text{with } f_{i,j}^{s,d} \in \mathbb{R}^{+} \qquad \forall i,j,s,d \tag{6}$$

where $\mathbf{K}$ is a specific flow indicator matrix. It is used to force the flow passing from a node exactly to another one, without splitting it. $\mathbf{FL}$ is the flow-specific matrix. Its element $f_{i,j}^{s,d}$ contains the portion of traffic generated from the source-node s, to the destination-node d, passing through the lightpath that links the *i-th* node to the *j-th* node. $\mathbf{F}$ is the flow matrix, it aggregates the flows passing through the lightpath *(i,j)* of all the s-d couples. F_{max} is the upper bound of the maximum flow passing through all the used lightpaths.

[1]The diagonal entries are equal to 0

1.1.3 Objective Function

$$minF_{max} \tag{7}$$

Since the LTD problem described by such a model is a non linear problem, the objective function is the result of the linearization process. In this way, instead of maximizing a large number of variables, the solution is found by minimizing an upper bound of them.

1.1.4 Constraints

Flow Relationships:

$$f_{i,j} \le F_{max}, \qquad \forall i,j \tag{8}$$

$$f_{i,j} = \sum_{s,d} f_{i,j}^{s,d}, \qquad \forall i,j \tag{9}$$

$$\sum_{j} f_{s,j}^{s,d} = t_{s,d} \cdot k_{i,j}^{s,d}, \qquad \forall s,d \tag{10}$$

Flow Conservation:

$$\sum_{j \neq s,d} f_{i,j}^{s,d} - \sum_{j \neq s,d} f_{j,i}^{s,d} = 0, \qquad \forall i \neq s,d \qquad \forall s,d \tag{11}$$

Flow Conservation at sources

$$\sum_{j} f_{s,j}^{s,d} = t_{s,d}, \qquad \forall s,d \tag{12}$$

$$\sum_{j} k_{s,j}^{s,d} = 1, \qquad \forall s,d \tag{13}$$

$$\sum_{j} f_{j,s}^{s,d} = 0, \qquad \forall s,d \tag{14}$$

Flow Conservation at destinations

$$\sum_{j} f_{j,d}^{s,d} = t_{s,d}, \qquad \forall s,d \tag{15}$$

$$\sum_{j} k_{j,d}^{s,d} = 1, \qquad \forall s,d \tag{16}$$

$$\sum_{j} f_{d,j}^{s,d} = 0, \qquad \forall s,d \tag{17}$$

Zero flow when $s = d$

$$f_{i,j}^{s,d} = 0, \qquad \forall i,j \qquad \forall s = d \tag{18}$$

Connectivity:

$$\sum_i b_{i,j} \le \Delta, \qquad \forall j \tag{19}$$

$$\sum_j b_{i,j} \le \Delta, \qquad \forall i \tag{20}$$

Feasibility:

$$f_{i,j}^{s,d} \le b_{i,j} \cdot t_{s,d} \qquad \forall i, j, s \tag{21}$$

Equations (8) and (9) define the total flow passing through the lightpath *(i,j)* as the sum of the source-destination specific flows passing through that path and force these $f_{i,j}$ variables of being less than an upper bound: the maximum flow.

Equation (10) states that all the traffic from a source node s to a destination node d is directed to a single node, instead of splitting the flow between other nodes. This can be derived from Equations (13) and (16) which exploit the binariness of the $k_{i,j}^{s,d}$ variable by linking only one node to the source and destination ones.

Equation (11) ensures that the amount of total flow (its sum over all the *s,d* couples) entering a node is exactly the amount of flow that exits from it.

Equations (12) and (15) ensure that the flow exiting the source and entering the destination satisfies the traffic requirement, whereas Equations (14) and (17) speed up the convergence of the algorithm by defining the sources and the destination: a source node cannot accepts incoming flow and a destination one cannot bring out flow.

Equation (18) prevent the flow from looping over the same node.

Equations (30) and (31) force the number of lightpaths linked to each node to be at maximum equal to the number of receivers and transmitters of the nodes.

Equation (32) states that if a lightpath is chosen, $b_{i,j}$ is equal to 1, so $f_{i,j}^{s,d}$ is equal to the traffic from s to d.

1.1.5 Flow Splitting

Enabling Flow Splitting means that for a specific *s,d* couple, the flow coming from the *i-th* node going to the *j-th* node, can be transmitted to more than one node, by splitting the flow.

From a mathematical point of view, this means remove all the equations containing the **K** matrix (Equations (3), (10), (13) and (16)).

1.2 LP Model: Square Mesh Grid Topology

According to what has been said in Section 1.1, the optimum solution and the LP model depend on the topology. Instead of a full mesh, it can be interesting to evaluate the problem solving algorithm performances when a Square Mesh Grid (SMG) topology is used.

The model, and so the code implementation, are the same of the Section 1.1 case with the flow splitting enabled, with the addition of the square grid constraint, which assigns non-zero flow only to adjacent nodes (Equation (24)).

$$r = \left\lfloor \sqrt{N} \right\rfloor \tag{22}$$

$$c = \left\lceil \frac{N}{r} \right\rceil \tag{23}$$

$$f_{i,j}^{s,d} \begin{cases} & \forall \left\{(i-1) \mod c = 0, j = c \cdot \left\lceil \frac{i}{r} \right\rceil \right\} \wedge \\ & \forall \left\{i \neq j, j = i+1, i \mod c \neq 0\right\} \wedge \\ & \forall \left\{i \neq j, j = i+c\right\} \wedge \\ \neq 0 & \forall \left\{i \leq c, j = i + c \cdot (r-1)\right\} \wedge \\ & \forall \left\{(j-1) \mod c = 0, i = c \cdot \left\lceil \frac{j}{r} \right\rceil \right\} \wedge \\ & \forall \left\{j \neq i, i = j+1, j \mod c \neq 0\right\} \wedge \\ & \forall \left\{j \neq i, i = j+c\right\} \wedge \\ & \forall \left\{j \leq c, i = j + c \cdot (r-1)\right\} \\ = 0 & \text{otherwise} \end{cases} \tag{24}$$

where N is the number of nodes, r is the number of the grid rows, c is the number of grid columns and $f_{i,j}^{s,d}$ is the same specific flow variable of Equation (6).

1.3 Aggregated Flow

A similar model can be obtained by aggregating the flows with respect to the destinations. To do this a new traffic array $\mathbf{t^s}$ is defined as

$$t_s^s = \sum_d t_{s,d} \tag{25}$$

Thanks to these new variables, the equations of Section 1.1.4 become:
Flow Relationships:

$$f_{i,j} \leq F_{max}, \qquad \forall i,j \tag{26}$$

$$f_{i,j} = \sum_s f_{i,j}^s, \qquad \forall i,j \tag{27}$$

Flow Conservation:

$$\sum_j f_{i,j}^s - \sum_j f_{j,i}^s = -t_{s,i}, \qquad \forall i \neq s \qquad \forall s \tag{28}$$

$$\sum_j f_{i,j}^s - \sum_j f_{j,i}^s = t_s^s, \qquad \forall i = s \qquad \forall s$$

$$\tag{29}$$

Connectivity:

$$\sum_i b_{i,j} \leq \Delta, \qquad \forall j \tag{30}$$

$$\sum_j b_{i,j} \leq \Delta, \qquad \forall i \tag{31}$$

Feasibility:

$$f_{i,j}^s \leq b_{i,j} \cdot t_s^s \qquad \forall i, j, s \tag{32}$$

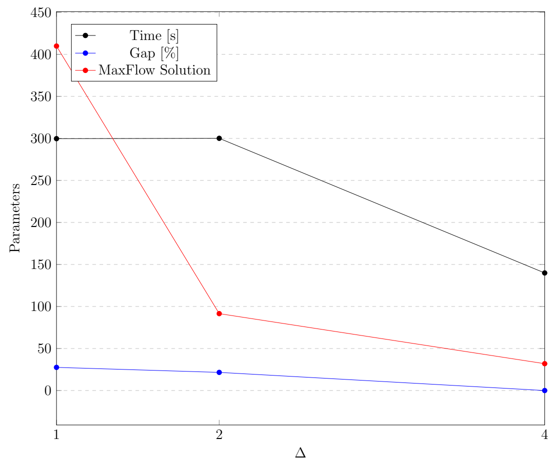
1.4 Experimental Results

When the full mesh topology is used, the traffic matrices are a balanced one (initialized with random values uniformly distributed between [6,12]) and an unbalanced one (initialized with 50% of the random values uniformly distributed between [1,3], and 50% between [15,35].). When the SMG topology is used, the traffic matrix is initialized with random values uniformly distributed between [1,10].

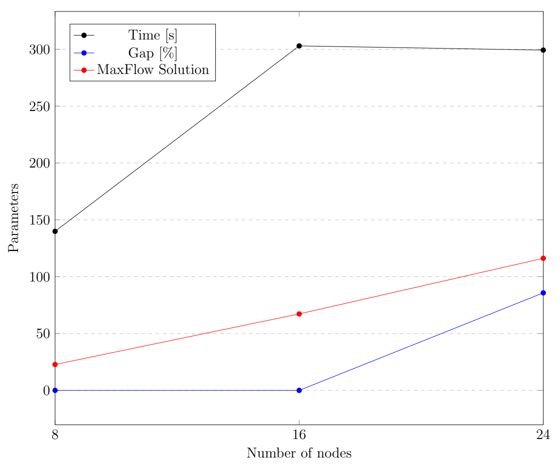
	Splitting Balanced	Non-SplittingBalanced
Max Time [s]	300.60	300.20
Avg. Time [s]	300.27	299.93
Min Time [s]	300.00	299.60
Max Gap [%]	1.311	19.650
Avg. Gap [%]	1.197	18.466
Min Gap [%]	1.042	17.027
Max MaxFlow Solution	40.623	50.037
Avg. MaxFlow Solution	40.687	49.336
Min MaxFlow Solution	40.104	48.706

Table 1: Full Mesh. Performance results with N=12, $\Delta = 4$. The minimum, maximum and average values are obtained with different seeds.

According to Table 1, when the flow splitting is enabled, the routing strategy allows to obtain an upper bound of the total flow passing through each path less than the one obtained when the flow splitting is disabled thanks to the fact that if the traffic is routed through more than one node per hop the amount of flow of each node is less than the total traffic. In this way the optimization algorithm can be seen as more "dynamic", so the gap between the optimal solution and the one determined by the algorithm is close to zero for the flow splitting case, whereas in the non-splitting one the performances are worse. The design constraints N and Δ are the same in both the cases, so the computational times are quite the same.



(a) $N = 10$



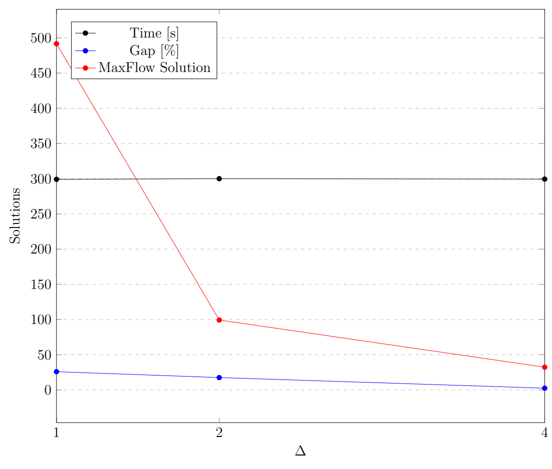
(b) $\Delta = 4$

Figure 1: Average computational time, gap and solutions for the algorithm performances evaluation.
Full Mesh Topology. Balanced **T**. Flow Splitting enabled. Algorithm timeout = 300s

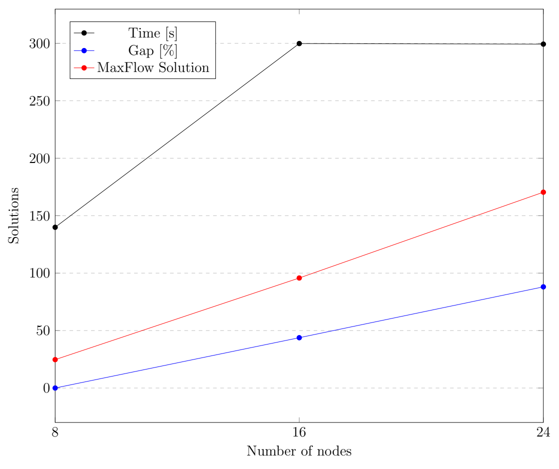
Figure 1 shows the performance evaluation parameters when the full mesh topology with balanced traffic matrix is used and the flow splitting is enabled. The computational time is directly proportional to the number of nodes N in the topology and inversely proportional to the number of transceivers Δ. In fact, if N is fixed (Figure 1(a)), $\Delta = 1$ is a tight constraint, since the algorithm performs a high number of steps until the condition is not satisfied (in a way $\Delta = 1$ is the same of disabling flow splitting). By increasing the value of Δ, the flow is actually splitted and the algorithm requires a lower number of iterations to reach the solution.

Dually, when Δ is fixed (Figure 1(b)), more nodes means a bigger topology with the same constraint, so the computational time is higher.

A similar reasoning leads to the obtained gap and MaxFlow solution results.



(a) $N = 10$



(b) $\Delta = 4$

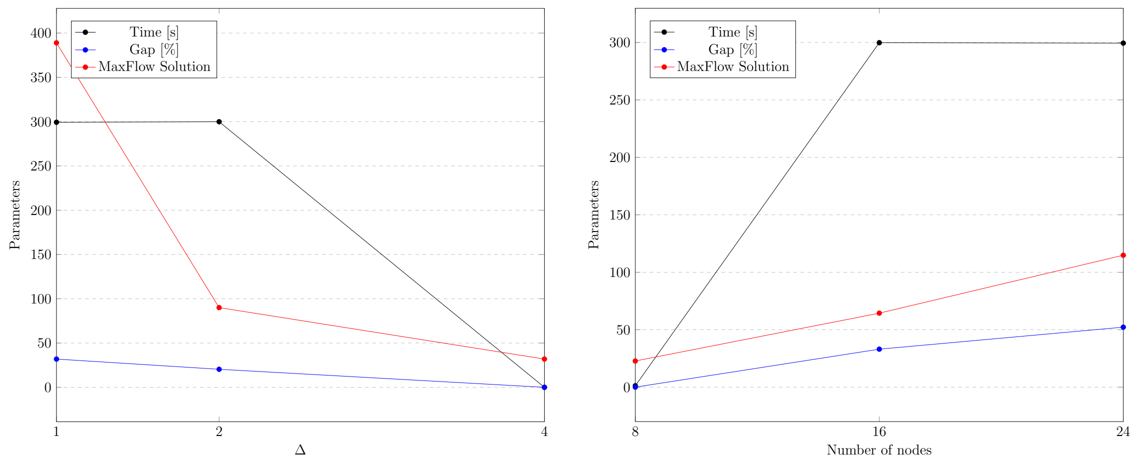
Figure 2: Computational time, gap and solutions for the algorithm performances evaluation.
Full Mesh Topology. Unbalanced **T**. Flow Splitting enabled. Algorithm timeout = 300s

The same considerations made for the full mesh topology with a balanced traffic matrix can be extended to the unbalanced traffic case shown in Figure 2: the overall results are coherent with the ones of Table 1. The performances are generally worse because the new traffic matrix is initialized with inhomogeneous traffic values.

	Splitting Balanced
Time $[s]$	0.45
Gap [%]	0
Best Bound	48.272
MaxFlow Solution	48.272

Table 2: Square Mesh Grid. Performance results with N=16, $\Delta = 4$.

Thanks to the square mesh grid topology described in Section 1.2, the flow is forced to pass only through a precise combination of nodes. Even if this is a tight constraint, the range of choice is decreased and the optimum solution (0% Gap) is obtained in less time than one of the full mesh topology.



(a) $N = 10$ (b) $\Delta = 4$

Figure 3: Computational time, gap and solutions for the algorithm performances evaluation. Aggregated flows. Full Mesh Topology. Balanced **T**. Flow Splitting enabled. Algorithm timeout = 300s

Similar considerations can be made by observing the results of Figure 3. The maximum flow solution and the gap are the same of Figure 7, except for the computational time. For the most of the cases its values are around 300s because of the timeout of the algorithm, but with the setting $N = 10, \Delta = 4$ (Figure 3(a)) and $N = 8, \Delta = 4$ (Figure 3(b)) the computational time tends to 0 s. This happens thanks to the lower number of constraints described in Section 1.3 and the algorithm is more efficient.

2 Greedy Heuristic

The greedy heuristic algorithm intends to find the optimal final solution by taking as intermediate steps the local optimal one. The result may be a sub-optimal solution but the algorithm is cheaper in terms of computational time and resources. An example of such algorithm implemented in MATLAB is proposed in this section.

2.1 Algorithms: Implementation

The aim of the algorithm is to solve the LTD problem described in Section 1 by considering a topology of N nodes with Δ transceivers.

It is based on three parts: the actual heuristic algorithm, the Local Search one which allows to improve the solution obtained by the heuristic and finally the additional Node Placement algorithm which implements the Manhattan topology.

2.1.1 Heuristic

Algorithm 1 Pseudocode of the greedy heuristic algorithm

```
procedure HEURISTIC ALGORITHM(T matrix, $\Delta$)

    graph and edges weights random initialization;
    while nodes' degree $> \Delta$ do
        sort the weights;
        get the minimum weight $\notin$ Taboo;
        if the minimum weight $\in$ Taboo then
            skip the element and proceed with the next minimum;
        if Taboo == weights then
            break;
        remove the edge(i,j) of the minimum weight from the graph;
        if new graph is not strongly connected then
            add the edge(i,j) just removed to Taboo;
            re-add the edge(i,j) to the graph;
        else
            add the edge(i,j) just removed to Free;
            determine the shortest path from i to j with respect to the weights;
            for edges in shortest path do
                add the weights of the removed edge(i,j);
results: Graph solution, Taboo matrix, Free matrix
```

The heuristic starts from a full-mesh topology and creates a directed graph from it. The graph weights are the amounts of traffic of the couples of nodes defined in the traffic matrix, **T**.

The edge matched with the minimum weight is removed. If the graph is no more strongly connected, the edge is re-added to the graph and is marked as unremovable by storing it in the Taboo matrix. Otherwise it is stored in the Free matrix.

Now the flow is re-routed through the new topology by determining the minimum shortest path between the source-destination nodes with respect to the weights. Once the shortest path is found, the removed weights are reassigned to the edges belonging to the path.

This is repeated until the nodes' degree constraints are not satisfied or until all the edges of the graph resulting at each iteration are not in the Taboo matrix. In this case the found solution is not feasible.

2.1.2 Nodes Placement

Algorithm 2 Pseudocode of the nodes placement algorithm

```
procedure NODES PLACEMENT(T matrix, B matrix, Map matrix)

    B matrix initialization with the default SGM topology;
    graph initialization with T;
    sort the weights;
    for all edges(s,d) in sorted weights graph do
        if node s is not in Map then
            add node s in the first empty entry of Map;
            for entries of B(s,:) do
                find the first j entry marked as 1;
                if node j is not in Map then
                    add destination node d in the j-th row of Map;
                    set B(i,j)=B(j,i)=0;
                    break;
        else
            determine $x$ as the position of node s in Map;
            for entries of B(x,:) do
                find the first j entry marked as 1;
                if node j is not in Map then
                    add destination node "d" in the j-th row of Map;
                    set B(x,j)=B(j,x)=0;
                    break;
                else
                    set B(x,j)=B(j,x)=0;
results: Map matrix
```

The nodes placement algorithm implements a placement strategy based on the traffic exchanged between nodes (**T** matrix). The idea is to place the most traffic exchanging nodes close to each other in a provided topology (logical matrix **B**). The algorithm creates a look-up table (Map matrix) in witch the nodes mapping information are stored.

To create the look-up table, the list of the edges of the graph (Section 2.1.1) is parsed and the edge (s, d) corresponding to the maximum weight is removed. If the source node s is not already in the table, it is mapped to the first available node (e.g. if the

Map array contains already 3 mapped nodes and the source node is $s = 5$, then node 5 is mapped into node 4) and the the destination node d is assigned to the first available position linked to the mapped node. This decision exploits the binariness of $\mathbf{B}$ and once the assignment $s \to s^*$, $d \to d^*$is completed, the corresponding entries b_{s^*,d^*} and b_{d^*,s^*} are set equal to 0.

This is repeated for all the edges in the original graph. When the look-up table is created, a new topology is obtained by mapping $\mathbf{B}$ into the new one.

2.1.3 Local Search

Algorithm 3 Pseudocode of the local search algorithm

```
procedure LOCAL SEARCH(Graph, Taboo matrix, Free Matrix)

    for all the edges in Taboo do
        remove the edge(i,j) $\in$ Taboo from the graph;
        store the removed edge weight;
        for all the edges in Free do
            add the edge $\in$ Free to the graph;
            if the resulting new graph is still strongly connected then
                determine the shortest path from i to j with respect to the weights;
                for edges in shortest path do
                    add the weights of the removed edge(i,j);
                re-apply Algorithm 1 to the new graph;
results: Graph solution, Taboo matrix, Free matrix
```

The Local Search is a metaheuristic algorithm which allows to find the minimum solution by searching between all the neighbors of the starting solution. The neighbors are defined as the solutions obtained when the inputs of the algorithm are modified by a step called *distance*.

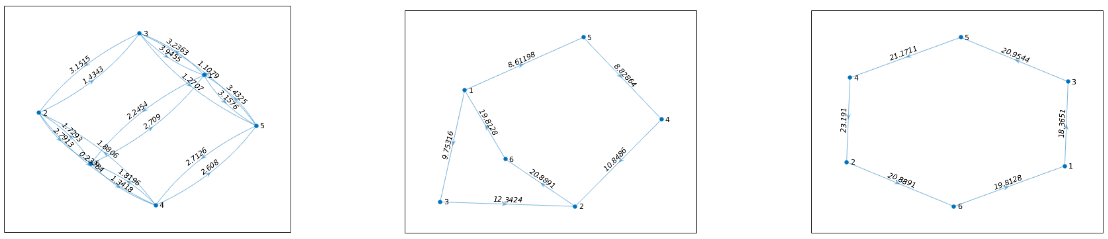
The Local Search takes as input the solution of the greedy heuristics (the outputted graph, the Taboo matrix and the Free one) shown in Section 2.1.1 and determines the neighbors of this solution by re-routing the flow when one of the edge in Taboo is replaced by an edge in Free. Before the re-routing, the strongly connectivity of the graph is checked.

After having determined all the solutions of the *search space*, the feasibility is checked and the minimum solution is taken.

2.2 Manhattan Topology

The Manhattan topology is a topology where each node is connected to other 4 nodes: left, right, above and below. This means that the number of transmitters and receivers must be equal to 4. This kind of topology is completely regular, so it does not change if nodes are rotated or shifted and, given two nodes (s and d), many routes of equal length exist between them.

To obtain this topology, the constraints of Section 1.2 are implemented in the algorithm of Section 2.1.2



(a) Manhattan Topology Nodes Placement

(b) Manhattan Topology Heuristic solution

(c) Manhattan Topology Local Search solution

Figure 4: Step-by-step results of the greedy heuristic and the local search algorithms applied to a Manhattan topology. $N = 6$, $\Delta = 1$

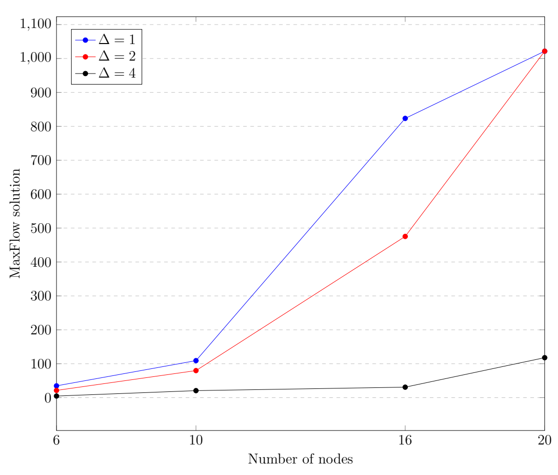
Figure 4(a) shows the Manhattan topology with a reduced number of nodes: in this case the number of transceivers is $\Delta = 3$. Figures 4(b) and 4(c) are an overview of the two algorithms. By setting the degree constraints $\Delta = 1$, the heuristic cannot find a feasible solution, because of the random choice of an edge if some of them have the same weights. The unfeasibility is that some nodes of Figure 4(b) have $\Delta = 2$. Figure 4(c) shows that in this case the Local Search algorithm allows to find a feasible solution by starting from the unfeasible one. Now all the nodes have $\Delta = 1$.

2.3 Experimental Results

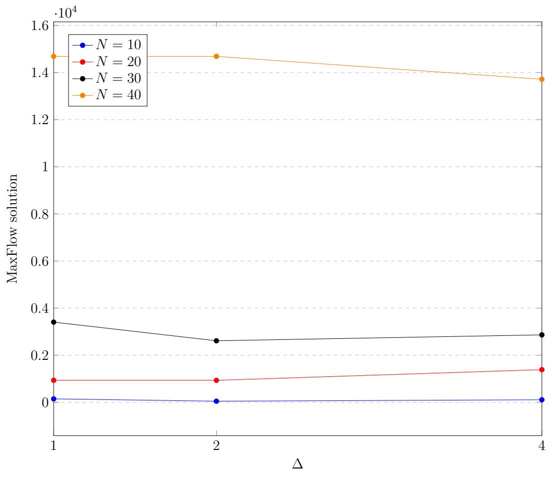
When the full mesh topology is used, the traffic matrices are a balanced one (initialized with random values uniformly distributed between [0,4]) and an unbalanced one (initialized with 10% of the random values uniformly distributed between [5,15], and 90% between [0,3].). When the Manhattan topology is used, the traffic matrix is initialized with random values uniformly distributed between [0,4].

The algorithm described in Section 2.1.1 can be seen as a *Route and Remove* one, since it starts from the full-mesh topology and progressively remove the edges. The complexity of the basic R&R is really dependent on the number of nodes and the solutions shown in Figure 5(b) are coherent with that, since the results grow linearly with N. The slope between $\Delta = 2$ and $\Delta = 4$ is explained in Section 1.4.

Figure 5(a) shows even more coherent result with what has been said in Section 1, since the maximum flow solution decrease linearly with a decreasing number of Δ because of the tightness of the constraints.

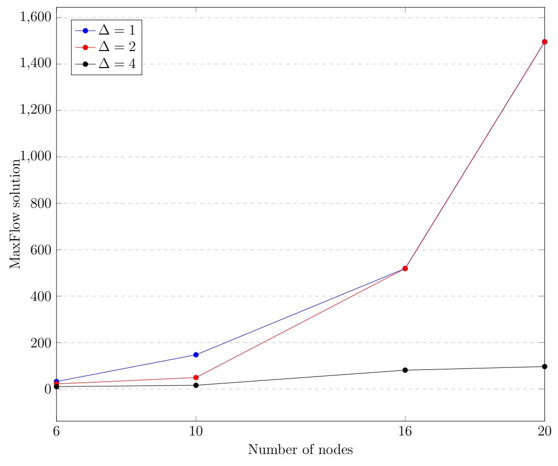


(a) Increasing number of nodes N

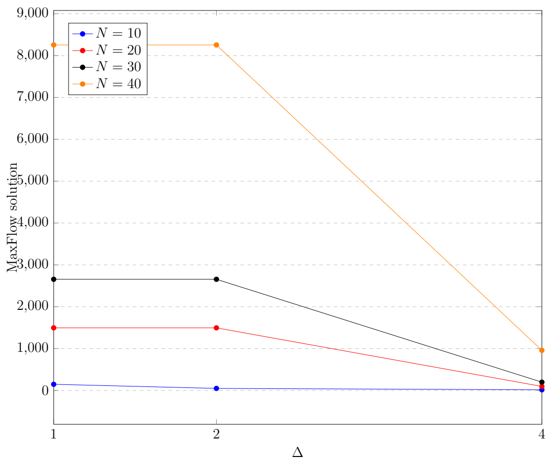


(b) Increasing value of Δ

Figure 5: Maximum Flow resulting from the application of the greedy heuristic algorithm. Full mesh topology. Balanced traffic matrix $\mathbf{T}$



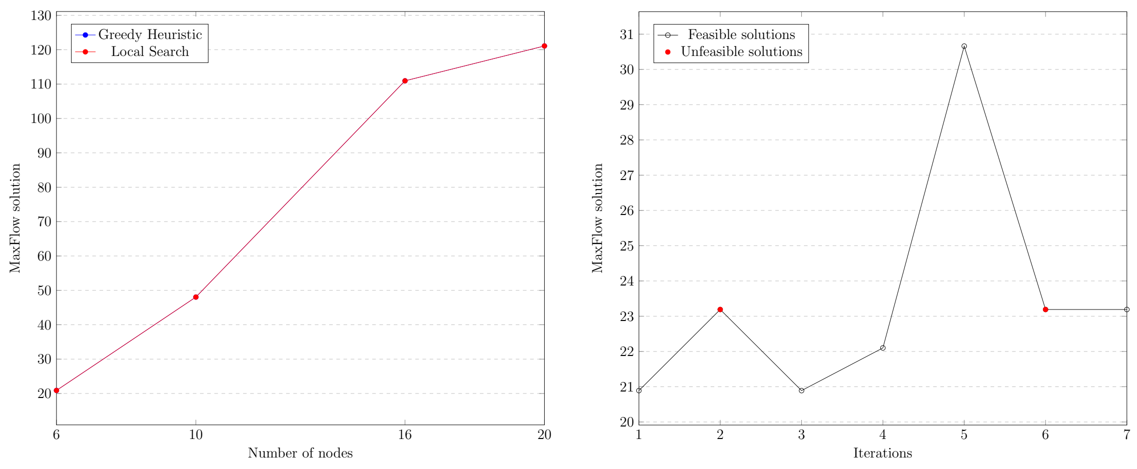
(a) Increasing value of Δ



(b) Increasing number of nodes N

Figure 6: Maximum Flow resulting from the application of the greedy heuristic algorithm. Full mesh topology. Unbalanced traffic matrix $\mathbf{T}$

By comparing Figures 5 and 6 with Figures 1 and 2 it is clear that the same conclusions regarding the unbalanced $\mathbf{T}$ can be drawn. Moreover, the overlapping of the results shown in Figure 6(a) when 16 and 20 nodes are used with $\Delta = 1$ and $\Delta = 2$ can be explained by considering the stopping condition of the heuristic algorithm. In these cases the algorithm stopped because the Taboo matrix contains all the edge of graph and the solutions are unfeasible.



(a) $N = 16, \Delta = 4$ (b) Search Space. $N = 5$, $\Delta = 1$

Figure 7: Local Search performances. Manhattan Topology. Balanced **T**.

Figure 7(a) shows that when the Manhattan topology is used, the application of the Local Search algorithm does not improve the solution. This is due to loose degree constraint and to the regularity of the topology which allow to obtain optimal feasible solution in both cases.

On the other hand, Figure 7(b) shows that if the Δ constraint is tighter, the local search allow to improve the solution found by the greedy heuristic by making it feasible. Since the unfeasibility means an higher degree for some nodes, so a greater flow distributions among the edges, feasible solutions have higher values.

3 Green Networking

Green Networking means projecting and developing a network by optimizing power consumption parameters. There are different techniques used to reach such goal, like solving the LTD problem described in Section 1 when some nodes of the network are temporarily shut down or put in idle (or sleep mode).

3.1 LP Model

Provided a full mesh topology, the capacity of each link of the network and a static power consumption of the nodes, the green networking LTD problem can be solved with LP. The variables and the constraints are quite the same of Section 1, with some changes.

3.1.1 Inputs/Outputs

$$\mathbf{T} \in \mathbb{R}^{NxN} \tag{33}$$
$$\mathbf{p}_{idle} \in \mathbb{R}^{N} \tag{34}$$
$$\mathbf{q} \in \mathbb{R}^{N} \tag{35}$$
$$\mathbf{CL} \in \mathbb{R}^{NxN} \tag{36}$$

$$\mathbf{x} \in \mathbb{N}^{N} \qquad \text{with } x_i \in \{0,1\} \qquad \forall i \tag{37}$$
$$\mathbf{Y} \in \mathbb{N}^{NxN} \qquad \text{with } y_{i,j} \in \{0,1\} \qquad \forall i,j \tag{38}$$

where $\mathbf{T}$ is the traffic matrix, $\mathbf{p}_{idle}$ is the array of the static power consumption of each node. When the node is putting in idle, this is the unique power consumption. By assuming that when the node is active its power consumption is a linear function of the load, $\mathbf{q}$ stores the angular coefficient of the plot with respect to the static power consumption. $\mathbf{CL}$ is the link capacity matrix.

Since the cost to minimize are changed, the new outputs are the $\mathbf{x}$ array, which is an indicator one stating if a node is in idle ($x_i = 0$) or not, and the $\mathbf{Y}$ indicator matrices which states if the link is used or not.

3.1.2 Variables

The variables are the same Equations (3), (5) and (6) of Section 1.1.2 with the addition of the total power consumption one:

$$P_{tot} = \sum_i \left\{ P_{idle}(i) \cdot x_i + q_i \cdot \sum_j f_{i,j} \right\} \tag{39}$$

The power consumption is defined as the sum of the static power consumption and the load of the nodes with an inclination equal to the angular coefficient (q_i), over all the nodes.

3.1.3 Objective Function

Even if the LTD problem solution aims at optimizing a routing problem, the Green Networking one requires the minimization of the total power, so the new objective function is:

$$minP_{tot} \tag{40}$$

3.1.4 Constraints

The constraints of the model are the same of Section 1.1.4, with the addition of the **Y** and **x** output contribution. The model must be changed by adding to Equations (8) to (32) the new constraint
Flow relationship:

$$\sum_j f_{s,j}^{s,d} = t_{s,d} \cdot y_{i,j} \qquad \forall s, d \tag{41}$$

$$f_{i,j} \leq \alpha \cdot c_{i,j} \cdot y_{i,j} \qquad \forall i, j \tag{42}$$

and by replacing Equations (30) and (31) with
Connectivity:

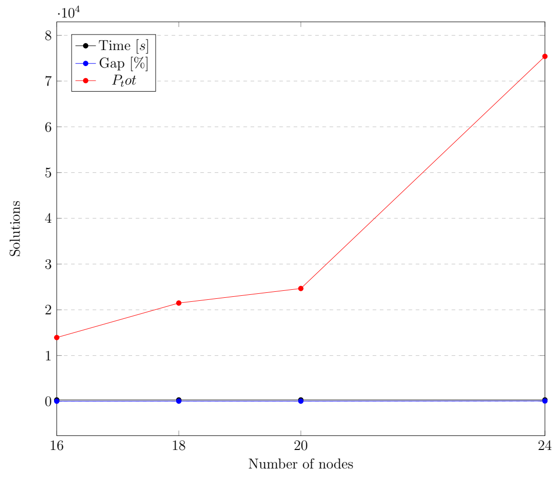
$$\sum_j y_{i,j} + \sum_j y_{j,i} \leq 2\Delta \cdot x_i, \qquad \forall i, j \tag{43}$$

Where Δ is the same of Section 1.1 and $\alpha \in [0, 1]$ ensures that the maximum link capacity is not exceeded.

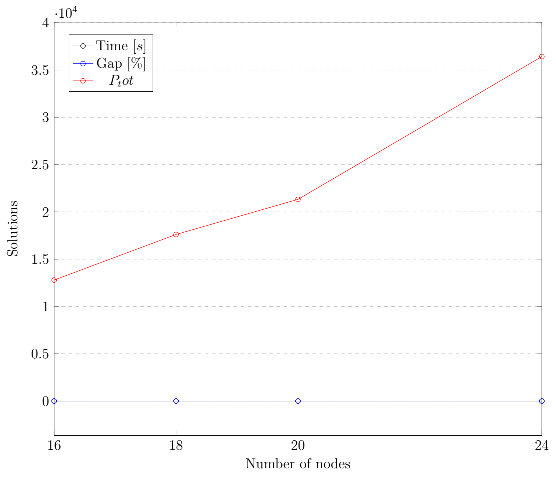
3.2 Experimental Results

The topology used are the full mesh one with slow splitting disabled described in Section 3.1 and the SMG one obtained by adding the constraints of Section 1.2 to the full mesh model.

The traffic matrix is initialized with random values uniformly distributed between [6,12], the link capacity between [800,1000], the angular coefficient between [1,2] and the static power consumption between [10,15].



(a) Full Mesh Topology. $\Delta = 4$. Increasing number of nodes



(b) Manhattan Topology. $\Delta = 4$. Increasing number of nodes

Figure 8: Green Networking performances. Balanced **T**

According to what has been said in Sections 1.4 and 2.3, Figure 8 the loose degree constraint ($\Delta = 4$) allows to reach optimal solution with a 0% gap in low computational times. Figure 8(b) confirm the better results in terms of objective functions, since the overall power obtained when the Manhattan topology is used with an increasing number of nodes is about the half of the one of Figure 8(a).