TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2015-2016

Lab work #10. Joint topology, routing and capacity design (TCFA)

(1 session)

Author:

Pablo Pavón Mariño

1 Objectives

The goals of this lab work are:

1. Create Net2Plan algorithms that solve several variants of the TCFA problem (Topology, Flow (routing), Capacity assignment) solving the formulations using the *Java Optimization Modeler* (JOM) library.
2. Gain experience with the different forms of writing optimization problems in JOM.

2 Duration

This lab work is designed for one session of two hours.

3 Evaluation

This lab work has been designed to guide the students in their learning of Net2Plan. The annotations the students make in this document are for their use when studying the course, and do not have to be delivered to the teacher for evaluation.

4 Documentation

The resources needed for this lab work are:

- JOM library documentation (see `http://www.net2plan.com/jom`).
- Net2Plan tool and their documentation (see `http://www.net2plan.com/`).
- Instructions in this wording.

5 Previous work before coming to the lab

- Read Section 7.3 of [1], and the lecture notes regarding flow-link formulations.
- Refresh your reading in the JOM documentation in `http://www.net2plan.com/jom`, in particular, how vector of variables and constraints are handled.

6 TCFA problem

Let $\mathcal{N}$ be a given set of nodes, and $\mathcal{D}$ the offered traffic to the network. For each demand d, h_d denotes its known offered traffic. We are interested in creating a Net2Plan algorithm that solves the TCFA problem that finds (i) the links to install in the network, (ii) their capacities, and (iii) how the traffic in the demands is routed over the links. Two nodes can be connected by zero or one link, and the maximum link capacity is denoted as U.

The optimization target is minimizing the total network cost, which sums a fixed cost c_{km} per km of link installed (whatever its capacity is), and a fixed cost c_u per capacity unit installed (whatever the link length is).

The problem is formulated as follows:

- Input parameters (known constants):
  - $\mathcal{N}$: Set of network nodes.
  - $\mathcal{E}$: Set of *candidate* network links. We have one candidate link for each node pair. From this information, $\delta^+(n)$ denotes the set of candidate links outgoing from node n, and $\delta^-(n)$ the set of incoming candidate links to n.
  - $\mathcal{D}$: Set of offered unicast demands.
  - $h_d, d \in \mathcal{D}$: Offered traffic of a demand d.
  - U: Maximum capacity of a link.
- Decision variables:
  - $z_e, e \in \mathcal{E}$: 1 if candidate link e is actually installed, and 0 otherwise (there is no link there, and then the capacity of this candidate link must be zero).
  - $x_{de}, d \in \mathcal{D}, e \in \mathcal{E}$: Traffic of demand d that traverses candidate link e.
  - $u_e, e \in \mathcal{E}$: Capacity of candidate link e.
- Formulation:

$$\min \quad c_{km} \sum_e d_e z_e + c_u \sum_e u_e, \quad \text{subject to:} \tag{1a}$$

$$u_e \leq U z_e, \quad \forall e \in \mathcal{E} \tag{1b}$$

$$\sum_{e \in \delta^+(n)} x_{de} - \sum_{e \in \delta^-(n)} x_{de} = \begin{cases} h_d, & \text{if } n = a(d) \\ -h_d, & \text{if } n = b(d) \\ 0, & \text{otherwise} \end{cases}, \quad \forall d \in \mathcal{D}, \forall n \in \mathcal{N} \tag{1c}$$

$$\sum_d x_{de} \leq u_e, \quad \forall e \in \mathcal{E} \tag{1d}$$

$$u_e \geq 0, x_{de} \geq 0, \quad \forall d \in \mathcal{D}, e \in \mathcal{E} \tag{1e}$$

The objective function (1a) minimizes the total network cost, summing the cost of the links, and the cost of the capacity in them. Constraints (1b) makes that (i) if a candidate link e does not exist ($z_e = 0$), then there cannot be capacity in it ($u_e = 0$), and (ii) if a candidate link exists ($z_e = 1$), the link capacity is limited to U. Constraints (1c) are the flow conservation constraints. Constraints (1d) mean that for each link, the traffic carried in the link is less or equal than its capacity (and thus, no link is oversubscribed). Finally, (1e) means that capacity links and carried traffics are non-negative.

7 Net2Plan algorithm

The student should develop a Net2Plan algorithm solving problem (1) following the next steps:

1. Copy the `AlgorithmTemplate.java` file in Aula Virtual and rename it as `FlowLinkUnicast.java`.

2. The algorithm has the input parameters:
   - `maximumLinkCapacity`, of default value 1000 (U in (1)).
   - `costPerGbps`, of default value 1 (c_u in (1)).
   - `costPerKm`, of default value 1 (c_{km} in (1)).
3. Remove all the network links.
4. Set the routing type as *source routing*.
5. Add a full mesh of links in the `netPlan` design (one betweem each pair of nodes). They are the candidate links $\mathcal{E}$ in (1). Link capacity is set to zero, its length in km is set as the Euclidean distance between the nodes (use `getNodePairEuclideanDistance`), and the propagation speed is et to 200000 km/s.
6. Create an object of the type `OptimizationProblem` (e.g. of name `op`).
7. Add the problem decision variables:
   - `z_e`: one variable per candidate link, which can take the values 0 or 1. *Double.MAX_ VALUE.*
   - `u_e`: one variable per candidate link, with the link capacity which can take values between 0 and `maximumLinkCapacity`.
   - `x_de`: one variable per demand and candidate link. The minimum value of the variables is set to zero, the maximum to *Double.MAX_ VALUE.*
8. Set the problem objective function.
9. Use a for loop, with one iteration per link (candidate), to add the constraints (1b). One constraint is added inside each iteration of the loop. For adding the constraint of a candidate link `e`, set a JOM input parameter of name `e`, with a value equal to the index of the link.
10. Use a double for loop with one iteration per network node, and then one inner iteration per demand, to add the constraints (1c). For adding the conservation constraint of a demand `d` and node `n`:
    - Set the JOM input parameters:
      - `deltaPlus` with the indexes of the output candidate links of the current node. For this, use the method *getOutgoingLinks* of the node object to get the output links, and the method *NetPlan.getIndexes* to convert the collection of links to their indexes.
      - `deltaMinus` with the indexes of the incoming candidate links of the current node. For this, use the method *getIncomingLinks* of the node object to get the incoming links, and the method *NetPlan.getIndexes* to convert the collection of links to their indexes.
      - `h_d` with the current demand offered traffic.
      - `d` with the current demand index.
    - Set the constraint using the function `sum`, over `x_de`, but restricting the sum to the elements in row `d` and the columns in `deltaPlus` or `deltaMinus`.
11. Use a for loop with one iteration per candidate link, to add the constraints (1d). One constraint is added inside each iteration of the loop. For adding the constraint of a candidate link `e`:
    - Set a JOM input parameter of name `e` with the current candidate link index.
    - Set the constraint using the function `sum`, over all the demands (using the JOM keyword `all` in the demand coordinate), and the candidate link of index `e`.

12. Call the solver to find a numerical solution. Use the option `maxSolverTimeInSeconds` to set the maximum solver time to 10 seconds. The solver will return the best solution found so far after 10 seconds, if an optimum solution was not found before. It may happen that the solver could not find any feasible solution. To check this situation use the method `solutionIsFeasible` of the `OptimizationProblem` object, and raise an exception if a feasible solution was not found by the solver.

13. Retrieve the primal solution obtained for `x_de`, and convert it into a *DoubleMatrix2D* object using the method *view2D*. An object of the class *DoubleMatrix2D* contains a 2D matrix and permits making operations with it efficiently. Use the method of *NetPlan* called *setRoutingFromDemandLinkCarriedTraffic*. This method automatically creates the routes in the network that are consistent to what appears in the x_{de} 2D matrix computed. Note that in the x_{de} values obtained, each (d, e) coordinate contains the amount of traffic of d in link e, and *not* the fraction of traffic respect to h_d. This is important when calling the method *setRoutingFromDemandLinkCarriedTraffic*. Since, this formulation cannot create loops (solutions with loops are never optimal, since they are strictly worse than the same routes without loops), set the option that automatically eliminates them to false.

14. Retrieve the primal solution obtained for `u_e`. Set the capacity of the candidate links to that given by the `u_e`.

15. Remove all the candidate links without capacity installed, since these links are not part of the design. For this, use the method `removeAllLinksUnused` in the `netPlan` object. This method removes all the links with a capacity lower than a threshold. Use 0.01 as a threshold[1].

7.1 Check the algorithm

Load the network `example4nodes.n2p`, and the traffic matrix in `tm4nodes.n2p`. The algorithm should produce a solution with six links and a total capacity installed summing all the network links of 108.18 units.

Quiz 1. Load the network `example4nodes.n2p`, and the traffic matrix in `tm4nodes.n2p` (total offered traffic equal to 100).

- Run the algorithm with parameter `costPerKm = 0`, and any no-zero value in `costtPerGbps`. How is the topology created? why?

  Answer: The topology is a full-mesh with all the links. The reason is that links are unexpensive, while capacity has a cost. Then, it is better to carry the traffic in direct links between the end nodes, instead of making the traffic traverse more than one link.

- Fix parameter `costPerKm = 1`, `maximumLinkCapacity = 1000` and fill in Table 1 running the algorithm for different values of `costPerGbps`. Explain the observed of the topologies as the `costPerGbps` grows.

  Answer: As the cost per Gbps of capacity grows, it is better the cost of the links becomes more and more unimportant. Then, it is better to add more links to the design, so more traffic follows shortest routes (and thus the total consumed bandwidth is lower).

[1]The reason for not using zero as a threshold is that the solver can assign non-zero but small capacities to some links, because of its finite numerical precision. These links should be eliminated from the design.

Table for `example4nodes.n2p`

`c_Gbps`	Number of links	Total capacity installed
0.01	4	200
0.1	5	144
1	6	108.18
10	8	102.2
100	12	100

8 Problem variations

Quiz 2. Modify formulation (1) to force the capacity in the links to be an integer multiple of C, the so-called capacity module. Then, solve the formulation:

$$\min \quad c_{km}\sum_e d_e z_e + c_u \sum_e C a_e, \quad \text{subject to:} \tag{2a}$$

$$C a_e \leq U z_e, \quad \forall e \in \mathcal{E} \tag{2b}$$

$$\sum_{e\in\delta^+(n)} x_{de} - \sum_{e\in\delta^-(n)} x_{de} = \begin{cases} h_d, \text{ if } n = a(d) \\ -h_d, \text{ if } n = b(d) \\ 0, \text{ otherwise} \end{cases}, \quad \forall d \in \mathcal{D}, \forall n \in \mathcal{N} \tag{2c}$$

$$\sum_d x_{de} \leq C a_e, \quad \forall e \in \mathcal{E} \tag{2d}$$

$$a_e \in \{0,1,2,\ldots\}, x_{de} \geq 0, \quad \forall d \in \mathcal{D}, e \in \mathcal{E} \tag{2e}$$

where decision variables $a_e, e \in \mathcal{E}$ are now the (integer) number of capacity modules installed in the link e, and Ca_e is now the capacity of link e, and replaces u_e in formulation (1). Introduce C as a user-defined parameter with name `capacityModule`, and default value 10.

Note: The algorithm with the default parameters should install 8 links and 120 capacity units in the network `example4nodes.n2p` with the traffic matrix in `tm4nodes.n2p`.

Quiz 3. Will *always* formulation (2) produce results with a higher cost than formulation (1)? Why?

Answer: The cost will be always higher. The reason is that problem (2) is the same as problem (**??**) with more constraints: now the capacities have to be modular. Any feasible solution in (2) is also valid for (1), but the other way around may not hold.

9 Matricial form of problem constraints (optional)

As has been shown in other lab sessions, flow conservation constraints (1c) and link capacity constraints (1d) can be set in only single call to `addConstraint` method, using matrix and vectors of constraints.

- Flow conservation constraints can be set with:

$$A_{ne} \times x'_{de} = A_{nd} \times \mathbf{diag}(h) \tag{3}$$

where A_{ne} is the node-link incidence matrix which can be obtained as an sparse matrix in Net2Plan using the `NetPlan` method `getMatrixNodeLinkIncidence`. A_{nd} is the node-demand incidence matrix, which can be obtained with the `NetPlan` method `getMatrixNodeDemandIncidence`. Finally, `h` is the vector with the demands' offered traffic, which can be obtained with `getVectorDemandOffered`

- Link capacity constraints can be set with the expression:

  ```
  sum(x_de,1) <= u_e
  ```

  which makes the vector of traffic carried in each link, be element-by-element less or equals than the vector of link capacities.

Constraints (1b), can be easily set in a vectorial form with:

```
u_e <= U * z_e
```

Quiz 4. Rewrite the algorithm of (1) using their matricial form.

10 Work at home after the lab work

The student is encouraged to complete all the *Quizs* that he/she could not finish during the lab session.

Bibliography

[1] *P. Pavón Mariño, "Optimization of computer networks. Modeling and algorithms. A hands-on approach", Wiley 2016.*