Universidad Politécnica de Cartagena


TEORÍA DE REDES DE TELECOMUNICACIONES

Grado en Ingeniería Telemática
Grado en Ingeniería en Sistemas de Telecomunicación

Curso 2015-2016

# Lab work #9. Node location problems

**(1 session)**

*Author:*

Pablo Pavón Mariño

# 1 Objectives

The goals of this lab work are:

1. Create Net2Plan algorithms that solve formulations of variants of the node location problem using the *Java Optimization Modeler* (JOM) library.
2. Gain experience with the different forms of writing optimization problems in JOM, making benefit of its vectorial representation capabilities.

# 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 7.2 of [1], and the lecture notes regarding node location problems.
- Refresh your reading in the JOM documentation in `http://www.net2plan.com/jom`.

# 6 Node location problem

Let $\mathcal{N}$ be a set of locations. In each location, one *access node* exists, an entity producing traffic. Each location can host also zero or one core nodes. Each access node should be connected to exactly one core node. Each core node can be connected to at most $K$ access nodes. We denote as $c_{ij}$ to the cost of connecting an access node in location $i$, to a core node in location $j$. We denote as $C$ to the cost of each core node (wherever it is placed).

We are interested in determining the placement of core nodes in the network, and how the access nodes are connected to them, such that the total network cost is minimum. The problem is formulated as follows:

- Input parameters (known constants):
  - $\mathcal{N}$: Set of locations.
  - $c_{ij}, i, j \in \mathcal{N}$: Cost of connecting an access node in $i$ to a core node in $j$.
  - $C$: Cost of a core node.
  - $K$: Maximum number of access nodes that can be connected to the same core node.
- Decision variables:
  - $z_j, j \in \mathcal{N}$: 1 if a core node is placed in location $j$, 0 if not.
  - $e_{ij}, i, jin\mathcal{N}$: 1 if access node in location $i$ is connected to core node in location $j$, 0 if not.
- Formulation:

$$\min \quad C\sum_j z_j + \sum_{ij} z_{ij}e_{ij}, \quad \text{subject to:} \tag{1a}$$

$$\sum_j e_{ij} = 1, \quad \forall i \in \mathcal{N} \tag{1b}$$

$$\sum_i e_{ij} \leq K z_j, \quad \forall j \in \mathcal{N} \tag{1c}$$

The objective function (1a) sums the network cost (core nodes and access-core links). Constraints (1b) make that each access node is connected to exactly one core node. Finally, (1c) make that if a location has zero core nodes ($z_j = 0$), then no access node can be connected to it. If the location has a core node ($z_j = 1$), then at most $K$ access nodes can be connected to it.

# 7 Net2Plan algorithm

The student should develop a Net2Plan algorithm solving problem (1). The algorithm will receive an input design with nodes. These nodes are assumed to be the locations of the access nodes, which are at the same time the potential locations of the core nodes. It should produce a design that reflects where the core nodes are placed, and how they are connected to the access nodes. To develop such algorithm, follow the next steps:

1. Copy the `AlgorithmTemplate.java` file in Aula Virtual and rename it as `NodeLocation.java`.
2. The algorithm should have one input parameter named `C`, with default value 10. This is the $C$ constant in (1). Also, it should have an input parameter named `K`, with default value 5. This is the $K$ constant in (1).
3. Remove all the links in the network.
4. Create an object of the type `OptimizationProblem` (e.g. of name `op`).
5. Add the decision variables with name `z_j`: one variable per node in the input design. The $j$-th coordinate corresponds to the `Node` object of index $j$.

6. Add the decision variables with name `e_ij`: a 2D array of variables with one variable per node pair in the input design. The $(i, j)$-th coordinate corresponds to the existence or not of a connection from the (access) `Node` of index $i$ to the (core) `Node` in location of index $j$.

7. Set the following JOM input parameters:

   - User-defined parameter `C`.
   - User-defined parameter `K`.
   - We consider that the $c_{ij}$ costs are equal to the Euclidean distance between $(i, j)$ locations. Use the method:

     `getMatrixNode2NodeEuclideanDistance`

     of the `NetPlan` object to obtain a matrix with as many rows and columns as nodes, and the $(i, j)$ coordinate providing the Euclidean distance between nodes of indexes $(i, j)$.

8. Set the problem objective function. Recall that the JOM operator `.*` can be used to make an element-by-element multiplication of two arrays, and that JOM function `sum` (without arguments), sums all the elements of an array.

9. Use a `for` loop with as many iterations as nodes, to add the constraints (1b). Set a JOM input parameter of name `i` inside the loop, with the index of the access node location. Use JOM function `sum` over $i$-th row of array `e_ij`.

10. Use a `for` loop with as many iterations as nodes, to add the constraints (1c). Set a JOM input parameter of name `j` inside the loop, with the index of the potential core node location. Use JOM function `sum` over $j$-th column of array `e_ij`.

11. Call the solver to find a numerical solution. Since the problem is linear with integer constraints, the solver to use is `glpk`.

12. Retrieve the primal solution obtained.

13. Save the access-to-core links in the design (not as bidirectional links, so do not add the links core-to-access). All the links are of capacity 1, have a distance equal to the Euclidean distance between the end nodes (use the method `getNodePairEuclideanDistance`), and a propagation speed of 200000 km/s.

14. Return a message containing (i) the number of core nodes placed, (ii) the total cost of the design returned (use `getOptimalCost` method of the `OptimizationProblem` object for that).

## 7.1 Check the algorithm

Load the network `NSFNet_N14_E42.n2p`. The algorithm should produce a solution with a total of 6 core nodes, and a cost of 103.8 units.

**Quiz 1.** Fill in the following table, running the algorithm in `NSFNet_N14_E42.n2p` topology, for different values of `K` (maximum connectivity) and `C` (core node cost) parameters.

NSFNET topology test

| `C` | # core nodes for `K=5` | # core nodes for `K=14` |
|---|---|---|
| 0 | 14 | 14 |
| 5 | 11 | 11 |
| 10 | 6 | 6 |
| 12 | 5 | 4 |
| 15 | 4 | 3 |
| 20 | 3 | 3 |
| 1000 | 3 | 3 |

Answer the following questions:

- How does the number of nodes is affected by $K$?

  *Answer*: Lower $K$ make that in some occasions more core nodes are needed, since the existing ones are already congested (connected to $K$ access nodes).

- Why the number of core nodes in `K=5` does not go below 3?

  *Answer*: Because then I need at least 3 core nodes to connected to 14 access nodes.

- Why the number of core nodes in non-increasing with $C$?

  *Answer*: Because a higher cost of the core node is never an incentive to buy more of them.

## 8 Problem variations

**Quiz 2.** Modify the algorithm by adding the constraint that the maximum number of core nodes available is limited to a user-defined parameter `M`. The constraint to add to (1) is:

$$\sum_j z_j \leq M$$

Note that if $M < N/K$ the problem has no solution, since there is not enough core nodes available to connect all the access nodes.

## 9 Matricial form of problem constraints (optional)

Let us assume that decision variables $z_j$ are defined in JOM as a $^{\text{o}} \times |\mathcal{N}|$ row vector, and $e_{ij}$ as a $|\mathcal{N}| \times |\mathcal{N}|$ matrix.

Then, according to the JOM syntax:

- `sum (e_ij , 2)` sums along the second dimension of `e_ij` (the columns). Then, it produces a column vector ($|\mathcal{N}| \times 1$), where the $i$-th coordinate is given by $\sum_j e_{ij}$.
- `sum (e_ij , 1)` sums along the first dimension of `e_ij` (the rows). Then, it produces a row vector ($1 \times |\mathcal{N}|$), where the $j$-th coordinate is given by $\sum_i e_{ij}$.

Acoording to this, constraints (1b) can be introduced in one single call to `addConstraint` method, as a vectorial contraint as follows:

```
sum (e_ij , 2) == 1
```

Also, constraints (1c) can be set in one single call to `addConstraint` method, as a vectorial contraint as follows:

```
sum (e_ij , 1) <= K * z_j
```

**Quiz 3.** Rewrite the algorithm 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.*