Universidad Politécnica de Cartagena

PLANIFICACIÓN Y GESTIÓN DE REDES

Grado en Ingeniería Telemática
Course 2019-2020

Case of study. Heuristic algorithms for a node location problem variant

Autor:

Pablo Pavón Mariño

Introduction

In this case of study students should develop a Net2Plan algorithm for solving a variant of the node location problem, described below.

Let $i = 1, \ldots, N$ a set of locations for a node. In each location i:

- There *always* exists exactly one access node.
- There can be any number $z_i = 0, 1, 2, \ldots$ of core nodes, placed in that location.

Each access node must be connected to *exactly two* core nodes in different locations (one of them can be the same location as the access node). The cost of each access link is equal to the Euclidean distance between the locations of the access and core node.

Each core node has a limited capacity, and because of this, each core node can be connected to a maximum of M access nodes.

The problem to solve is.

- Given the set of locations of the access nodes.
- Find for each location i how many core nodes z_i are placed in there. Additionally, for each location i, indicate the two core nodes to which it is connected to.
- The solution found should minimize the total network cost $TotalCost$ computed as the sum of (i) the access link costs, (ii) the cost of the core nodes:

$$TotalCost = \sum_e d_e + C \sum_i z_i$$

We use e to denote the index that ranges all the access links in the network. The access links' costs $\sum_e d_e$ is assumed to be the sum of the **lengths** of the access links. The cost of the core nodes to place $C \sum_i z_i$, is given by the multiplication of the cost of one core node (C), and the total number of core nodes placed ($\sum_i z_i$).

Implementation in Net2Plan

The students must individually develop a Net2Plan algorithm that solves the described problem.

- The algorithm will be applied on input Net2Plan designs, with no links, and just nodes. Each node in the input design is a *location* where exactly one access node exists, and that would be able to host an arbitrary number of core nodes.
- The algorithm must accept the following input parameters:
  - `maxExecTimeSecs` (default value 60): Maximum running time (in seconds) of the algorithm. The algorithm must implement internally the procedure to stop the algorithm after, at most, this amount of time[1]. When stopped, the algorithm should return the best solution found so far.

[1] To measure the running time, use the Java method `System.nanoTime`

  - `M` (default value 5): Maximum number of access nodes that can be connceted to a single core node.
  - `C` (default value 100): The cost of a core node.

The cost of the link between an access node in location i and a core node in location j (where i and j can be the same location), is given by the Euclidean distance between the two locations, as provided by the method `getNodePairEuclideanDistance` in class `NetPlan`. Note that if a location has one or more core nodes, the best choice is to make the access node in that location to connect to any of them, since this adds zero cost for the access-to-core link.

The output of the algorithm is returned by just adding to the input design the access links, which are **unidirectional**, from the access node location, to the core node location. Important considerations:

- When in the access link, the core node is in the same location as the access node, no link is added to the `NetPlan` object.
- There is no need to add to the design the core nodes, since this information is automatically derived.
  - If a `NetPlan` node has only one outgoing link, this means that the access node in it is connected to i) a core node in the same location, ii) the core node of the outgoing link.
  - The number of core nodes in a location is implicitly derived by counting the number of input links entering the node, plus potentially one more if the node has one output link. Then, the number of core nodes is given by the number of access links, divided by `M` (rounded up).

The output design should minimize the total network cost, while satisfying all the problem constraints.

> **IMPORTANT**: The videos uploaded to UPCTmedia describing the use case have slight differences with what is expected from the students. In particular, the default values of parameters `M` and `C` are different, and correspond to an outdated version. The correct values are the ones shown in THIS document. The template provided to the student has the correct values.

Net2Plan template

To assist the development of the algorithm, the student is provided with a `.java` template of the algorithm[2]. The template will include:

- A general skeleton of the algorithm.
- The code for reading the input attributes.
- A function called `evaluateDesign`, that receives a NetPlan object as an input, and the value of the `C` and `M` parameters. The function makes the following actions:
  - Check that the design is valid from a formal point of view, e.g. no node has more than one output link (since access nodes can only connect to one core node).

[2]The file is placed in Aula Virtual with name `NodeLocation_TEMPLATE.java`

– Returns the total cost of the design, an the number of core nodes of the solution[3].

Students are encouraged to reuse the provided template.

Additionally, the students are encouraged to follow the videos of the case of study, where the teacher will show how to setup the environment to program and test Net2Plan algorithms, show some algorithmic options, and comment on best practices for a clean and well structured coding.

IMPORTANT: The videos uploaded to UPCTmedia describing the use case have slight differences with what is expected from the students. In particular, the default values of parameters `M` and `C` are different, and correspond to an outdated version. The correct values are the ones shown in THIS document. The template provided to the student has the correct values.

Delivery

The student should develop, reusing the provided template, a Net2Plan algorithm that solves the node location problem described. The student is free to choose any combination of the mathematical techniques and heuristics described in the course (e.g. tabu search, GRASP, ant-colony, evolutionary algorithm, ...).

This lab work should be solved individually. The students must produce **one** single `.java` file with name `NodeLocation_XXXX.java`, where XXXX is the first and second surnames of the student[4].

The students should send by email to `pablo.pavon@upct.es` the Java file, with deadline May 28th, at 23:59. The work reception time will be taken as the time of reception of the email. Each hour of delay in the delivery is penalized with one point (over 10) in the grade.

Students do not have to send any other material than the Java file. It is not allowed to spread the code in several Java files. If the student needs to define more than one class, they should be integrated in the single Java file to run.

Evaluation

The evaluation of the case of study is performed as follows:

1. 60%: Average cost obtained in a the sample topology that will be available in Aula Virtual (`NodeLocationSampleTopology.n2p`), tested with different values of the parameter `M` (`M = 3, M = 5`) different values of `C` (`C = 20, C = 100`), and a running time of 60 seconds in all the cases. The costs obtained by the different students will be ranked, and the grade received will be dependent on the average ranking among the tests. Algorithms failing to produce a valid solution (that does not pass the `evaluateDesign` method), will be heavily penalized. Algorithms for which the running time exceeds the `maxExecTimeSecs` limit in more than three seconds, will be also penalized.

[3]For this, the function returns a `Pair` object, with the two mentioned values.

[4]No special characters like 'ñ' or accents can be used, since this violates Java file naming conventions.

IMPORTANT: The videos uploaded to UPCTmedia describing the use case have slight differences with what is expected from the students. In particular, the values of parameters `M` and `C` for which the tests will be made are different, and correspond to an outdated version. The correct values are the ones shown in THIS document. Also, the videos show a topology of 200 nodes, while the tests will be made in the topology of 50 nodes provided.

2. 40%: Creativity, technical depth of the algorithm, as well as clarity, adherence to recommended practices in Java programming and cleanness in the code. A more structured code, with an adequate amount of comments (not too much and not too few!) is preferred. Please follow the indications of the teacher in the lab videos.

The complete evaluation will be modulated by a personal online interview that each student will have with the teacher on June 4th, during the last lab session (from 9am to 3pm). The specific slot time and procedure for the online interview for each person will be informed in Aula Virtual on June 2nd.

Just as an indication, the table below shows close to optimal cost values for the parameters of interest. For those students obtaining better values, congratulations!! ;-)

Orientative close-to-optimal costs

Cost	`M=3`	`M=5`
`C=20`	1334.19	1307.25
`C=100`	4065.1	3000.83