Computer Science and Engineering
Foundation Of Operation Research - Prof. Malucelli Federico

Heuristic Algorithm to Optimize Market Installation and Refurbish

Marco Di Gennaro
Francesco Corso

Contents

1. Algorithm

To Implement the solution of the problem we split it into two sub-problems:

1. Minimize the markets opening cost;
2. Minimize the cost of refurbishing the markets starting from the depot(the market located in the first village).

We solved the first problem by formalizing a model to find an optimal solution, while the second one, based on the output data of the first problem, is solved with a heuristic approach.

1.1 First Problem Solution

The Model to solve the first problem has been formalized in AMPL, you can find the implementation in the file `10596841_10867595_opening.mod`, to connect it with the second problem we used the `amplpy` python library. This library allows solving a problem, formalized as an ampl model, directly in the python environment and to insert, in ampl, input data and obtain output data.

1.2 Second Problem Solution

Once obtained the open markets through the optimization of the first problem, with the relative value of the objective function, we passed to the resolution of the second problem. To facilitate the subsequent solution, we chose to represent the markets as nodes of a dense graph, each market is connected, through arcs, to all other markets, and the weight of the arc is given by the distance between the two markets multiplied by the cost in km given as input to the main problem. To represent the graph described we used the `networkx` python library.

The first approximation made in solving the problem is given by the choice of the number of trucks to use, this is fixed upstream by the following function:

$trucks = \lceil number\_of\_markets/capacity \rceil$

Once done this approximation, we used a heuristic algorithm to obtain the trucks' path.

The Heuristic implemented is based on a local search algorithm, at the base of which, solutions are identified through a greedy approach.

1.2.1 Greedy Algorithm to find solution

The first thing the algorithm do is to split nodes in two zones. One zone is identified as the zone north of the depot, while the other is south of it.

The algorithm builds the solution one truck at a time. To do so it starts from the first truck and the node representing the *depot*. It calculates a priority score for each *reachable* market from the depot ("reachable" means a node that has not already been supplied). This score is the cost of the arc from depot to the market in the graph. The scores calculated are normalized (value from 0 to 1) and sorted with ascending criteria (in our solution the lower score is, better is the solution).

The next node the truck has to refurbish is chosen as the node number *first_ step* in the sorted score list. Where the *first_ step* is chosen in the local search part of the algorithm. The choice of the node number *first_ step* in the sorted score list is carried out **only** for the first movement of the first truck. The first movement of the following trucks will be chosen through the node with the lowest score.

After this first step, the algorithm moves to analyze the node chosen, mark it as supplied, and does the same thing done with the depot. The difference from the score calculate starting from a depot and non-depot market is that:

- If the truck has already refurbished at least $capacity - magnet$ nodes, the score, before the normalization, is calculated as the mean between the cost of arc($current\_market$, $reachable\_market$) and cost of arc($depot$,$reachable\_market$). The value of *magnet* is chosen in the local search part of the algorithm and can be interpreted as "how close the truck route must keep to the depot";
- After the normalization, each reachable market is penalized ($score = score + penalty$) if the current market isn't in the same zone of the reachable market, the value of *penalty* is chosen in the local search part of the algorithm.

The next node the truck has to refurbish is chosen as the first node in the sorted score list.

The last action is repeated until the truck has supplied $capacity - capacity\_reducer$ nodes. After it comes back to the depot and a path for another truck is searched. The value of*capacity_ reducer* is chosen in the local search part of the algorithm. The Greedy algorithm stops his execution when all trucks' path is defined and all node have been supplied.

1.2.2 Local Search Algorithm to find better solution

The core of the algorithm is represented by the previously described greedy approach. However, using some values for the parameters in the greedy algorithm, rather than others, can make a difference in the final solution. We have decided, therefore, not to fix those parameters but, to modify them from iteration to iteration in order to move in a certain space of solutions and to maintain from time to time only the best solution. The previously presented greedy algorithm is, therefore, executed by varying:

- *first_ step*: for this parameter, all the choices are tried. This parameter allows to try, for each iteration, a new node from which to start building the solution, starting from the nearest to the farthest;
- *capacity_ reducer*: the algorithm uses this parameter when in the solution there will surely be a truck that will not exhaust all its capacity. We use this parameter because, building the final solution truck by truck, if we fill all the first trucks and then fill the final one with the rest, we would lose a large solution space. This parameter is calculated so that we never get unfeasible solutions;
- *magnet*: this parameter can be interpreted as "how close the truck route must keep to the depot". So, starting from 0 we find all possible values up to half of capacity. This parameter is used to ensure that, from a certain point onwards of the route (usually in the final nodes, hence the choice to try only up to capacity/2), the truck can approach the depot in anticipation of the return;
- *penalty*: this parameter is used to encourage truck movement in the same area rather than another area. The value starts its search from a range between 0 and 0.5, and then gradually refines until it becomes fixed. The algorithm will then tend to speed up its execution as iterations pass.

2. Running Instructions

In the delivery folder is provided a python script called `heuristic.py` and an AMPL model called `opening_model.mod`, and the two result text files. To execute the algorithm you have to run the python script. To run the script, installation of AMPL and python is required. Before running the python script you need to open the script and replace the value of the constant `AMPL_INSTALLATION_PATH` in line 12 with the absolute AMPL installation path. You will also need to install, via pip, the libraries used in the script, including `amplpy`, `numpy`, `matplotlib`, `scipy` and `networkx`.

The script executes both the first instance and the second, it writes each result in a file called solution-instancename in the results folder. It is possible to plot the results of the problem running the script with the argument `-plot`.