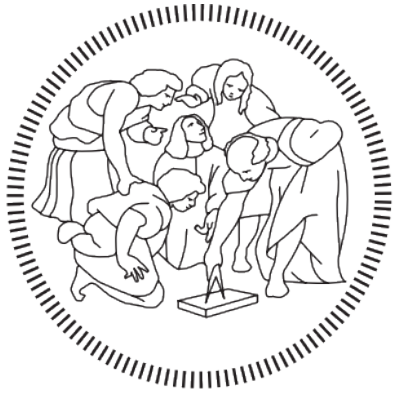
FOUNDATIONS OF OPERATIONS RESEARCH PROJECT REPORT 2021/2022



POLITECNICO
MILANO 1863

Prof: Federico Malucelli
Tutor: Tommaso Schettini

Authors:
Barbieri Fabio / 10567083 / 953470
Chiari Giuseppe / 10576799 / 964770
Conti Alessandro / 10669820 / 996848

Indice

1 Introduction to the project

This file contains a short description of an algorithm, which was built from scratch by the components of our group, to solve the assigned problem, without taking inspiration from any other existing algorithm and following the approach for heuristic algorithms design learnt during FOR classes.

Problem's parameters

- *n locations* identified by these parameters
  - *Cx*, the x coordinate
  - *Cy*, the y coordinate
  - *usable*, representing the possibility of building the market in that specific location
  - *Dc*, cost of building the market in that specific location
- *Fc*, "Fixed cost", the cost which must be payed for each truck
- *Vc*, "Variable cost", the cost that must be payed for each kilometer travelled by a truck
- *capacity*, how many stores can be visited (at most) by each truck
- *range*, maximum distance at which, for each location, can be found at least a market

The goal of the project is to solve a planning problem:

- We must choose in which of the *n locations* (geometrically identified by two coordinates) we should build a market
- We must plan some routes (starting from node 1 and ending at node 1) such that each market is visited at most once

We must respect some additional constraints:

- For each one of the *n locations* it must exist a market which is distant at most *range* kilometers by the location
- Markets can be built only in locations in which *usable = 1*
- Each route can visit at most *capacity* markets
- A market is always built at location 1

We must minimize :

- *Total installation cost*, sum of all the *Dc* related to locations in which a market is built
- *Total refurbishing cost* , sum of all the *refurbishing cost* for each truck, the *refurbishing cost* is given by the sum of *Fc "the fixed cost"* and the total kilometers travelled by the truck multiplied by *Vc "Variable Cost"*

2 Underlying assumptions

In order to simplify a bit the problem, after deeply analyzing the data available for the 2 problem instances *(minimart-I-50.dat , minimart-I-100.dat)*, we've made some reasonable assumptions (which obviously may cause the loss of the optimal solution and of some other good solutions).

- We can decouple the main problem into 2 sub-problems:
  - Find the market network
  - Starting from the market network solve the VRP (Vehicle Routing Problem) in order to plan the routes

  The decoupling will cause: *min (Total installation cost + Total refurbishing cost)* to be simplified as *min (Total installation cost) + min(Total refurbishing cost)*
- For each truck (except the last one) is always convenient to visit exactly *capacity* markets
- It is convenient to design "circular" routes (see section 3.2)

These are the guidelines that helped us in designing our algorithm. In particular, the last two assumptions hold in the case that we identify as the ideal one, which is the one that our algorithm is trying to approximate.

3 Algorithm

As mentioned before the problem is decoupled into 2 sub-problems, because of this the algorithm can be divided into 2 main sub-algorithms

- Sub-Algorithm-1, which manages the finding of the market network
- Sub-Algorithm-2, which manages the planning of the routes

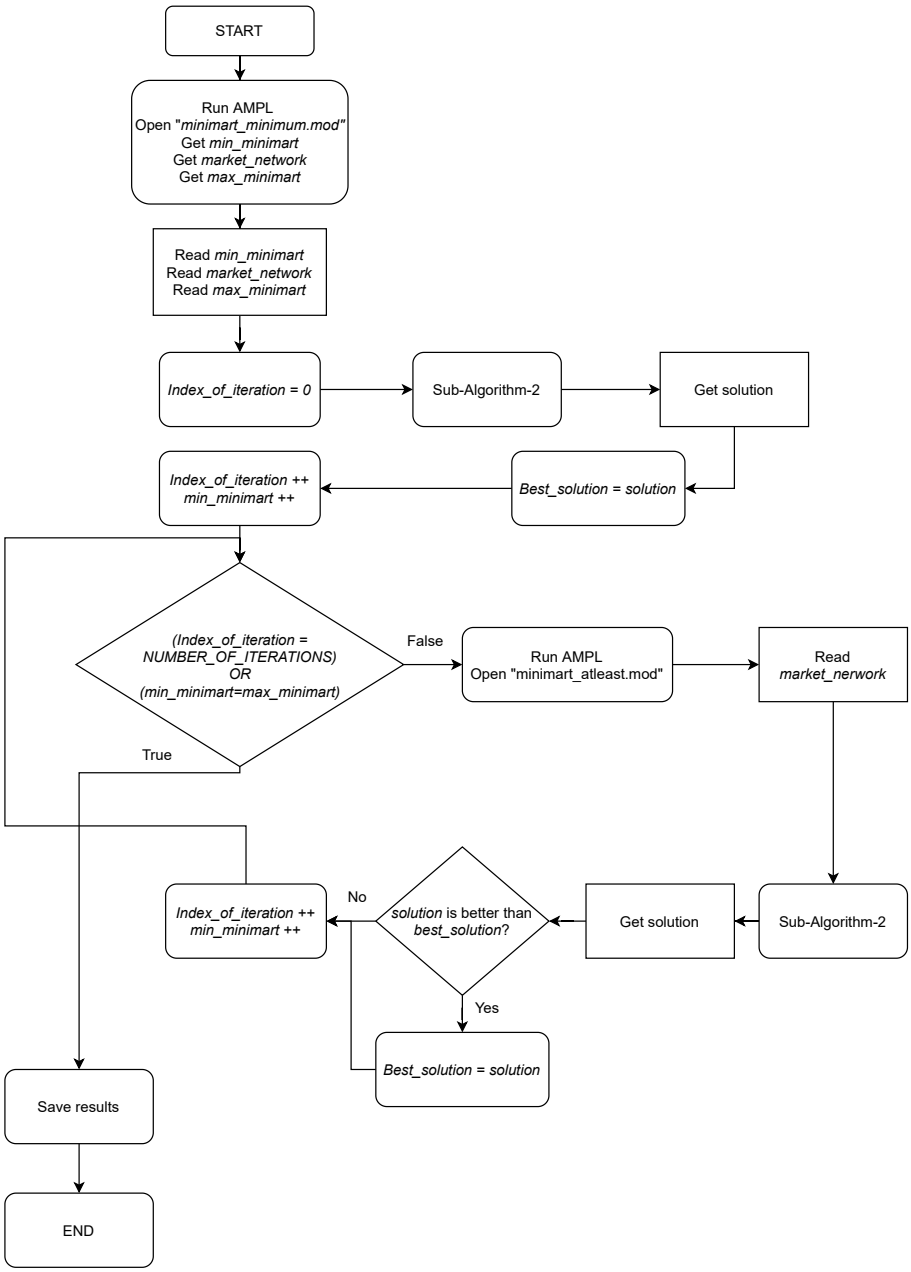
The fixed starting point of the second sub-problem is the solution of the first sub-problem, modified by adding a constraint that enforces the minimization of the number of built markets while satisfying the reachability constraint.

3.1 Sub-Algorithm-1

The Sub-Algorithm-1 is run for each dat file in the *config.DATFILES* array. These are the main steps:

- Run the *minimart_minimum.mod* file through AMPL in order to get the market network with the minimum number of markets installed regardless of the cost of installation
  We also get the *min_minimart* and the *max_minimart* parameters (which will be set to the count of the markets satisfying the clause : *usable >1*)
- Get the routes plan by calling the *Sub-Algorithm-2* and store it in the *best_solution* variable
- If *NUMBER_OF_ITERATIONS >1*, increment *min_minimart* by one and run *minimart_atleast.mod* file through AMPL, it will return the optimal solution (minimum installation cost) of the market network in which there must be exactly *min_minimart nodes*, then get the routes plan by calling the *Sub-Algorithm-2* and eventually assign it to the *best_solution* variable

- The operations written above are inserted into a loop which cab be iterated at most *NUMBER_OF_ITERATIONS* times (it depends on the condition *min_minimart>max_minimart*) By doing this we're trying to solve the VRP starting by different network market configurations (for each iteration we increase by one the number of markets composing the network) and obviously we are keeping track only of the best solution.



3.2 Sub-Algorithm-2

The *Sub-Algorithm-2* essentially perform *NUM_ITERATIONS_VRP* times the VRP algorithm, it obviously keeps track only of the best solution.
Now we're going to discuss how the VRP is implemented.
The main idea is that, in order to reduce travelling costs, each route (truck) must visit (if it isn't the last one) exactly *capacity* markets, the distance between two adjacent nodes of the route should be reasonable and the path of each routes should be something "circular", which means that the truck should optimize the route by correctly combining the round and the return paths.

3.3 Node Ordering and clustering

The first thing that the algorithm performs, after having found a solution (via AMPL) to the first sub-problem, is to divide the nodes in groups in the following way:

1. The *minimum number of trucks* to be used in the routes planning is defined as: $\lceil \frac{number\_of\_nodes}{capacity} \rceil$
2. Each group contains the same number of nodes, let us call it *cardinality*, defined as: $2 \times capacity$
3. The first cluster contains exactly the *cardinality* nodes that are nearest to the market known as *main branch*, which is the one in the first location.
4. The centroid of the first group is computed as the point with coordinates equal to the average of the coordinates of all the points contained in it.
5. The second cluster will contain exactly the *cardinality* nodes nearest to first group's centroid (and consequentially, in average, nearest to all the first group's nodes).
6. By iterating steps 4 and 5, nodes are divided into well distinct groups, as desired.

The last group of nodes will contain *cardinality* nodes only if the total number of nodes is a multiple of *cardinality*. If this condition is not satisfied, the last group will contain less nodes than the other groups.

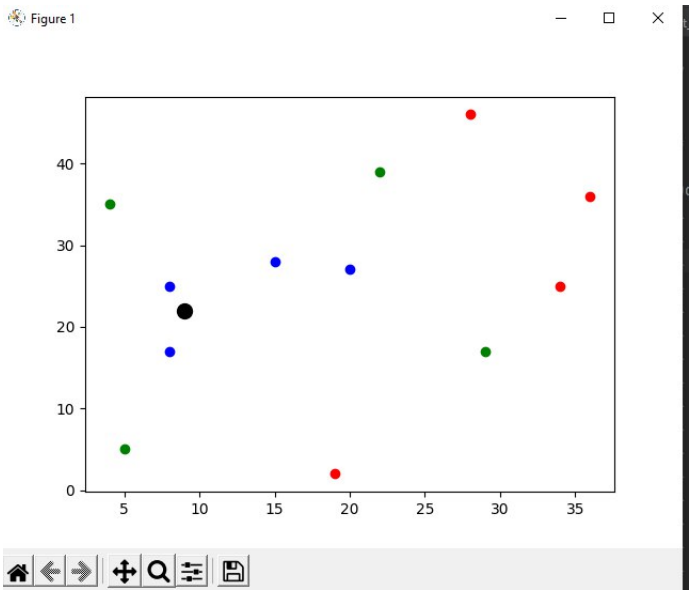


Figura 1: `Example of clustering of a market network`

3.4 Routes Planning

Once nodes are ordered, the algorithms tries to find the best routes to connect them all.
The main idea is to construct in parallel all the routes, for each step of the route we try to reach a market in the next cluster, if we are in the last cluster we will search in the previous cluster in order to start constructing the return path to the *main branch*. The used approach is a greedy/random one and it is articulated in the following phases:

1. The first thing to do is to add to each route the *main branch* node as starting node.
2. Each route will be expanded by adding to it a node from the first cluster. The order in which the routes are expanded is randomized (this is the cause of the randomize approach and the consequent needing of the multiple execution of the VRP algorithm).
3. Once every route has (exactly) one node belonging to the first cluster, the algorithm will repeat step 2 with the second cluster and with every other cluster until the last one is reached.
4. Still following a randomized order, each route adds another node of the last cluster if there are left.
5. The expansion of each route will continue as explained but going from the last one cluster to the first one.
6. Finally the *main branch* is added to each route as their last node.

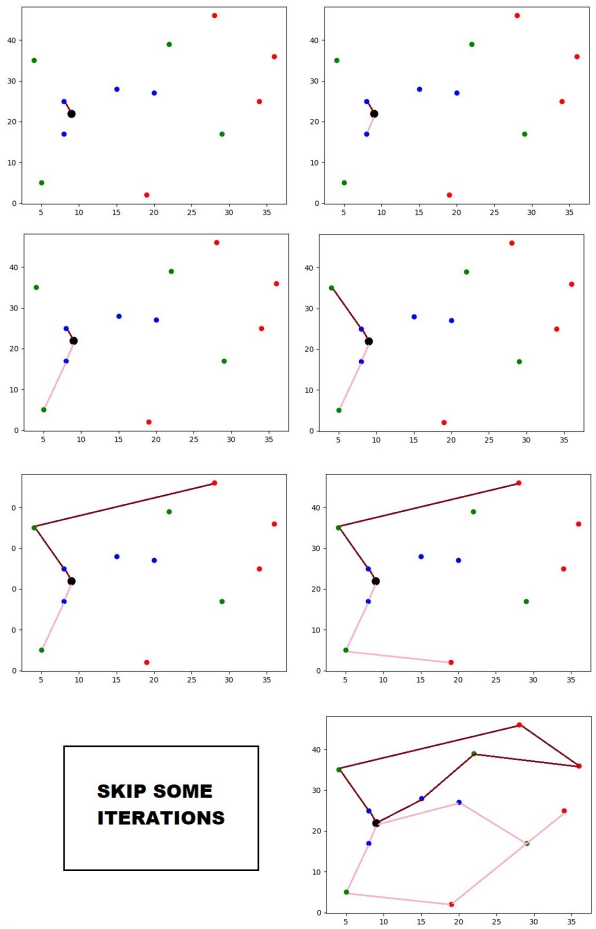


Figura 2: `Example of VRP execution`

4 Code parameters

Parameters that can be set into *config.py* file

- *DATFILES*, list of string containing all the input dat files
- *PATH_TO_AMPL_EXECUTABLE*, path to the AMPL executable
- *NUMBER_OF_ITERATIONS*, how many (maximum) different network configurations should be considered for each problem instance
- *FRACTIONS_OF_MINIMART*, it is a scale parameter useful to regulate variable controlling the *min_minimart - max_minimart* condition which eventually stop the *Sub-Algorithm-1* loop.
- *NUM_ITERATIONS_VRP*, for each different network configuration how many times must be executed the VRP algorithm
- *PLOT_SOLUTION*, if *TRUE* solutions are displayed on screen or saved as png.
- *SAVEFIG*, if *TRUE* for each instance of the problem will be created an image representing the routes, if *FALSE* the image will be printed in a separated window.

5 Instruction for code usage

After having correctly set *PATH_TO_AMPL_EXECUTABLE* (remember it must be an absolute path), you must also set the CPLEX path inside both *minimart_atleast.mod* , *minimart_minnum.mod* (it can be a relative path).
If another dat file needs to be tested it must be added inside the project folder and its name must be added inside the *DATFILES* which can be found in *config.py*. The file to be executed is *big_project.py*
The solution files are created inside the project folder.
If you don't want to see results you must set *PLOT_SOLUTION* to *FALSE* otherwise if it is set to *TRUE* solutions will be displayed on screen or saved as png (depending upon *SAVEFIG*).

6 Python libraries used

The following libraries have been used to implement the algorithm and are required to be able to run the code:

- numpy
- pandas
- amplpy
- matplotlib

7 Our results

Since our program in the VRP section has a stochastic behaviour we will print our best execution results for both *minimart-I-50.dat* , *minimart-I-100.dat* instances.

minimart-I-50.dat results:

389.2476132712119
87.0
302.2476132712119
1,34,3,26,33,43,36,31,22,12,29,42,16,17
1,3,22,29,17,12,31,33,1
1,34,43,16,42,36,26,1

minimart-I-100.dat results:

1087.3060509266413
221.0
866.3060509266413
1,98,4,86,83,38,50,8,37,90,5,60,87,62,74,26,21,23,84,43,47,27,40,56,19,31,18,36,75,64,100,44,82,17,51,85,14,33,16,68
1,98,74,56,64,33,68,31,43,62,38,1
1,83,60,47,44,16,14,82,23,5,8,1
1,86,90,84,18,51,17,36,19,21,37,1
1,4,87,40,100,85,75,27,26,50,1

8 Links to external resources

- Numpy library
- Pandas library
- Matplotlib library
- Amplpy library