RoboMarkt Report

To solve the assigned problem, we've decided to split it into two parts: the first one consists in finding where to install the stores and the second one consists in finding the routes used by the trucks to refill them periodically while minimizing the total costs defined as follows:

$$totalCosts = buildingCosts + drivingCosts$$

Part 1: Installation problem

We've modeled this sub-problem as a binary linear programming problem as follows, and we've solved it using AMPL and CPLEX

$$buildingCosts = \ min \sum_{i \in N} Dc[i] * y[i]$$

$$Subject\ to: \quad \sum_{j \in N:\, distance[i,j] \leq range} y[j] \geq 1 \qquad \forall i \in N \quad \text{(each point have a store within the range)}$$

$$y[i] = 0 \qquad \forall i \in N: usable[i] = 0 \quad \text{(non-usable points cannot be used)}$$

$$y[1] = 1 \qquad \text{(depot must be built)}$$

where *N* is the set of the small villages, *Dc[i]* is the cost of building a store at village *i*, *distance[i,j]* is the Euclidian distance between the villages *i* and *j* and *range* is the maximum feasible distance between a store and a village.

The binary *y* variable will tell us which stores to build so that each client can find a store in *range* kilometers while minimizing the building costs.

Part 2: Refurbishing routing problem

This sub-problem is a variation of the known Vehicle Routing Problem (VRP) where, in this case, each store to refurbish (the ones found in Part 1) has a unity demand and each truck has the same capacity.
Determining the optimal solution to a VRP problem is known being NP-hard[1] thus a heuristic method is needed to have a polynomial computation time. After studying some papers about VRP, we've decided to implement the idea of a greedy algorithm called the *Savings Algorithm*[2] which is based on the concept of "saving" the number of kilometers needed to visit two points *i, j* starting from an origin point (the *depot*, assumed to be the point number 1). The heuristic function of our algorithm is:

$$saving(i,j) = 2 * d\_cost(1,i) + 2 * d\_cost(1,j) - (d\_cost(1,i) + d\_cost(i,j) + d\_cost(j,1))$$

[1] https://doi.org/10.1016/S0304-0208(08)73235-3
[2] https://neo.lcc.uma.es/vrp/solution-methods/heuristics/savings-algorithms/

$$= d\_cost(1,i) + d\_cost(1,j) - d\_cost(i,j)$$

and it means ''how much'' it is worth to travel directly to *j* right after having visited *i*, without first returning to the depot.[3] The $d\_cost(i,j)$ is calculated as $dist(i,j) * Vc$ in general and as $dist(i,j) * Vc + Fc$ when *i=1,* and it describes the cost of travelling between two stores.

Our implemented algorithm follows the greedy paradigm[4], thus it can be described as follows:

$routes \leftarrow (1,i,1)$ $\quad \forall i$ in $stores$ //a route is a list of stores saved in the order they are visited
$S \leftarrow saving(i,j)$ $\quad \forall (i,j)$ in $stores$
repeat:
    $currSaving \leftarrow Best(S)$
    $S \leftarrow S \setminus \{currSaving\}$
    if $Ind(routes, currSaving)$ then:
        $newRoute = mergeRoutesUsing(currSaving)$
        $routes \leftarrow routes \setminus \{r1, r2\} \cup \{newRoute\}$
until $S \leftarrow \emptyset$

where:

- *Best(S)* at the first iteration sorts *S* in decreasing order, and simply return the first element (the greatest one) removing it from *S*
- *Ind(routes, currSaving)* is true if and only if there are two routes in *routes* that can be merged exploiting the *currSaving,* checking also if the truck capacity isn't exceeded.
- *mergeRoutesUsing(currSaving):* merges the two routes *r1* and *r2* that pass through *s1* and *s2* (the points of *currSaving*). These two routes are found by another function which ensures that they are such that *r1=(1, … , s1 ,1)* and *r2=(1, s2 , … , 1).* It also checks the reversed order of *r1* and *r2* given that the *routes* can be considered as undirected. The merged route is *newRoute=(1,… , s1, s2, …, 1).*

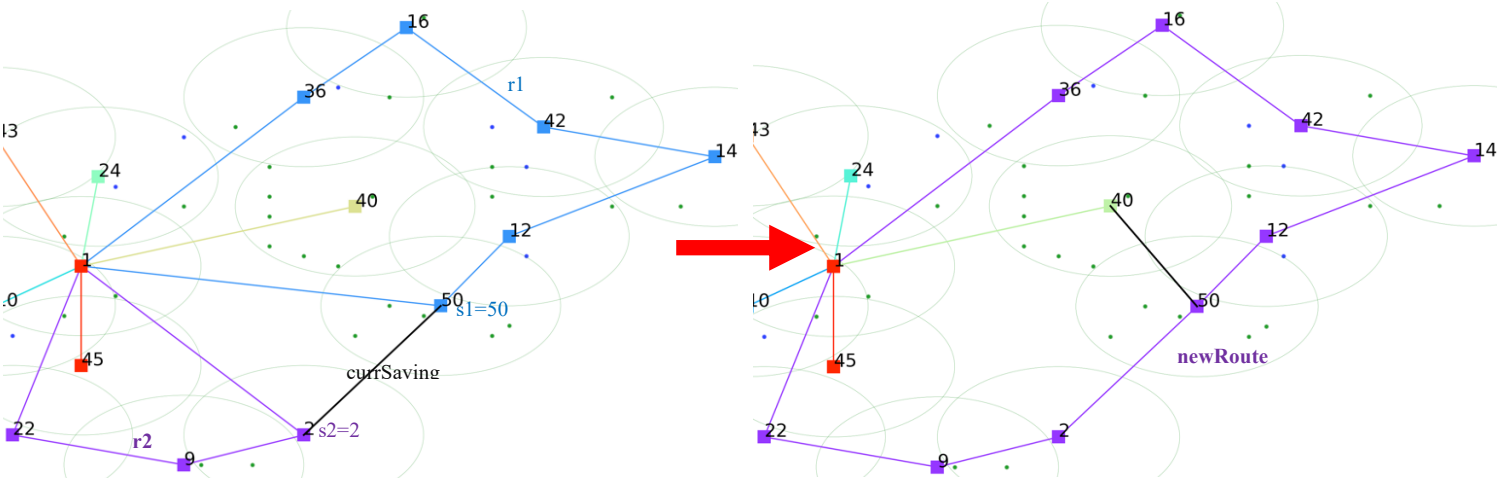

Figure 1: Routes before and after a merging

[3] https://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html
[4] Federico Malucelli, Appunti di introduzione alla Ricerca Operativa

Conclusions

The proposed algorithm tries to minimize the total cost for building the stores and refurbishing them.

The cost for building the *stores* is computed by AMPL from our description of the problem in Part 1, so we are sure that the provided result is optimal. Instead, the cost of filling the *stores*, $drivingCosts$, is calculated as follows:

$$drivingCosts = \sum_{r \in routes} \sum_{(i,j) \in r} d\_cost(i,j)$$

which is just an estimation recalling the fact that the implemented *Savings Algorithm* find a routing configuration that might not be optimal.

We've plotted using another script our solution to a provided instance of the problem and, as shown in *Figure 2*, it can be considered acceptable.

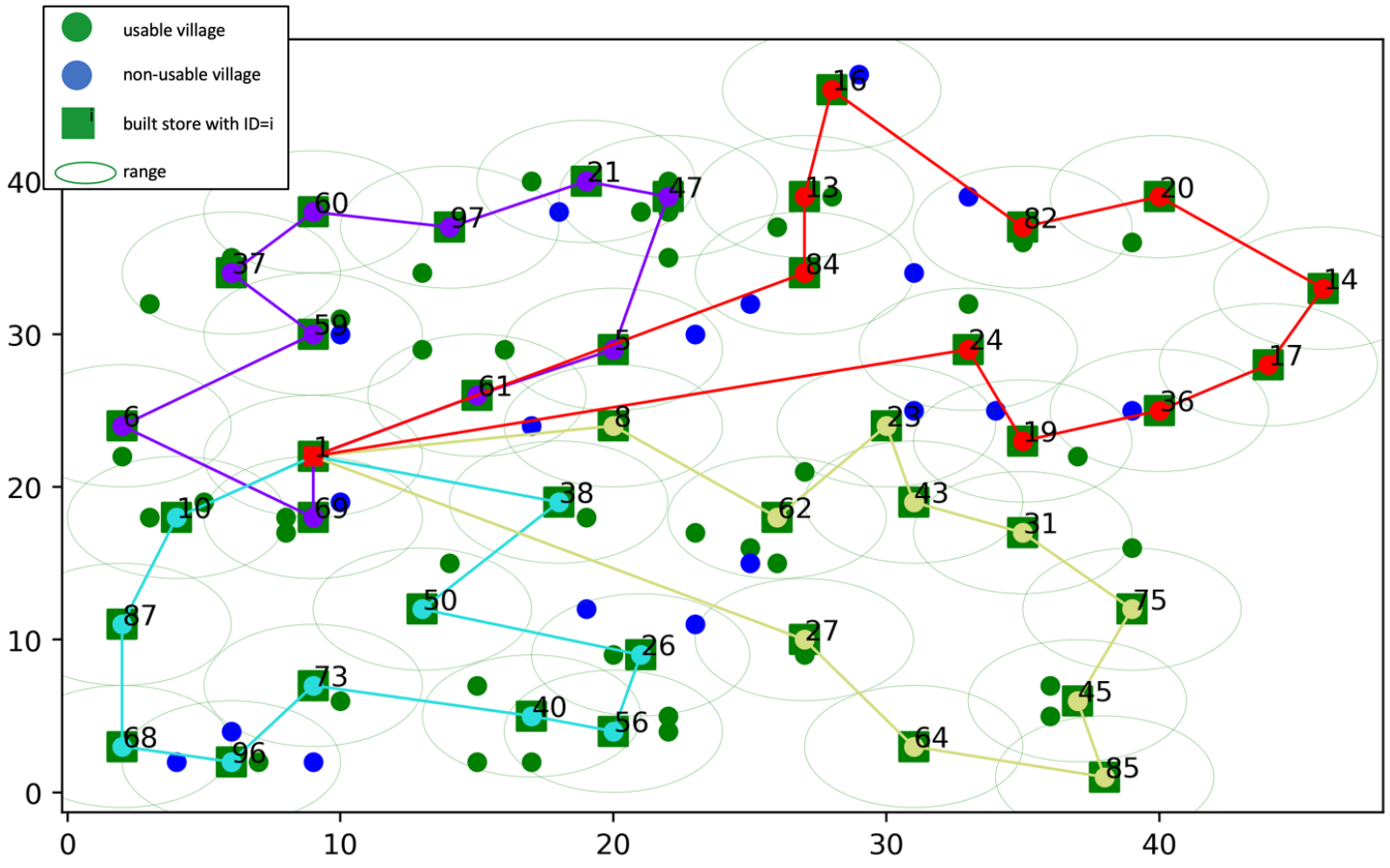

Figure 2: Solution of the minimart-I-100.dat file

Finally, we can also note that the total temporal complexity of the proposed VRP algorithm is $O(m^3)$ with m being the cardinality of *stores*; in fact, the core part is looping through all the possible savings, which are $m*m$, and for each saving, loop again through all the possible routes, which are at most m.