M-IRIFS:1 — 2020-2021

INFO-F424 - Combinatorial Optimization

Project: Facility Location Problem

Authors:
Nicolas WALLEMACQ
Maxime LANGLET

Professor:
Renaud CHICOISNE

Assistant:
Jérôme DE BOECK

3rd of May, 2021

Contents

Introduction

The overall goal of this year's INFO-F424's project is to is to implement Heuristic methods based on Linear Programming relaxations, greedy heuristics and local search moves for the Facility Location Problem.

FLP

The Facility Location Problem (FLP) is a classical problem in combinatorial optimization consisting of deciding where to open a set of facilities such that the demand of customers is satisfied at minimum cost. *Each customer $i \in I$ has a demand $d_i \in Z_+$. A set of locations J is available to build facilities: each location $j \in J$ has an opening cost f_j and can satisfy at most $u_j \in Z_+$ units of customers demand. Sending one unit of demand of a customer i to the location j has a travel cost t_{ij}. The FLP consists in determining where the facilities are built and which facilities supply the demand of which clients without exceeding the supply limit of each built facility, and minimizing the sum of opening and travel costs. .*
Firstly, it was asked to implement a Brute Force algorithm using **glpk** and **pyomo** on instances at most middle sized. The LP relaxation removes the integrality constraints of each variables. The resulting relaxation is thus a linear program, confirmed by the results of figure 3. This technique transforms an NP-hard problem into a related problem solvable in polynomial time. The solution to the LP relaxation can be used to gain information about the solution to the original integer problem. Then, it was asked to find an initial solution built with the help of the solution obtained via a LP relaxation, supplemented with a rounding procedure to generate a feasible solution. Lastly, the last solution might be improved via local search mechanisms. In fact, two movements were implemented, these are discussed in Section 3.

1 Brute Force Solution

The FLP can be casted as an Integer Programming problem (IP), described next :

$$\begin{aligned}
&\min_{x,y} f^T y + t^T x, && (1)\\
&s.t. \sum_{i \in I} x_{ij} \leq u_j y_j, \forall j \in J && (2)\\
&\sum_{j \in J} x_{ij} \geq d_i, \forall i \in I && (3)\\
&x \in \mathbb{Z}_+^{IJ}, && (4)\\
&y \in \{0,1\}^J && (5)\\
& && (6)
\end{aligned}$$

where x and y are defined as follows:

$$y_j := \begin{cases} 1 & \text{if facility } j \text{ is built} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{ij} := \text{Integer amount of demand of client } i \text{ that is satisfied by facility } j.$$

Note that equation (1) can be re-written, helping us with the implementation of the objective function. Here is the updated formula :

$$\min_{x,y} \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} x_{ij} t_{ij}$$

The implementation using **pyomo** and **glpk** is pretty straight forward. After defining the model, the range sets, each parameters and our variables, we can add our constraints and the objective function to the model. Note that a boolean parameter *linear* defines whether we will solve the integer model or the LP relaxation. This will change the domain of our variables, for x going from Non Negative Integers to Non Negative Reals, and for y from Binary to Non Negative Reals but bounded to between 0 and 1.

1.1 Results

For the results presented below, we limited ourselves to instances that are solved below 10 minutes. Those instances are:
FLP-100-20-0.txt, FLP-100-20-1.txt, FLP-100-20-2.txt, FLP-100-30-0.txt, FLP-100-30-1.txt, FLP-100-30-2.txt, FLP-100-40-0.txt, FLP-100-40-2.txt, FLP-150-30-0.txt, FLP-150-30-1.txt, FLP-150-30-2.txt, FLP-150-45-0.txt, FLP-150-45-2.txt, FLP-200-40-0.txt, FLP-200-40-1.txt, FLP-200-40-2.txt, FLP-250-50-0.txt, FLP-250-50-1.txt, FLP-250-50-2.txt.

Since some instances were excluded, some results might appear strange at first glance but those will be discussed.

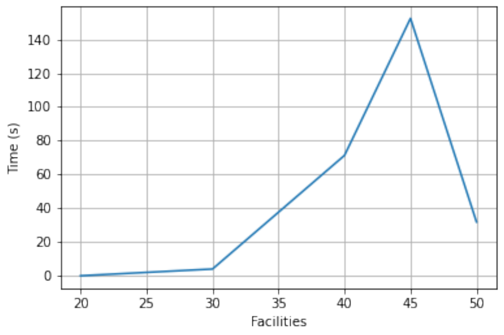


Figure 1: Average solution time as a function of the number of facilities J

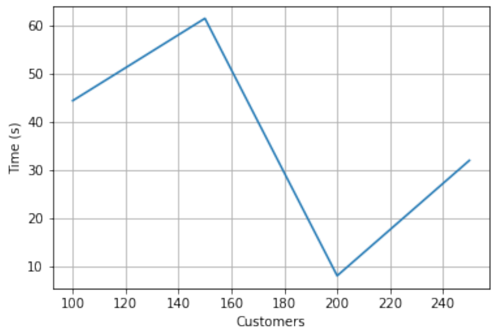


Figure 2: Average solution time as a function of the number of the customers I

Note that on Figure 1, the average time per number of facilities is taken here, meaning that different number of customers per instances are averaged together. Note also for 50 facilities, only the instances with 200 customers were averaged (because of the 10min constraint). This would explain the drop of values from 45 facilities to 50.

On Figure 2, we can observe another case of the effect of the fact that not many instances were present. In particular, for the instances with 200 customers, only the ones with 40 facilities respected this criterion. Similarly, the instances with 250 clients and more than 50 facilities were not computed. This would explain the drop observed.

Solving instances straightforwardly is not a scalable approach because we expect the problem to be solvable in exponential time when the dimensionality of the problem increases.As can be seen in Figure 1 we have a somewhat exponential curve showing, this of course when neglecting the last value. Given our results, increasing the number of facilities might have more weight in the time increase than an increasing number of customers.

Here below is given the average time taken to find a solution with the LP relaxation as function of the number of facilities J or as a function of the number of customers I.

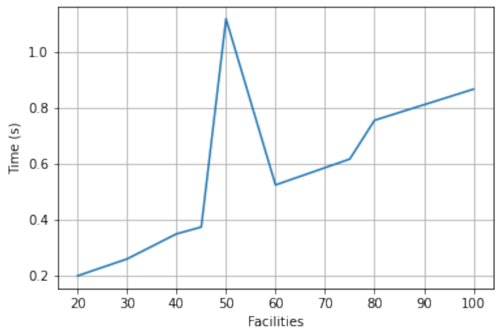


Figure 3: Average solution time of the LP relaxation resolution as a function of the number of the facilities J

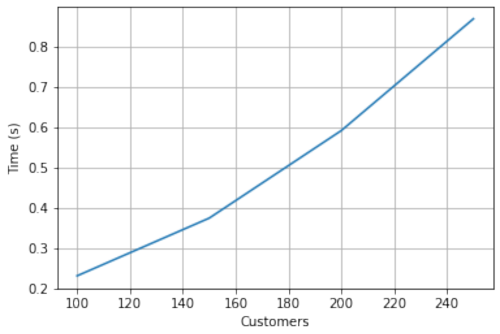


Figure 4: Average solution time of the LP relaxation resolution as a function of the number of the customers I

Both figures above show a linear time complexity rather than an exponential one that we estimated from before. This confirms the assessment from the introduction. We can also observe on both Figures 3 and 4 that both the the facilities and the customers influence the computing time.

We can now compute the Integrality Gap (IG) from the LP relaxation solution and the best solution. The IG is given by the following equation [1]:

$$IG = \frac{M_{int}}{M_{frac}}$$

where M_{int} is the minimum of the integer problem and M_{frac} the minimum of the linear programming relaxation.

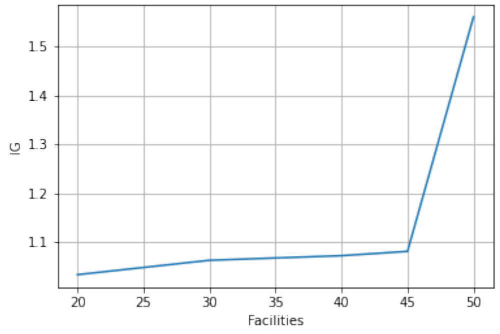


Figure 5: IG between the best solution and the solution found with the LP relaxation per number of facilities

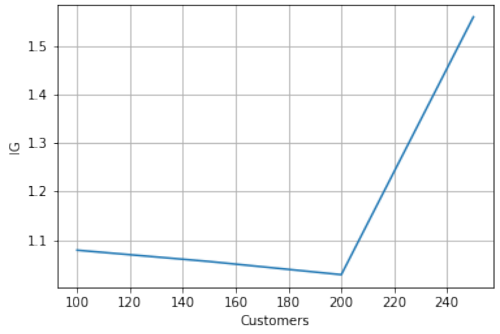


Figure 6: IG between the best solution and the solution found with the LP relaxation per number of customers

In both cases, we see at one point that the IG increases at a number of facilities and customers respectively. So, down the road, when computing a solution via a local search, we can expect a bigger IG for big instances than for small to medium sized instances.

2 A Greedy Algorithm

```
Algorithm 1: Greedy Rounding
  Data: An FLP instance
  Result: An integer feasible solution $(\bar{x}, \bar{y})$ for FLP
  Initialize $(\bar{x}, \bar{y}) \leftarrow (0, 0)$;
  Solve the LP relaxation of FLP. Let $(x^*, y^*)$ be an optimal solution;
  Sort $(y_j^*)_{j \in J}$ in decreasing order: $1 \geq y^*_{j(1)} \geq y^*_{j(2)} \geq ... \geq y^*_{j(J)} \geq 0$;
  for $j^{'} = 1...J$ do
      $j \leftarrow j(j^{'})$;
      $\bar{y}_j \leftarrow 1$;
      Sort $(x^*_{ij})_{i \in I})$ in decreasing order: $x^*_{i(1)j} \geq x^*_{i(2)j} \geq ... \geq x^*_{i(I)j} \geq 0$;
      for $i^{'} = 1...I$ do
          $i \leftarrow i(i^{'})$;
          if $\sum_{k \in I} \bar{x}_{kj} < u_j$ and $\sum_{l \in J} \bar{x}_{il} < d_i$ then
              $\bar{x}_{ij} \leftarrow \min\{u_j - \sum_{k \in I} \bar{x}_{kj}, d_i - \sum_{l \in J} \bar{x}_{il}\}$;
          end
      end
      if $\sum_{l \in J} \bar{x}_{il} \geq d_i$ for each $i \in I$ then
          return $(\bar{x}, \bar{y})$;
      end
  end
```

The Greedy Algorithm works as follows. It takes as input a FLP instance.
It first initializes the 2 vectors $\bar{x}$ and $\bar{y}$ with 0's. Those will be the output vectors of the algorithm. They will be fed with (integer) values throughout the algorithm.
In order to fill them with *well-chosen* values, we first solve the FLP with the LP relaxation and store the optimal values of the LP relaxation in the vectors x^* and y^*. Those optimal solutions x^* and y^* are not feasible solution of the FLP since they don't respect the constraints $x^* \in \mathbb{Z}_+^{IJ}$ and $y^* \in \{0,1\}^J$. Indeed, because of the LP relaxation, x^* and y^* can be fractional. Nevertheless, they offer a great heuristic. Thus we can sort y^* in decreasing order in order to recover their indexes j.
We iterate for $j' = 1...J$ and we take as j value the index of the j'th element of the sorted y^* list and set it to 1 (meaning we open this facility).
We do likewise with x^* (with j fixed). We iterate for $i' = 1...I$ and we take as i value the index of the i'th element of the sorted x^* list.

At that point we have 2 indexes (i,j) and we verify 2 conditions:

- the amount of demand of all the clients for a facility j should not exceed the capacity this facility
- the amount of demand satisfied by all the facilities for a client i should not exceed the of demand this client.

If indeed those conditions are met, we fill $\bar{x}_{ij}$ with the smallest of the 2 following values:

- the difference between the capacity of the facility j and the total demand of the clients satisfied by the facility j
- the difference between the demand of the client i and the total demand satisfied by the facilities for this client

After each round of j', we verify if the matrix $\bar{x}$ is such that all the demands of all the clients are satisfied with the facilities that we've opened until now. If it does, then we return the vectors $\bar{x}$ and $\bar{y}$ constructed so far. If it doesn't, then we continue with a new value for j'.

$\bar{x}$ and $\bar{y}$ are initialized with 0's. Since we only fill certain values of $\bar{y}$ with 1, it is obvious that $\bar{y}$ also contains only integers. $\bar{x}$ is constructed by adding only values which are a linear combination of integers. It results that $\bar{x}$ also only contains integers.
We have thus $\bar{x}$ and $\bar{y}$ composed of only integers. Computing the optimal function associated to $\bar{x}$ and $\bar{y}$ also leads to an integer since it is the sum and product of integers. We are assured that it is a feasible solution because we checked in the algorithm that the constraints over the capacity and the demand are met.

2.1 Results

Table 1: Results of the objective function of the initial solution for all the instances

Instances	Greedy Initial Solution
FLP-100-20-0.txt	78688.0
FLP-100-20-1.txt	82414.0
FLP-100-20-2.txt	61466.0
FLP-100-30-0.txt	98080.0
FLP-100-30-1.txt	94657.0
FLP-100-30-2.txt	106777.0
FLP-100-40-0.txt	102893.0
FLP-100-40-1.txt	104679.0
FLP-100-40-2.txt	102072.0
FLP-150-30-0.txt	162391.0
FLP-150-30-1.txt	190730.0
FLP-150-30-2.txt	171322.0
FLP-150-45-0.txt	249479.0
FLP-150-45-1.txt	192929.0
FLP-150-45-2.txt	158739.0
FLP-150-60-0.txt	216700.0
FLP-150-60-1.txt	182223.0
FLP-150-60-2.txt	207894.0

Instances	Greedy Initial Solution
FLP-200-40-0.txt	240875.0
FLP-200-40-1.txt	314549.0
FLP-200-40-2.txt	256592.0
FLP-200-60-0.txt	386715.0
FLP-200-60-1.txt	388829.0
FLP-200-60-2.txt	339996.0
FLP-200-80-0.txt	352508.0
FLP-200-80-1.txt	382577.0
FLP-200-80-2.txt	393787.0
FLP-250-50-0.txt	429803.0
FLP-250-50-1.txt	457342.0
FLP-250-50-2.txt	397449.0
FLP-250-75-0.txt	537410.0
FLP-250-75-1.txt	510417.0
FLP-250-75-2.txt	561531.0
FLP-250-100-0.txt	630709.0
FLP-250-100-1.txt	603129.0
FLP-250-100-2.txt	594299.0

Here is the integrality gap for the greedy solution obtained:

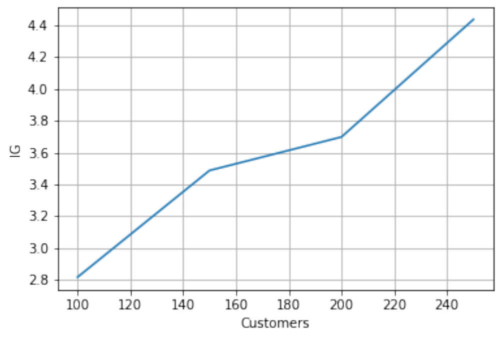


Figure 7: IG between initial solution and LP relaxation averaged over the number of customers

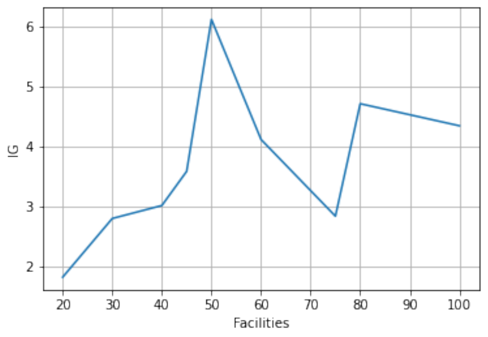


Figure 8: IG between initial solution and LP relaxation averaged over the number of facilities

We can see that these graphs present an integrality gap larger than the IG between the best solution and the LP relaxation. We can see that increasing the number of customers as a direct impact on the IG, whilst strictly increasing the number of facilities as a more disparate effect.

3 Improvement via Local Search

Algorithm 2: Greedy Reassign

```
Data: An FLP instance, an integer feasible solution $(x, y)$, facilities $(j_1^+, j_2^+, j_1^-, j_2^-)$
Result: An integer feasible solution $(\bar{x}, \bar{y})$ for FLP
Initialize $(\bar{x}, \bar{y}) \leftarrow (x, y)$;
Set $\bar{y}_j = 1$ for each $j \in \{j_1^+, j_2^+\}$;
Set $\bar{y}_j = 0$ and $\bar{x}_{ij} = 0$ for each $j \in \{j_1^-, j_2^-\}$ and each $i \in I$;
Sort the demands in decreasing order: $d_{i(1)} \geq d_{i(2)} \geq ... \geq d_{i(I)}$;
for $i^{'} = 1...I$ do
    $i \leftarrow i(i^{'})$;
    Sort the facilities by increasing travel cost to $i$: $t_{ij(1)} \leq t_{ij(2)} \leq ... \leq t_{ij(J)}$;
    for $j^{'} = 1...I$ do
        $j \leftarrow j(j^{'})$;
        if $\bar{y}_i = 1$ and $\sum_{k \in I} \bar{x}_{kj} < u_j$ and $\sum_{l \in J} \bar{x}_{il} < d_i$ then
            $\bar{x}_{ij} \leftarrow \min\{u_j - \sum_{k \in I} \bar{x}_{kj}, d_i - \sum_{l \in J} \bar{x}_{il}\}$;
        end
    end
end
return $(\bar{x}, \bar{y})$;
```

We implemented a local search using an iterative improvement algorithm based on the Greedy Reassign algorithm.

The two movements, **assignement movement** and **facility movement** are not used at the same time. We iterate those movements on the best solution found until that point. To that end, before accepting a new best solution, we need to verify that our initial constraints are indeed respected, let's discuss this more. We begin with one movement, let's say **assignement movement**, we try to improve our solution (if we do, we keep it in memory and becomes the best solution) but if after k iterations of the Greedy Reassign algorithm we didn't find a better solution than the best one that we already had, then we switch to the other local move (**facility movement** in this case). Again, we improve the solution and if again we have k iterations of the algorithm without improvement, we switch to the other local move. We repeat this again and again until a maximum number of iterations (3000 times) or the time constraint is exceeded. The key idea is to escape the local minimas we could encounter in our quest for an optimal solution. It is worth noting that at each iteration of our local search, we use a different random seed to select our new values of j and i.

Since our implementation of the movements iterates over the best solution found so far, it is possible to find a configuration that doesn't respect the constraints of our problem but yielding a better result. The facility movement closes randomly up to 2 facilities and opens up to 2 facilities. Since we have this iteration, down the road, it can happen that more than 2 facilities are closed compared to the initial solution found with the greedy rounding. Then it follows that, starting from any starting solution $(\bar{x}, \bar{y})$, it is **in theory** possible to always find a solution with those 2 moves if we apply them attractively on the best solution found so far but we need to discuss this claim. First, thanks to the facility movement which opens & closes facilities, it can converge towards the ideal number and attribution of facilities opened (and closed). Then, thanks to the assignment problem, it can converge towards the ideal distribution of the demand of the clients between the open facilities. However, due to the randomness of the moves, finding the optimal solution is only feasible in an arbitrarily long time although we have here a time constraint of 30 minutes.

In practice, we want to reduce the search space. So, in order to speed up the search, we added a new constraint: we check that the *y_new* proposed by the facility movement doesn't close more than 10% of the number of initially opened facilities from the initial solution. So with this new constraint, we are **not** guaranteed to find an optimal solution since we restricted the search to solutions with at a minimum of 90% of their initial number of facilities opened. Let's say we start with an initial solution of 100 facilities, our algorithm won't search through the space of solutions where the number of opened facilities is less than 90, although we could imagine that the optimal solution only has 88 facilities opened.

However, if we consider that the facility movement can't close more than a maximum of two facilities from the initial solution, down the road, then it follows that we are not guaranteed to find an optimal solution for our problem because we restrict our search to solutions where the number of facilities opened stays close to the number of facilities opened initially proposed by the greedy initial solution, even though it might be needed to close more facilities to reach the optimal solution.

3.1 Flowchart of the local search

The flowchart of our local search can be found in Appendix A. For the sake of clarity, it is a simplified view of the code and doesn't take into account small details such as

- where the counter variables which are used to change the local moves are initialized, incremented or set to 0.
- where the seed used for the random attributions is initialized, incremented or set to 0

Note that, as you can see on the flowchart, we added a heuristic to our algorithm. Indeed, with the facility movement, in order to limit the scope of our search to avoid encountering predominantly infeasible solutions, we added the constraint that we cannot close more than 10% of the number of available facilities compared from the number of initially opened facilities by the greedy rounding initial solution. So for instance, with an instance containing 50 candidate facilities, if the the sum of the facilities that are open in the initial solution y is 32, we won't search for new solutions with a total number of open facilities lower than $32 - \frac{50}{10} = 27$.

3.2 Results

The Figure 9 shows the evolution of the objective function over time for the local search on the biggest instance. Note that the plot on other instances showed the same trend as Figure 9. So it was not deemed necessary to show more. We can see that we rapidly decrease then gradually decreasing ever so slightly improving the quality of our solution. This slow gradual improvement motivates furthermore our decision to apply the algorithm on the best solution attractively.

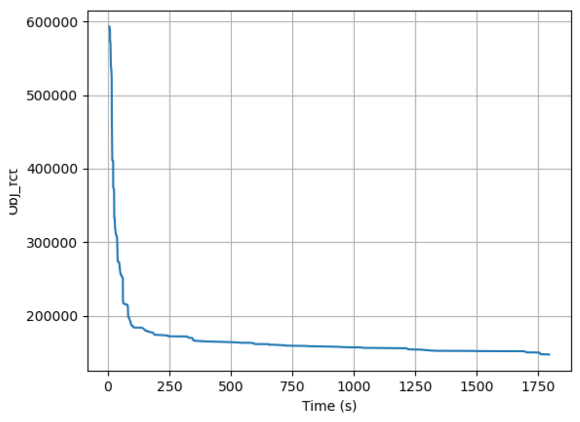


Figure 9: Evolution of the objective function over time for the local search on the biggest instance FLP-250-100-2.txt

In addition, although not asked in the requirements, we evaluated the quality of the solutions of our local search. We compared the best solution known (so only for the instances whose solutions were found under 10min in Section 1) and the ones given by our algorithm. To that end, the metric used is the Relative percentage deviation (RPD). It is given by the following formula:

$$RPD = 100 * \frac{(sol - best\_sol)}{best\_sol}$$

.

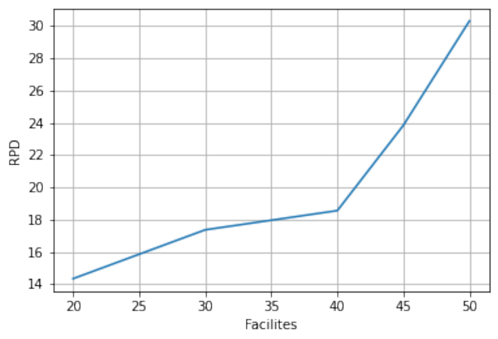


Figure 10: RPD function of the number of facilities

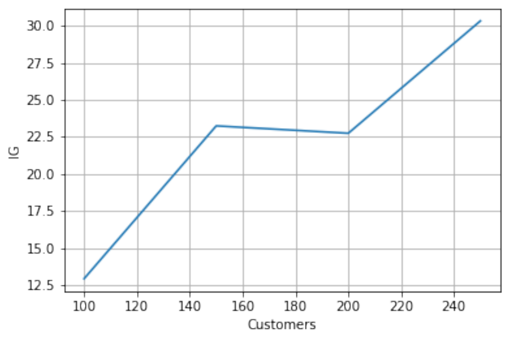


Figure 11: RPD function of the number of customers

We can see that the bigger the instance, the bigger the RPD will be. This is expected since, for large instances, the solution space will be less and less covered by our random local moves of our local search algorithm. The table of the solutions found by our local search algorithm for various instances can be found in Appendix B.

4 Conclusion

This project allowed us to familiarize ourselves with how to solve NP-hard problems, taking the classical Facility Location Problem as an example.

In the first implementation, we learned how to solve it with brute force using the Pyomo environment. We realized that this is not a scalable approach as the time taken to solve it is too long for instances with many facilities and clients.

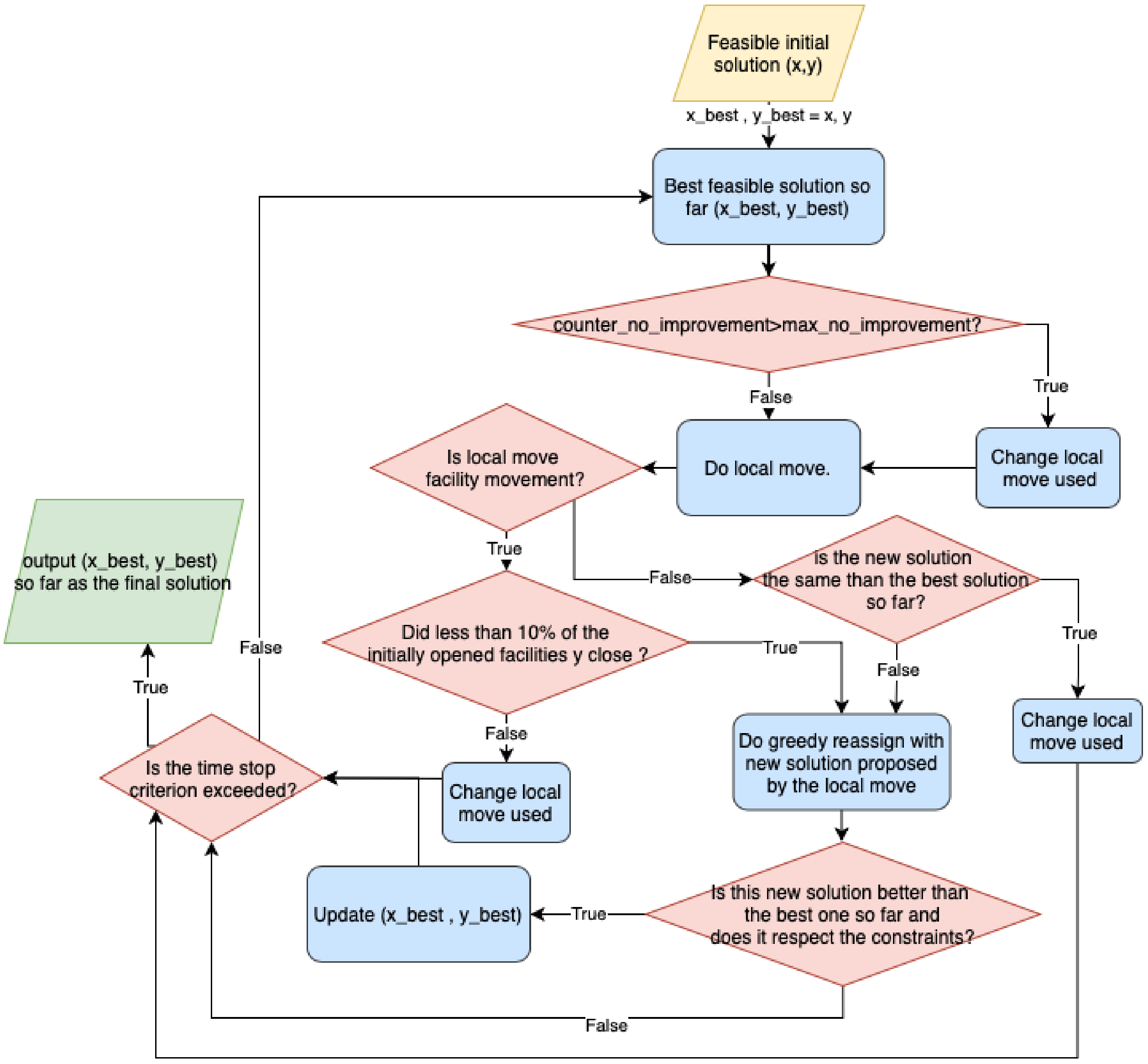
Then, in the second implementation, we aimed to find an relatively good initial solution thanks to a a LP relaxation and a greedy rounding to turn the LP solution to a feasible one, i.e. respecting the constraints of the problem.

Finally, this (feasible) initial solution was used as a starting point for a local search algorithm using 2 different moves: assignment movement and facility movement. Iteratively, the algorithm tries to modify the solution using one of those 2 local moves and applies a greedy reassignment. If the new solution is feasible and better than the best one so far, this solution is kept and a new local search is performed on this new solution. Otherwise, we start again our search starting from the best solution so far. When our search seems stuck in a local minima, spotted thanks to a counter indicating the number of iterations of the local search without any improvement, we changed the move to perform before the greedy reassignment in order to escape from the local minima and continue our search. We added a time constraint of 30 minutes after which the local search finally stops and outputs the results. This gave us decent results (see Figure 10 and 10) and we've realized that the solution quality obtained decreases when the size of the problem increases. Here are some ideas to further improve our local search: we could try to tune even better the different counters that we implemented in our search or more radically, we could add new local moves, change the heuristic to find our initial solution, add new heuristics, ... Also, increasing the time of the search would let the algorithm search through a bigger space of solutions.

References

[1] *Linear programming relaxation.* URL: `https://en.wikipedia.org/wiki/Linear_programming_relaxation`.

A Flowchart of the local search algorithm


Feasible initial solution (x,y)
x_best , y_best = x, y
Best feasible solution so far (x_best, y_best)
counter_no_improvement>max_no_improvement?
True
False
Change local move used
Do local move.
Is local move facility movement?
True
False
Is the new solution the same than the best solution so far?
True
False
Change local move used
Did less than 10% of the initially opened facilities y close ?
True
False
Do greedy reassign with new solution proposed by the local move
Change local move used
Is the time stop criterion exceeded?
True
False
output (x_best, y_best) so far as the final solution
Is this new solution better than the best one so far and does it respect the constraints?
True
Update (x_best , y_best)
False

B Table of results for the local search

Table 2: Results of the objective function of the local search for some instances

Instances	Local search solution
FLP-100-20-0.txt	41414
FLP-100-20-1.txt	52843
FLP-100-20-2.txt	49751
FLP-100-30-0.txt	41206
FLP-100-30-1.txt	38200
FLP-100-30-2.txt	44045
FLP-100-40-0.txt	38479
FLP-100-40-1.txt	/
FLP-100-40-2.txt	33089
FLP-150-30-0.txt	83049
FLP-150-30-1.txt	90072
FLP-150-30-2.txt	79879
FLP-150-45-0.txt	72767
FLP-150-45-1.txt	/
FLP-150-45-2.txt	74596
FLP-150-60-0.txt	/
FLP-150-60-1.txt	/
FLP-150-60-2.txt	/

Instances	Local search solution
FLP-200-40-0.txt	165615
FLP-200-40-1.txt	135369
FLP-200-40-2.txt	142111
FLP-200-60-0.txt	/
FLP-200-60-1.txt	/
FLP-200-60-2.txt	/
FLP-200-80-0.txt	/
FLP-200-80-1.txt	/
FLP-200-80-2.txt	/
FLP-250-50-0.txt	200004
FLP-250-50-1.txt	196556
FLP-250-50-2.txt	211025
FLP-250-75-0.txt	/
FLP-250-75-1.txt	/
FLP-250-75-2.txt	/
FLP-250-100-0.txt	/
FLP-250-100-1.txt	/
FLP-250-100-2.txt	/