Combinatorial Optimization - INFO-F-424 Project '21 - Facility Location Problem [*]

Renaud Chicoisne renaud.chicoisne@ulb.ac.be

Jérôme de Boeck jdeboeck@ulb.ac.be

Problem description

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 $i \in \mathcal{I}$ has a demand $d_i \in \mathbb{Z}_+$. A set of locations $\mathcal{J}$ is available to build facilities: each location $j \in \mathcal{J}$ has an opening cost f_j and can satisfy at most $u_j \in \mathbb{Z}_+$ 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.

Even though FLP can be easily modeled as an IP, the high number of integer variables makes it challenging to solve past some magnitude of $I = |\mathcal{I}|$ and $J = |\mathcal{J}|$. In this project, we propose heuristic methods based on Linear Programming relaxations (LP), greedy heuristics and local search moves.

1 Brute force solution

Let us define the variables x and y as follows:

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

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

FLP can be cast as the following Integer Programming problem (IP)

$$\min_{x,y} \quad f^\top y + t^\top x \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}} x_{ij} \leqslant u_j y_j, \qquad \forall j \in \mathcal{J} \tag{2}$$

$$\sum_{j \in \mathcal{J}} x_{ij} \geqslant d_i, \qquad \forall i \in \mathcal{I} \tag{3}$$

$$x \in \mathbb{Z}_+^{IJ} \tag{4}$$

$$y \in \{0,1\}^J \tag{5}$$

1. Solve the small and medium sized instances directly with glpk in a python script `flp.py` using pyomo. Your script must contain a function `solve_flp(instance_name,linear) : return (obj,x,y)` where `instance_name` is the name of an instance in the same folder (including the extension) and the return value is a tuple tuple containing the optimal value `obj` and an optimal solution `x` that is a list of lists where `x[i][j]` $= x_{ij}$ and `y` is a list where `y[j]` $= y_j$. The parameter `linear` is a boolean that indicates if the integer model must be solved (`False`) or the LP relaxation (`True`).

[*]This `pdf` is available on the UV and the course's Teams channel.

2. Graph the average solution time as a function of the number of facilities J and as a function of the number of clients I. Is solving it straightforwardly a scalable approach? Provide the results of the LP relaxation as well in terms of solving time and integrality gap. For simplicity, you are allowed to study only instances that are solved in 10 minutes or less.

2 A Greedy Algorithm

We now present a rounding procedure that generates a feasible integer solution for FLP from the - potentially fractional - optimal solution (x^*, y^*) of its LP relaxation (i.e. without constraints (4) and (5)).

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 $(y^*_j)_{j \in \mathcal{J}}$ in decreasing order: $1 \geqslant y^*_{j(1)} \geqslant y^*_{j(2)} \geqslant ... \geqslant y^*_{j(J)} \geqslant 0$;
for $j' = 1...J$ do
    $j \leftarrow j(j')$;
    $\bar{y}_j \leftarrow 1$;
    Sort $(x^*_{ij})_{i \in \mathcal{I}}$ in decreasing order: $1 \geqslant x^*_{i(1)j} \geqslant x^*_{i(2)j} \geqslant ... \geqslant x^*_{i(I)j} \geqslant 0$;
    for $i' = 1...I$ do
        $i \leftarrow i(i')$;
        if $\sum_{k \in \mathcal{I}} \bar{x}_{kj} < u_j$ and $\sum_{l \in \mathcal{J}} \bar{x}_{il} < d_i$ then
            $\bar{x}_{ij} \leftarrow \min\{u_j - \sum_{k \in \mathcal{I}} \bar{x}_{kj}, d_i - \sum_{l \in \mathcal{J}} \bar{x}_{il}\}$;
    if $\sum_{j \in \mathcal{J}} \bar{x}_{ij} \geqslant d_i$ for each $i \in \mathcal{I}$ then
        return $(\bar{x}, \bar{y})$;
```

1. Interpret in your own words each step of Algorithm 1 and explain why it always returns an integer feasible solution for FLP.
2. Implement a python function `initial_solution_flp(instance_name) : return (obj,x,y)` - where `instance_name` is the name of an instance in the same folder (including the extension) - that returns an initial solution of the FLP as described in Algorithm 1. The format of the returned solution is the same than for `solve_flp`.
3. Give an optimality gap for the greedy solution obtained.

3 Improvement via Local Search

The solution $(\bar{x}, \bar{y})$ returned by Algorithm 1 is feasible for FLP but can be improved through a Local Search mechanism. The two following types of movements can be used:

Assignment movements : Randomly select a maximum of 2 customers i_1 and i_2 and up to 2 facilities each (j_1^1 and j_1^2 for i_1 and j_2^1 and j_2^2 for i_2) such that $\bar{x}_{i_1 j_1^1}, \bar{x}_{i_1 j_1^2}, \bar{x}_{i_2 j_2^1}, \bar{x}_{i_2 j_2^2} > 0$. Reassign randomly each demand to up to 2 facilities each.

Facility movement : Randomly select a maximum of 2 facilities j_1^- and j_2^- that can be closed and up to 2 facilities j_1^+ and j_2^+ that can be opened. The total demand must still be met so consider only the tuples $(j_1^+, j_2^+, j_1^-, j_2^-)$ such that $\sum_{i \in \mathcal{I}} (\bar{x}_{ij_1^-} + \bar{x}_{ij_2^-}) \leqslant u_{j_1^+} + u_{j_2^+}$. The customers of the closed facilities are reassigned in a greedy way in increasing order of t_{ij} with Algorithm 2, which must be adapted if less than two facilities are opened or closed.

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 $i \in \mathcal{I}$;
Sort the demands in decreasing order: $d_{i(1)} \geqslant d_{i(2)} \geqslant ... \geqslant d_{i(I)}$;
for $i' = 1...I$ do
    $i \leftarrow i(i')$;
    Sort the facilities by increasing travel cost to $i$: $t_{ij(1)} \leqslant t_{ij(2)} \leqslant ... \leqslant t_{ij(J)}$;
    for $j' = 1...J$ do
        $j \leftarrow j(j')$;
        if $\bar{y}_j = 1$ and $\sum_{k \in \mathcal{I}} \bar{x}_{kj} < u_j$ and $\sum_{l \in \mathcal{J}} \bar{x}_{il} < d_i$ then
            $\bar{x}_{ij} \leftarrow \min\left\{u_j - \sum_{k \in \mathcal{I}} \bar{x}_{kj}, d_i - \sum_{l \in \mathcal{J}} \bar{x}_{il}\right\}$;
return $(\bar{x}, \bar{y})$;
```

1. Implement a local search mechanism `local_search_flp(x,y) : (obj,x,y)` using the two kinds of movement described above, starting from the initial solution $(\bar{x}, \bar{y})$ returned by `initial_solution_flp`. The tuning of these movements is left to you, e.g. the frequency at which you use each move, after how many unsuccessful moves, etc. The stopping criterion of your heuristic will be a time limit of 30 minutes.
2. For any starting solution $(\bar{x}, \bar{y})$, is it always possible to find an optimal solution with the moves described here? In the negative, show a counter example. Otherwise, briefly explain why.
3. Explain the execution of your local search through a flowchart and justify in detail the tuning of your movements. Graph the evolution of the best solution found from the starting solution during the execution of your local search on the biggest instances.

In every instance available, each customer can be assigned to any facility and the total capacity of all facilities is more than 150% the total demand. Moreover, there always exists a solution even if up to two facilities are closed. You can use these informations to justify the fine tuning of your Local Search moves.

4 Pyomo

All you code must be written in a python file `flp.py`. A draft is provided that contains a function `read_instance(file_name)`, which reads the data of an instance `file_name` placed in a folder `Intances` and returns it as dictionaries to be able to transfer it simply to Pyomo. Notice that a value t_{ij} is accessible via `travel_cost[(i,j)]`.

Details on how to install Pyomo are provided here, on Modelling features here and Examples here. It is recommended to work with a concrete model `model = pyo.ConcreteModel()`. To use the data returned by the `read_instance` function in Pyomo, first define sets for I and J with the following commands:

- `model.I = pyo.RangeSet(0,len(demand)-1)`
- `model.J = pyo.RangeSet(0,len(capacity)-1)`

then create the parameters:

- `model.f = pyo.Param(model.J,initialize=opening_cost,default=0)`
- `model.c = pyo.Param(model.J,initialize=capacity,default=0)`
- `model.d = pyo.Param(model.I,initialize=demand,default=0)`
- `model.t = pyo.Param(model.I,model.J,initialize=travel_cost,default=0)`

The solver `glpk` is to be used and can be installed as detailed here for Linux, here for Mac Os X and here for Windows. To solve a model, output the execution and print the value of an objective `model.obj` previously defined, add the following lines:

- `opt = pyo.SolverFactory('glpk')`
- `opt.solve(model,tee=True)`
- `print(pyo.value(model.obj))`

Questions over the functionalities can be asked on the UV. The use of classes is allowed but the three functions asked for in this project must remain functions outside of any class.

5 General instructions

The project must be made by groups of two. Please send a mail for your group containing the names and ids of the students to jdeboeck@ulb.ac.be by March 19th. If you do not find a partner, please contact me before the deadline. All students who have not taken contact by March 26th are considered as not doing the project.

The deadline for the project is May 3rd at 2 pm. For each group send a `zip` file containing your python script and the report in `pdf` format at jdeboeck@ulb.ac.be. The report should be no more than 7 pages long.