13-12-2020

AMMM PROJECT

CPLEX, Greedy, Local Search & GRASP

ANDREA DE LAS HERAS
ARNAU ABELLA
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Contents

Problem Statement and Linear Model

The problem of the Company could be formalized as following:

Given:

- The set L of possible locations. Each location l consists of a coordinate (l_x, l_y).
- The set C of cities. Each city c consists of a coordinate (c_x, c_y) and a population p_c.
- The set T of types. Each type t consist of a capacity cap_t, a working distance d_{city_t} and an installation cost $cost_t$
- The minimum distance between centres d_centre.

Find the locations to install the logistic centres, the type for each centre and the assignment of primary and secondary centre to each city, subject to the following constraints:

(1) Each city has assigned exactly one primary centre.
(2) Each city has assigned exactly one secondary centre.
(3) For each city, its primary centre must be different of its secondary centre.
(4) The distance between each pair of centres must be at least the minimum.
(5) The distance between a city and its primary centre cannot exceed the working distance of that centre's type.
(6) The distance between a city and its secondary centre cannot exceed three times the working distance of that centre's type.
(7) For each centre, the capacity of the centre's type cannot be exceeded by the sum of the populations of the cities it serves as a primary centre and the 10% of the populations of the cities it serves as a secondary centre.

With *objective* to minimize the total installation cost.

This implies that we need to define three variables, one for each assignment we want to do (Type Centres-Locations, Primaries centres and secondary centres). Then, using the usual metric $d\big((x_1, y_1), (x_2, y_2)\big) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, we can formulate the company's problem mathematically as following:

L	Set of possible locations, index l.
C	Ser of cities, index c.
T	Set of types, index t.
d_centre	Minimum distance between centres.
cap_t	capacity of the type t.
d_city_t	Working distance of the type t.
$cost_t$	Installation cost of the type t.
p_c	Population of the city c.
p_{cl}	binary. Equals to 1 if a centre in location l serve as primary centre the city c; 0 otherwise.
s_{cl}	binary. Equals to 1 if a centre in location l serve as secondary centre the city c; 0 otherwise.
x_{lt}	binary. Equals to 1 if a centre of type t is emplaced in the location l; 0 otherwise.

Minimize

$$\sum_{l \in L} \sum_{t \in T} x_{lt} \cdot cost_t$$

Subject to:

$$\sum_{l \in L} p_{cl} = 1 \quad \forall c \in C \tag{1}$$

$$\sum_{l \in L} s_{cl} = 1 \quad \forall c \in C \tag{2}$$

$$p_{cl} + s_{cl} \leq 1 \quad \forall c \in C, l \in L \tag{3}$$

$$d(l_1, l_2) \geq x_{l_1 t_1} \cdot x_{l_2 t_2} \cdot d\_centre$$
$$\forall l_1, l_2 \in L,\ t_1, t_2 \in T \quad s.t. \quad l_1 \leq l_2 \ \ and\ if\ l_1 = l_2\ then\ t_1 \neq t_2 \tag{4}$$

$$p_{cl} \cdot d(l, c) \leq \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L \tag{5}$$

$$s_{cl} \cdot d(l, c) \leq 3 \cdot \sum_{t \in T} x_{lt} \cdot d\_city_t \quad \forall c \in C, l \in L \tag{6}$$

$$\sum_{c \in C} p_{cl} \cdot p_c + 0.1 \cdot \sum_{c} s_{cl} \cdot p_c \leq \sum_{t \in T} x_{lt} \cdot cap_t \quad \forall\ l \in L \tag{7}$$

Note that the restrictions in the mathematical formulations follow the same order as the formal approach of the problem to make it easier to understand, even so, we want to give indications regarding some of them:

- An easy way to understand the model is the following: we have three vectors for location, one with the length of the number of types of centres and two with the length of the number of cities to represent the primary and secondary assignment, such that we can model the problem in matrix form.
- $d(l_1, l_2)$ and $d(l, c)$ denotes the usual distance between two locations and between a location and a city, respectively. We can always calculate this distance because they give us the coordinates and as they are constants values, they do not affect to the linearity of the inequalities.
- This is not a linear model as (4) are not linear inequalities, but we think that in this form it is easier to understand and it is also easy to transform into linear. The product in (4) follows the idea that we only want to check the distance between location with a centre, so:
    - If at least one of the two locations have not any centre of some type, then the product always will be zero and the equations always be satisfied.

- If the two locations have a centre of some type, then there exists an inequality for which the product is one and the inequality is verified just in case they serve the distance constraint.
- If there are two types of centres in the same location, then there exists an inequality for which the distance between the two centres is zero and fails if the minimum distance is nonzero. As we have not any indication about the distance or the number of centres per location, we cannot assume that the distance cannot be zero or that the number of centres must be one; that is why we model the problem in this manner and not in other way that generates less inequalities.

Heuristic Algorithms to Solve the Problem

Greedy Algorithm

A greedy algorithm is a simple, intuitive algorithm that makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. In this problem what we want is to minimise the cost, so, for every location our optimal choice is the type of centre with the minimum cost. But, as our problem also requires an assignation of primary and secondary centres, it is possible that just with this simple election of type we cannot make the assignation due to the capacity and distance restrictions. That is why we must consider the option of upgrading a centre type to the next one with the smallest cost (The smallest difference of cost). Then, our optimal choice for the assignation of primary and secondary centres is the one with the least increment of cost caused by the necessity of changing the type of centre to enable the assignation. See that if it is not necessary to change the type to make the assignation, the cost increment is zero.

In this way we can make a cost function that gives us the cost of every candidate such that at every step we can choose the smallest cost to the assignation of primary and secondary centres. Bellow we provide the pseudocode for the cost function and the constructive algorithm:

Algorithm CostFunction ($c, l, T, assignation\_type$)
Input: A city c, a location l, the set of types of centres and the type of assignation (Primary or Secondary)
Output: The cost increment of assigning the city to the location.
if the location l contains a facility, **then**:
    Check if the assignment is feasible with the current facility (range, capacity considering the type of assignation…)
    if feasible **then**:
        cost ← 0
    otherwise
        Try to upgrade the facility type
        if there is a feasible one in T, **then**:
            cost ← cost facility type - cost previous facility type
        otherwise
            return Infeasible
otherwise
    if you can place a facility in the location (d_center), then:
        Pick the facility with the smallest cost that is feasible (range, capacity…)
        cost ← cost facility type
    otherwise
        return Infeasible
return cost

Algorithm CONSTRUCTIVE($C, L, T, assignation\_type$)
Input: The set of cities C, the set of locations L, the set of types of centres T and the type of assignation (Primary or Secondary)
Output: the assignation of primary or secondary centre.
for each city c **do**:
    min_cost ← Infinity
    for each location l **do**:
        cost ← COSTFUNCTION ($c, l, T, assignation\_type$)
        if cost < min_cost **then:**
            min_cost ← cost
            optimal_choice ← l
    Assign to c the optimal_choice as primary/secondary centre
return all the assignments

Greedy Algorithm with local search

Once we have an initial solution given by the greedy algorithm, we can improve the solution using a local search. We can see that in the greedy algorithm the assignation of centres only depends on the previous assignation and it does not consider the posterior ones. So, it is possible that, in the last assignations, we changed the type of a centre in a way that with its new capacity it could be assigned to one of the first cities and downgrade its previous assignation reducing the cost. It is important to remark that this situation can be repeated after each update so, every time we have an improvement, we shall check if there is yet another possible improvement.

Algorithm LOCALSEARCH($assignation,\ mode$)
Input: The assignation given by the greedy algorithm and the criterion, $mode$, used to improve the solution. The possible criterions are FIRSTIMPROVEMENT (Use the first candidate that is found to make the improvement) or BESTIMPROVEMENT (Look for all the candidates to make an improvement and choose the best of them).
Output: An improvement in the assignation.
while there has been an improvement **do:**
    for each facility f used in the $assignation$ **do:**
        for each city c assigned to the facility f (as primary or secondary) **do:**
            if the facility can be downgraded (improving its cost) by removing c **then**:
                for the rest of facilities f' **do:**
                    if c can be assigned to f' **then:**
                        if $mode$ = FIRSTIMPROVEMENT **then:**
                            Update the solution with this reassignment and facility downgrade
                            break
                        if $mode$ = BESTIMPROVEMENT **then:**
                            candidates ← candidates ∪ $\{f'\}$
                If $mode$ = BESTIMPROVEMENT **then:**
                    Pick the candidate with smallest cost and upgrade the solution.

GRASP

The GRASP algorithm (*Greedy Randomized Adaptive Search Procedure*) consists of two phases that are reiterated. In the first one we generate a feasible solution to the problem using the costs criterion previously defined for the greedy algorithm (COSTFUNCTION). For the second phase we apply the LOCALSEARCH algorithm to explore the solution until finding a local minimum. According to this we develop the following algorithm for the first phase, in which we construct

the restricted candidate list using only the locations whose cost does not exceed a margin from the minimum:

Algorithm GREEDYRANDOMIZEDCONSTRUCTION($C, L, T, assignation\_type, \alpha$)
Input: The set of cities C, the set of locations L, the set of types of centres T, the type of assignation (Primary or Secondary) and the factor α which defines the margin applied in the construction of the RCL.
Output: A possible assignation
for each city c in C **do:**
  Build the restricted candidate list (RCL)
  for each location l in L **do :**
    CostList ← CostList ∪ { COSTFUNCTION $(c, l, T, assignation\_type)$}
  min ← min(CostList)
  max ← max(CostList)
  RCL ← $\{l \in L : min \leq cost \leq min + \alpha \cdot (max - min)\}$
  Assign to c a location chooses u.a.r. from RCL
return all the assignations

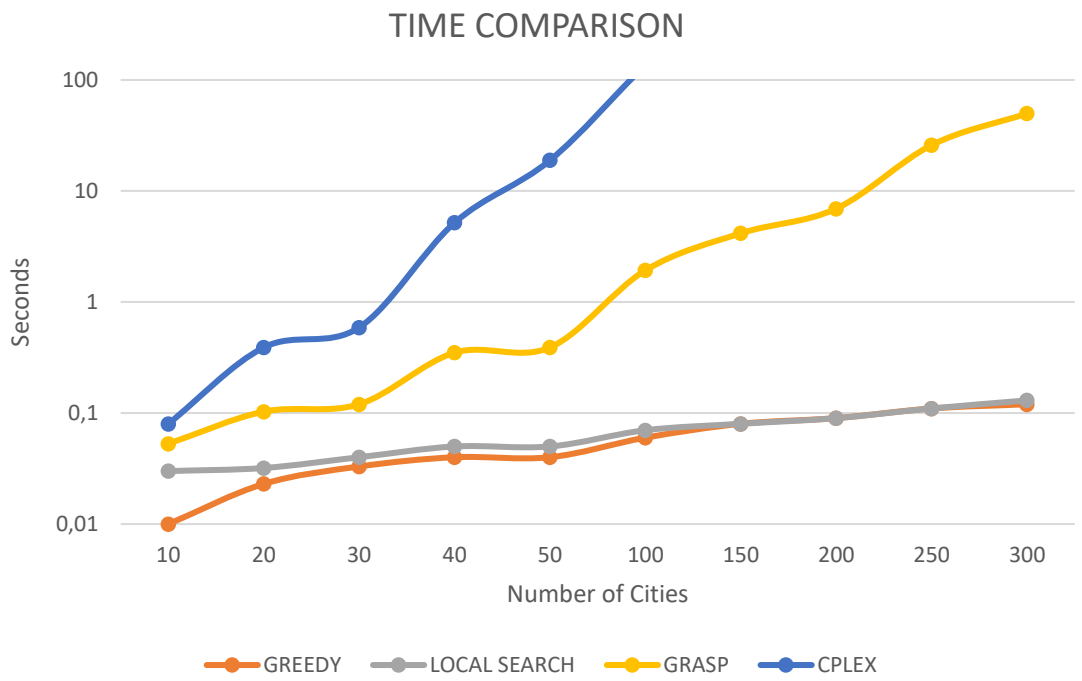
Finally, as the second phase is just to apply the LOCALSEARCH algorithm, we have that the GRASP would end up as follows.

Algorithm GRASP($C, L, T, assignation\_type,\ mode, \alpha, time\_limit, iterations\ \_limit$)
Input: The set of cities C, the set of locations L, the set of types of centres T, the type of assignation (Primary or Secondary), the criterion for the local search $mode$,the factor α which defines the margin applied in the construction of the RCL, the maximum limit of time for the algorithm $time\_limit$ and the maximum iterations without an improvement $iterations\ \_limit$.
Output: A solution to the problem assignation
while it is not exceeded the $time\_limit$ or the $iterations\ \_limit$ **do:**
  $assignation$ ← GREEDYRANDOMIZEDCONSTRUCTION($C, L, T, assignation\_type, \alpha$)
  $assignation$ ← LOCALSEARCH($assignation,\ mode$)
return $assignation$

Comparative Results and Conclusions

In order to make a comparative analysis of the algorithms, we computed a solution for the same instance with the different algorithms in such a way that, for different sizes of the instance, we can compare the time and the cost obtained by all of them. In the instances that we used, the number of locations is one third of the number of cities, for this reason in our graphics we only use the number of cities to give the idea of the size and the difficulty of the instance. To make the graphics more easily readable we decided to use a logarithmic scaling for the y-axis.

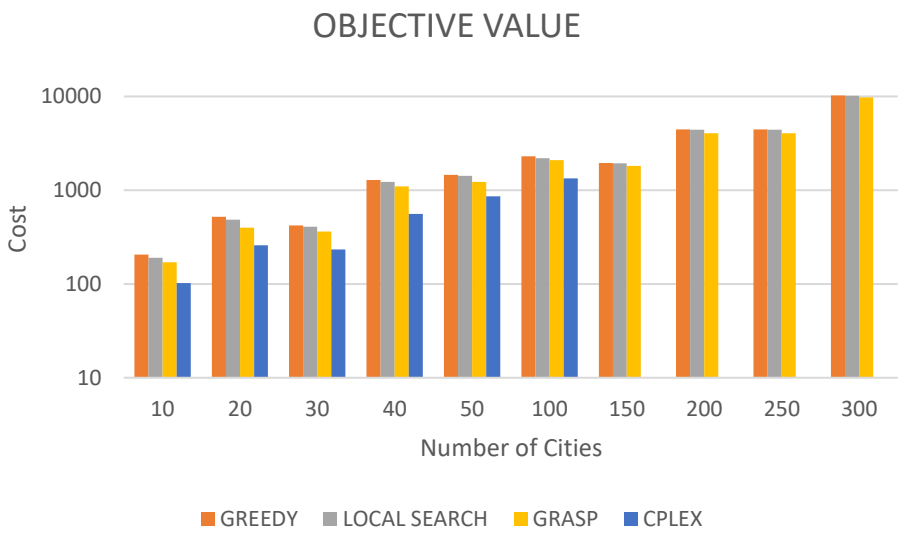
We are going to start comparing the time each algorithm needs for the same instance since the time requirement is our primary concern. The following chart presents an average of execution times for each size of the input.



As we already know this is an NP-Hard problem so, when we try to calculate the optimal solution using the CPLEX algorithm, the time grows so quickly in such a way that we cannot obtain the solution in a reasonable time and, for this reason, we usually opted to use the heuristic algorithms. That result is perfectly reflected in the time graph, for more than a hundred of cities the CPLEX algorithm requires more than two minutes while the GRASP algorithm just two seconds and the Greedy Algorithm and the Local search less than a second. It is clear that the GRASP algorithm grows a lot faster than both local search and greedy algorithms but the comparison between these two is more interesting. It can be observed how they behave in a similar way in the interval between 150 and 250 cities but with a bigger input the local search algorithm is a little bit slower again.

Another measure we have examined is the variability of the time to solve the same problem by the three heuristic algorithms. We have that the mean of the coefficients of variation in percentage for the greedy, local search and GRASP algorithms are 30%, 24% and 33% respectively. Then, we can conclude that the local search algorithm is the most stable with respect to the execution time.

In addition to the times analysis we have done, it is mandatory to also study the quality of the outputs of each algorithm. This is because if the improvement in the results is not proportional to the extra time invested, we should consider if the less precise algorithm fits better our needs.



As we expected, the best solution is given by the CPLEX algorithm but, this problem is so hard that, for more than 100 cities, we were not able to obtain the solution in a good time. If we compare the heuristics algorithm taking into account the previous time table, we can see that the relation between the time and the improvement of the local search respect to the greedy algorithm is proportionally the same at every size. Then, we can say that the local search is a good improvement to our problem and in general we will prefer to use it. However, the relation time – improvement for the GRASP is not as good because for larger values in the number of cities it is proportionally minor every time. In spite of this, we have that the GRASP algorithm has more parameters, including a time restriction, and it is possible that with certain configuration we can obtain a better result for our needs.

It is important to remark too that the time required by the algorithms not only depends on the size of the given set, but it also depends on how bad the instance of the problem is (see that there exist instances without solution, for example, one in which a city population exceeds the capacity of every centre). The problem is that as the coordinates lie in the Euclidean space $\mathbb{R}^2$, there exists a lot of bad instances for the problem that have a high probability to appear when we generate them at random and these make it so difficult to create a good instance generator. So, although we try to reduce the probability to obtain a bad instance in our implementation, it is still high, and we continue obtaining bad results. Referring to this problem about the instances and the computing time, we also have that the computational capacity of our computers is so limited and therefore we cannot compute too large instances of the problem. Finally, to work with that limitations we opted for make instances that do not have a big number of cities but that are ass well difficult of solve because the capacity of the centres type.

We want also to give the idea that this project could be improvement for future works. We can refine the implemented algorithms and add new strategies to the local search, for example: swapping two cities may not decrease the cost by itself, but the final solution can be improved by applying it to the local search. Another idea is to improve the computing time parallelizing the problem by dividing the search area into smaller ones. We can apply any of our algorithms to those subregions with an overlapping area between them so that we can combine the partial solutions by checking the output in those overlaps in order to get the final output. And of course, one can add new different algorithms that also solve this problem.

About the Project

Linear Programming

The project contains an IBM ILOG CPLEX program written in OPL that solves the stated problem.

- The OPL program
- The runner script.

Instructions

You only need import the whole project in OPLIDE and run the script.

The input instances are fixed in the script. Open the script with your favourite text editor and change the route.

Heuristic Algorithms

The project includes:

- A collection of heuristics algorithms to solve the statement problem.
- An instance generator for both the CPLEX and the heuristic approach.

Structure

- `app` contains the sources of the executable.
- `src` contains the sources of the library that the executable depends on.
- `test` contains the sources of the library's test.
- `instances` contain some instances to test the solver.

Installation

This project is build using GHC(compiler) and cabal(build tool).
The easiest way to install both is using ghcup

```
# Install ghcup
curl --proto '=https' --tlsv1.2 -sSf https://get-
ghcup.haskell.org | sh

# Install GHC using ghcup
ghcup install ghc 8.8.4

# Install cabal using ghcup
ghcup install cabal
```

Finally, we need to compile the project. This may take some minutes and requires internet connection. This project does not depend on any .so so it should be possible to compile it in any architecture that supports ghc.

```
# It may take some minutes
$ cabal build
...
Preprocessing executable 'heuristics' for heuristics-0.1.0.0..
Building executable 'heuristics' for heuristics-0.1.0.0..
HOME/heuristics/dist-newstyle/build/x86_64-linux/ghc-
8.8.4/heuristics-
0.1.0.0/x/heuristics/opt/build/heuristics/heuristics ...
```

How to

Let's start by showing the available options:

```
# Use the --help flag to display the options
$ cabal run heuristics -- --help

Usage: heuristics COMMAND

Available options:
  -h,--help                Show this help text

Available commands:
  generator                Problem instance generator
  solver                   Heuristic solver
```

From this point, you should be able to run the different algorithms. See examples below:

```
# Greedy (the most basic one)
cabal run heuristics -- solver greedy -f
'instances/sample_10.dat'

# Greedy with local search first improvement
cabal run heuristics -- solver greedy --localSearch -f
'instances/sample_10.dat'

# Greedy with local search best improvement
cabal run heuristics -- solver greedy --localSearch -b -f
'instances/sample_10.dat'

# GRASP with time limit and alpha threshold.
cabal run heuristics -- solver grasp --limit 300 --threshold 0.5
-f 'instances/sample_10.dat'
```

Instance Generator

If you are interested in creating new instances, a random instance generator is included with customizable size:

```
$ cabal run heuristics -- generator -n 10000 -f 'example.dat'
./example_10000.dat
```

There is the generate_instances.sh script that makes it easier to generate test samples.