Computational Experience with the Maximal Location Covering Problem

Abstract

On this paper, you can find heuristic approaches applied to the Maximal Covering Location Problems; this project could be useful to decision makers that want to generate a solution and try to improve a given solution as well in a short amount of time.

1 Introduction

Facility Location is a branch of Operations Research. This category of combinatorial optimization problems often deals with problems that seek to select the placement of a facility (often, from a given list of possibilities) that meets the best of certain constraints. These problems often consists of minimizing the total weighted distances from supplies and customers, and weights are representative of the difficulty of transporting materials.

Then, we have the Maximal Covering Location Problem. This problem is a derivative of Facility Location sort; however, this problems seeks to maximize the customers covered by a given set of facilities,instead of minimizing the distance between customers and supplies.

The Maximal Covering Location Problem (MCLP), is a classic problem in location analysis with applications in a good number of fields, such as health care, emergency planning, ecology, statistical classification, homeland security, etc.

The objective of the MCLP is to maximize the population covered by a given set of facilities; however, this could work as an analogy to another problems that involve "supply" and "demand" points to maximize the demand as much as possible.

Formally, this problem is described in Church and ReVelle 1974 [1] as follows: "The maximal covering location problem seeks the maximum population which can be served within a stated service distance or time given a limited number of facilities."

In this work, MCLP is analysed and solved on random instances generated within a certain range, by an implemented instance generator. A Constructive Heuristic approach was used to generate a solution, and a Local Search algorithm was implemented to try to improve the feasible solution generated by the constructive. Overall, the effectiveness and computational time of the implemented algorithms on small, medium and large instances were analysed. More details are given in the following sections.

2 Problem description - Mathematical model

According to the model defined on Church and ReVelle 1974 [1], this problem is defined on a network of nodes and arcs.

A mathematical formulation of this problem can be stated as Eq. 1:

$$max\ z = \sum_{i \in I} y_i \tag{1}$$

subject to:

$$x_j \in (0,1), j \in J \tag{2}$$

$$y_i \in (0,1), i \in I \tag{3}$$

$$\sum_{j \in J} x_j = S \tag{4}$$

$$\sum_{j \in N_i} x_j \geq y_i, N_i = \{j \in J : d_{ij} \leq r\} \tag{5}$$

where

- x_j : denotes that the j-th facility site is selected (1) or not (0). Boolean variable.
- y_i : denotes that if the demand point i is covered by any facility site within radius r (1) or not (0). Boolean variable.
- I : the set of demand or population points. Array of indexes referencing the coordinates of each demand point.
- J : the set of facility candidate sites. Array of indexes referencing the supply points.
- $d(i,j)$: the Euclidean distance from a demand point i to facility site j. 2-D matrix.
- S : the number of desired facilities to be selected from the candidate sites J. Integer variable.
- r : Maximal distance where a demand point from set I is covered by a facility site from set J. Integer variable.

The objective is to maximize the number of demand points covered within a desired service distance, as depicted in 1. A selection criteria for a site is described in restriction 2, where as restriction 3 alludes to all the demands point covered within a specified radius. Ideally, the sum of the selected sites must be equal to the desired by a decision maker as stated on restriction 4; however, it's possible that all the demand/population points are covered before this restriction is satisfied. Restriction 5 indicates that the distance between covered nodes by the selected sites must be under the specified radius from the user.

2.1 Example

The scatter plot shown in the figure 1 was generated by an instance generator coded in the programming language Python. The range of the population points are within a range of 5-500 units, and the potential candidate sites were generated inside the convex hull of the demand points. There are exactly 500 demand points, and 20 candidate sites. The values were generated randomly.

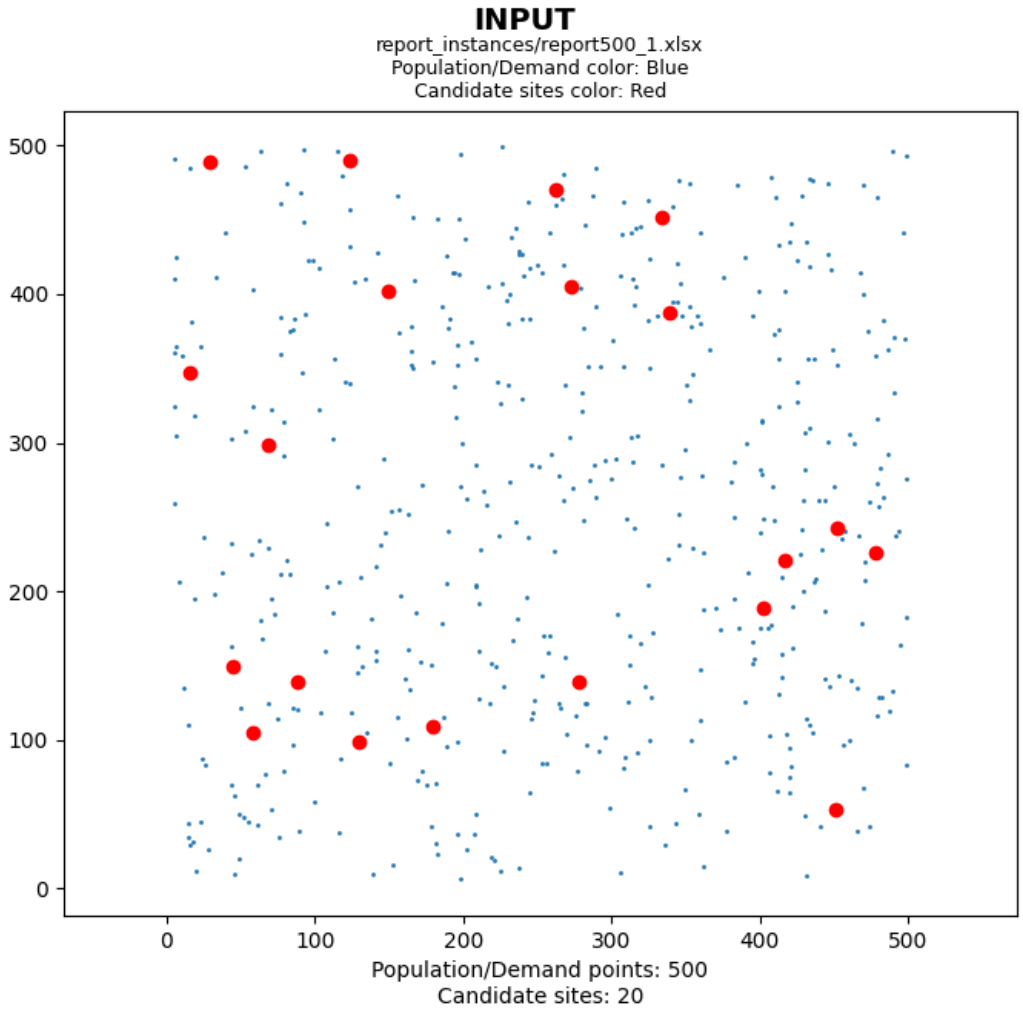

Fig. 1: Random instance of population and candidate sites points

As shown, the population points are illustrated in the scatter plot in blue, and the potential candidate sites are filled with red.

Once the input is succesfully computed, there are a few ways to figure out a first feasible solution. As seen in Church and ReVelle 1974 [1], a first heuristic approach called Greedy Adding (GA) was considered. This approach starts with an empty solution, and starts to iterate each candidate site until the one with maximal covered population is found; then, this site is added to the solution. It keeps iterating until either the S facilities have been selected or all the population is covered. However, there is a problem with this approach: the time complexity is denoted as $O(n\ logn)$, which means that it's not efficient for big instances.

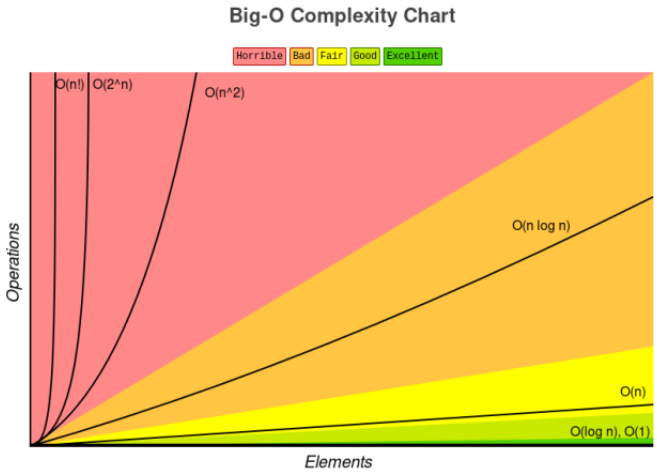

Fig. 2: Big O Chart, from https://www.bigocheatsheet.com/

Instead of using the Greedy Adding algorithm proposed, a Constructive Heuristic approach was implemented for generating first a feasible solution that meet the constraints. Then, a Local Search move approach tries to improve this solution. These heuristics are discused in more detail on the following sections.

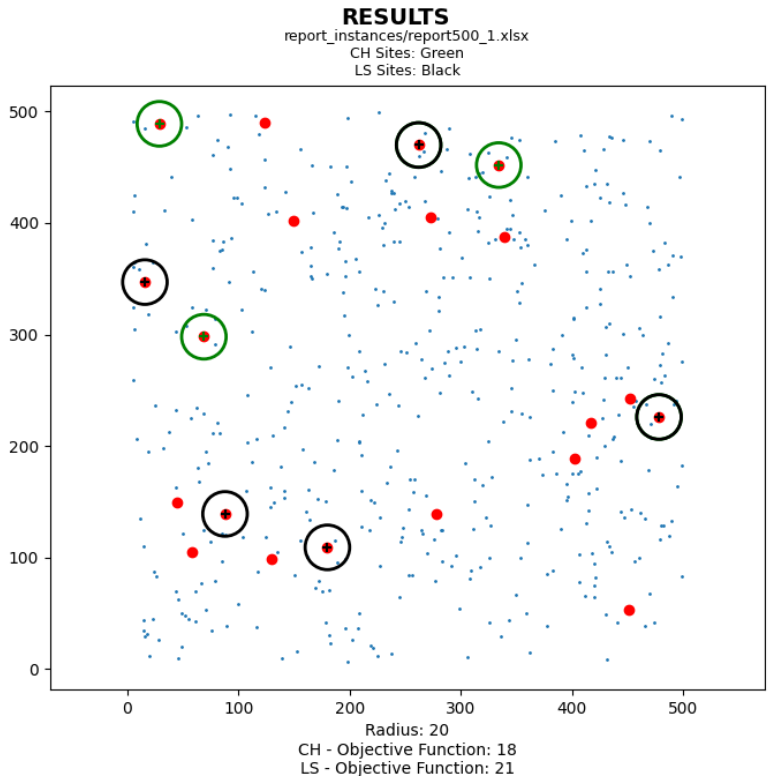

Fig. 3: Random instance of population and candidate sites points

The scatter plot shown in the figure 3 illustrates the output of the applied heuristics mentioned previously. The sites and population covered of the Constructive Heuristic approach are marked with green circles, and black circles illustrate the ones covered by Local Search.

3 Heuristics

3.1 Description of Constructive Heuristic

By definition, a constructive heuristic is a type of heuristic that starts with an empty solution and keeps extending with each iteration until a feasible solution is obtained.

The implementation of the Constructive Heuristic made for this problem receives as input the demand points I, the potential candidate sites J, the desired sites to be selected S and the *radius* of coverage. The Constructive Heuristic implemented for the MCLP is described in the Alg. 1.

Algorithm 1: Constructive Heuristic

Input:

- $I \leftarrow [1, 2, 3...i]$
- $J \leftarrow [1, 2, 3...j]$
- $S \leftarrow int\ value$
- $radius \leftarrow int\ value$

Output:

- $selected\_sites \leftarrow [1, 2, 3...S]$
- $objective\_function \leftarrow Population\ covered\ by\ selected\_sites$

Pseudocode:

```
selected_sites ← []
population_size ← length(I)
for site in J do
    current_covered_nodes ← Compute covered nodes(selected_sites)
    objective_function = length(current_covered_nodes)
    r1 = covered_nodes(under radius(site)) > 0
    r2 = length(selected_sites) < S
    r3 = objective_function < population_size
    if (r1 and r2 and r3) == True then
        Append site to selected_sites
    else if r1 == False then
        pass
    else
        return selected_sites, objective_function
        break for
end
```

3.2 Description of Local Search

Before describing the move of the Local Search (LS) algorithm used on this problem, it's important to know what a Local Search is; the main thing that differs between Local Search and Constructive heuristics is that Local Search needs a starting solution, whereas Constructive heursitics starts with a empty solution and build it from scratch.

In general, the objective of Local Search moves is to improve a given solution. Best-found strategy was used in the LS implementation for the MCLP, described in the alg. 2.

Algorithm 2: Local Search heuristic

Input:

- $objF\_value \leftarrow current\ objective\ function$
- $objF\_sites \leftarrow [1, 2, 3...S]$
- $free\_sites \leftarrow [1, 2, 3...j]\ from\ J\ not\ in S$
- $sites\_with\_obj\_F \leftarrow$ {j in J: obj_F of j} dict

Output:

- $new\_sites\_set \leftarrow [1, 2, 3...new(S)]$
- $new\_objF \leftarrow Population\ covered\ by\ new\_sites\_set$

Move(j,k): : Replace open facility j (currently in objF_sites[]) by a free candidate site k (currently in free_sites[])

```
objF_copy ← copy(objF_value)
objF_sites_copy ← copy(objF_sites)
free_sites_copy ← copy(free_sites)
sites_with_objF_copy ← copy(sites_with_objF)

for site in objF_sites_copy do
    site_objF ← sites_with_objF_copy[site]
    for free_site in free_sites_copy do
        free_site_objF = sites_with_objF_copy[free_site]
        if free_site_objF > site_objF then
            objF_sites_copy[site] ← free_site
        else
            pass
    end
end

new_sites_set ← objF_sites_copy
new_objF ← Population covered by new_sites_set
if free_site_objF > site_objF then
    return new_sites_set, new_objF
else
    print "Solution couldn't be improved"
```

4 Experiments

Experiments and testing of implementations of Constructive and Local Search heuristics seen in alg. 1 and alg. 2 were coded on Python programming language, and they were tested on a computer with an Intel Core i7-6500u processor. Basic frequency of this processor is between 2.5-3.1 GHz.

INSTANCE	CH_OF	CH_time (cpu sec)	LSH_OF	LS_time (cpu sec)	ABSOLUTE IMP	RELATIVE IMP
small500_1.xlsx	54	0.349726	63	0.000037	9	16.67%
small500_2.xlsx	73	0.036091	76	0.000232	3	4.11%
small500_3.xlsx	65	0.033529	69	0.000201	4	6.15%
small500_4.xlsx	61	0.032695	62	0.000249	1	1.64%
small500_5.xlsx	54	0.032827	63	0.000157	9	16.67%
small500_6.xlsx	68	0.014754	76	0.000169	8	11.76%
small500_7.xlsx	66	0.013740	73	0.000156	7	10.61%
small500_8.xlsx	64	0.013012	68	0.000172	4	6.25%
small500_9.xlsx	51	0.032844	71	0.000202	20	39.22%
small500_10.xlsx	54	0.015027	61	0.000120	7	12.96%
small500_11.xlsx	55	0.033229	59	0.000101	4	7.27%
small500_12.xlsx	47	0.016532	58	0.000529	11	23.40%
small500_13.xlsx	59	0.034043	74	0.000201	15	25.42%
small500_14.xlsx	49	0.033183	76	0.000244	27	55.10%
small500_15.xlsx	58	0.014746	78	0.000183	20	34.48%
small500_16.xlsx	44	0.015279	70	0.000199	26	59.09%
small500_17.xlsx	62	0.014306	71	0.000162	9	14.52%
small500_18.xlsx	49	0.013388	70	0.000186	21	42.86%
small500_19.xlsx	64	0.014963	66	0.000165	2	3.12%
small500_20.xlsx	72	0.014259	72	0.000110	0	0.00%

Table 1: Small instances - Demand: 500[5-500], Candidate sites: 10, Selected sites: 4, Radius: 50, Avg. improvement = 19.57%

INSTANCE	CH_OF	CH_time (cpu sec)	LSH_OF	LS_time (cpu sec)	ABSOLUTE IMP	RELATIVE IMP
medium5000_1.xlsx	1299	0.784653	1331	0.000099	32	2.46%
medium5000_2.xlsx	1001	0.672806	1103	0.000057	102	10.19%
medium5000_3.xlsx	1195	0.693486	1242	0.000081	47	3.93%
medium5000_4.xlsx	987	0.638173	1104	0.000085	117	11.85%
medium5000_5.xlsx	1177	0.684354	1216	0.000074	39	3.31%
medium5000_6.xlsx	1111	0.674954	1196	0.000090	85	7.65%
medium5000_7.xlsx	1027	0.665669	1192	0.000092	165	16.07%
medium5000_8.xlsx	971	0.683765	1067	0.000089	96	9.89%
medium5000_9.xlsx	1027	0.625440	1133	0.000077	106	10.32%
medium5000_10.xlsx	1138	0.641908	1183	0.000048	45	3.95%
medium5000_11.xlsx	1067	0.625906	1256	0.000079	189	17.71%
medium5000_12.xlsx	1304	0.642514	1304	0.000034	0	0.00%
medium5000_13.xlsx	1157	0.645659	1197	0.000076	40	3.46%
medium5000_14.xlsx	1021	0.674440	1129	0.000071	108	10.58%
medium5000_15.xlsx	1037	0.680675	1136	0.000089	99	9.55%
medium5000_16.xlsx	1036	0.676522	1075	0.000084	39	3.76%
medium5000_17.xlsx	1133	0.697644	1185	0.000046	52	4.59%
medium5000_18.xlsx	1038	0.693270	1175	0.000090	137	13.20%
medium5000_19.xlsx	1157	0.697294	1223	0.000100	66	5.70%
medium5000_20.xlsx	1317	0.699786	1328	0.000094	11	0.84%

Table 2: Medium instances - Demand: 5000[5-1000], Candidate sites: 20, Selected sites: 8, Radius: 100 Avg. improvement = 7.45%

INSTANCE	CH_OF	CH_time (cpu sec)	LSH_OF	LS_time (cpu sec)	ABSOLUTE IMP	RELATIVE IMP
large10000_1.xlsx	3582	2.880030	3873	0.000139	291	8.12%
large10000_2.xlsx	3825	2.851862	4041	0.000103	216	5.65%
large10000_3.xlsx	3528	2.858097	3899	0.000088	371	10.52%
large10000_4.xlsx	3113	2.768064	3330	0.000090	217	6.97%
large10000_5.xlsx	3751	2.967234	3955	0.000143	204	5.44%
large10000_6.xlsx	3001	2.931466	3140	0.000120	139	4.63%
large10000_7.xlsx	3599	2.907708	3705	0.000105	106	2.95%
large10000_8.xlsx	3882	2.686900	4013	0.000148	131	3.37%
large10000_9.xlsx	3298	2.898382	3387	0.000085	89	2.70%
large10000_10.xlsx	2734	2.731435	2942	0.000169	208	7.61%
large10000_11.xlsx	4031	3.139189	4103	0.000099	72	1.79%
large10000_12.xlsx	3515	2.914604	3661	0.000090	146	4.15%
large10000_13.xlsx	3505	3.009856	3697	0.000121	192	5.48%
large10000_14.xlsx	3307	2.829949	3456	0.000105	149	4.51%
large10000_15.xlsx	3846	3.067337	3983	0.000141	137	3.56%
large10000_16.xlsx	3238	3.004198	3349	0.000083	111	3.43%
large10000_17.xlsx	3339	3.088515	3623	0.000113	284	8.51%
large10000_18.xlsx	3836	2.865598	4018	0.000088	182	4.74%
large10000_19.xlsx	3584	2.766983	3656	0.000083	72	2.01%
large10000_20.xlsx	3513	2.998733	3636	0.000113	123	3.50%

Table 3: Large instances - Demand: 10000[5-1000], Candidate sites: 50, Selected sites: 15, Radius: 100, Avg. improvement = 4.98%

5 Conclusions

At first, it's tempting to say that implementing constructive and local search may be easy. Often, we assume that every problem can be solved. But then we have the NP-hard complete problems. The most important things of a good algorithm are efficiency and time complexity; however, the Maximal Covering Location Problems can be solved by a few approaches that are not necessarily exponential, but those are not as efficient as the heuristics presented on this paper. When constructive and local search heuristics are joined together on a set location problem as MCLP, the results are really good and easy to compute.

References

[1] Church, R. and ReVelle, C. (1974). The maximal covering location problem. In *Papers of the regional science association*, volume 32, pages 101–118. Springer-Verlag.