UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

18.4/20

FIME

FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA ELECTRICA

Selected Topics of Optimization

Doctor: **Roger Zirahuen Rios Mercado**

Computational Experience with Heuristics for the Maximum Covering Location Problem

Matricula	Integrates (TEAM I)
2169075	David Alejandro Ortiz Fuentes
2121131	Gabriel Sanchez Rivera
2226584	Juan Jose Montoya Zapata
2019069	Cristopher Led Juarez Luna

Due date: 22/05/2026

1. Introduction

The Maximum Covering Location Problem (MCLP) was originally introduced by Church and ReVelle [1] in 1974 as a discrete facility location model designed to address coverage maximization under resource limitations. Since its introduction, it has become one of the most influential models in location science and public service system design.

Unlike classical facility location problems such as the p-median or p-center, which focus on minimizing total distance or worst-case distance, the MCLP shifts the objective toward maximizing the total demand covered within a predefined service distance or response time threshold. This distinction is critical in real-world applications where full coverage is unattainable due to budget, infrastructure, or logistical constraints.

The MCLP is particularly relevant in public-sector planning contexts, including:

- Emergency medical service deployment
- Fire station placement
- Police patrol districting
- Disaster response center allocation
- Vaccination and health clinic positioning
- Telecommunications tower siting

In these systems, decision-makers often face a fixed budget that limits the number of facilities that can be installed. The strategic question is therefore not how to cover everyone, but rather how to cover as much demand as possible [1].

From a theoretical standpoint, the MCLP belongs to the class of combinatorial optimization problems and is known to be NP-hard [4]. This implies that exact solution approaches based on Mixed-Integer Linear Programming (MILP) become computationally expensive for medium and large instances. Consequently, heuristic and metaheuristic approaches such as greedy algorithms, Lagrangian relaxation, tabu search, genetic algorithms, and variable neighborhood search are frequently used to obtain high-quality solutions within reasonable computational time [2, 3, 5].

This project focuses on the implementation and computational evaluation of constructive heuristics and local search methods for the MCLP, using randomly generated instances across a range of sizes and parameters. The goal is to analyze the trade-off between solution quality and computational effort for the proposed methods.

2. Problem Description

The Maximum Covering Location Problem arises in situations where a set of demand points is distributed geographically, a finite set of candidate facility locations is available, each facility can cover demand points located within a predefined service radius, and only a limited number of facilities can be opened. The decision problem consists of selecting exactly p facilities from the candidate set such that the total demand covered is maximized [1].

Coverage is defined in binary terms: a demand node is considered covered if at least one selected facility is located within the acceptable service distance. This structure reflects practical constraints such as budget limitations (restricted number of facilities), service standards (maximum response time), and discrete infrastructure planning (facilities can only be located at predefined sites).

2.1 Data (Input Parameters)

Let:

I = set of demand nodes

J = set of candidate facility locations

a_i = demand (population) at node $i \in I$

p = maximum number of facilities that can be opened

d_{ij} = distance between demand node i and facility location j

S = service coverage distance threshold

$$N_i = \{j \in J : d_{ij} \leq S\}$$

That is, the set of facilities capable of covering demand node i.

2.2 Decision Variables

$x_j = 1$ if facility j is opened, 0 otherwise

$y_i = 1$ if demand node i is covered, 0 otherwise

2.3 Objective Function

The goal is to maximize the total demand covered:

$$\textbf{max} \ \Sigma_{i \in I} \ a_i \cdot y_i$$

2.4 Constraints

1. Only p facilities may be opened:

$$\Sigma_{j \in J} \ x_j = p$$

2. A demand node can only be covered if at least one covering facility is opened:

$$y_i \leq \Sigma_{j \in N_i} \ x_j \quad \forall i \in I$$

3. Binary restrictions:

$$x_j \in \{0,1\} \quad \forall j \in J$$

$$y_i \in \{0,1\} \quad \forall i \in I$$

2.5 Complete Mathematical Model

$$\textbf{max} \ \Sigma_{i \in I} \ a_i \cdot y_i$$

subject to:

$$\Sigma_{j \in J} \ x_j = p$$

$$y_i \leq \Sigma_{j \in N_i} \ x_j \quad \forall i \in I$$

$$x_j \in \{0,1\} \quad \forall j \in J, \quad y_i \in \{0,1\} \quad \forall i \in I$$

This formulation corresponds to the classical MCLP model as presented by Church and ReVelle [1].

3. Small Example

To illustrate the model, consider a small instance with five demand nodes and four candidate facility locations:

$I = \{1, 2, 3, 4, 5\}$, $J = \{A, B, C, D\}$, $p = 2$

Node	Demand
1	10
2	15
3	20
4	12
5	18

Coverage sets (N_i):

$N_1 = \{A, B\}$, $N_2 = \{A, C\}$, $N_3 = \{B, C\}$, $N_4 = \{C, D\}$, $N_5 = \{D\}$

Example Feasible Solution

Suppose we select $S = \{C, D\}$, thus $xC = 1$, $xD = 1$.

Coverage evaluation:

- Node 1 → not covered ($N_1 = \{A, B\}$, neither A nor B selected)
- Node 2 → covered by C
- Node 3 → covered by C
- Node 4 → covered by C or D
- Node 5 → covered by D

Therefore: $y_2 = y_3 = y_4 = y_5 = 1, \quad y_1 = 0$

Objective value: 15 + 20 + 12 + 18 = 65

The total covered demand is 65 out of 75, achieving an 86.7% coverage rate. This example illustrates how the objective function is evaluated for a feasible solution to the MCLP.

4. Heuristics Description

To address the Maximum Covering Location Problem, two classical constructive heuristics were implemented: the Greedy Adding (GA) algorithm and the Greedy Adding with Substitution (GAS) algorithm. Both methods follow a greedy philosophy, building a solution incrementally by making locally optimal decisions at each step. While neither guarantees a globally optimal solution, they produce high-quality feasible solutions in polynomial time, making them well-suited for large-scale instances [1, 2].

4.1 Greedy Adding (GA)

The Greedy Adding algorithm is a constructive heuristic that builds a solution from scratch by iteratively selecting the single best facility to open at each step [1]. The algorithm proceeds as follows:

- Initialize an empty solution set $S = \emptyset$ (no facilities open).
- At each iteration, compute the marginal gain of opening each candidate facility $j \notin S$, defined as the additional demand newly covered by j that is not already covered by any facility in S.
- Select the facility j^* that maximizes the marginal gain and add it to S.
- Repeat until exactly p facilities have been selected.

The key concept behind this algorithm is the notion of marginal gain, which avoids double-counting demand already served by previously opened facilities. By always choosing the candidate with the highest immediate benefit, GA tends to produce good solutions quickly. However, because it never revises past decisions, it can commit to early choices that later prove suboptimal when considered in the context of the full solution.

Time complexity: $O(p \cdot M \cdot N)$, where p is the number of facilities to open, M is the number of candidates, and N is the number of demand nodes.

4.2 Greedy Adding with Substitution (GAS)

The Greedy Adding with Substitution algorithm extends GA by incorporating a substitution phase after each greedy addition [1, 2]. This enhancement allows the algorithm to reconsider and correct earlier decisions, addressing the main limitation of pure greedy construction. The procedure is:

- Perform a greedy addition step identical to GA, selecting and opening the facility j* with the highest marginal gain.
- Enter the substitution phase: evaluate all possible swaps of a currently open facility jout $\in$ S for a closed facility jin $\notin$ S. Compute the net change in covered demand for each swap.
- If a swap with a strictly positive net gain exists, perform the best such swap (update S accordingly).
- Repeat the substitution phase until no improving swap can be found.
- Return to the greedy addition step and continue until p facilities are open.

The substitution phase effectively implements a local search within each construction iteration. By checking whether any open facility should be replaced by a better alternative given the current configuration, GAS can escape poor intermediate states that would trap the GA. In practice, GAS consistently achieves equal or better objective values than GA, at the cost of increased computational effort [2].

Time complexity: $O(p^2 \cdot M \cdot N)$ in the worst case, since the substitution phase may run multiple rounds per greedy iteration, each requiring $O(p \cdot M \cdot N)$ evaluations.

4.3 Local Search (LS)

The Local Search (LS) procedure is applied as a post-processing improvement phase after the GAS construction. It belongs to the family of neighborhood search methods, which explore solutions in the vicinity of a current solution by making small structural changes [5].

For the MCLP, the neighborhood of a solution S is defined as the set of all solutions obtainable by removing one open facility and replacing it with one closed candidate. This is known as a single-facility swap neighborhood. The LS procedure operates as follows:

- Start from the GAS solution S with its associated coverage value z(S).

- Define the neighborhood $N(S) = \{ S' : S' = (S \setminus \{jout\}) \cup \{jin\}, jout \in S, jin \notin S \}$.
- Evaluate all neighbors $S' \in N(S)$ and compute $z(S')$ for each.
- If any neighbor S' has $z(S') > z(S)$, accept the first such improvement found (first-improvement strategy) and set $S \leftarrow S'$.
- Repeat until no neighbor improves the current solution — the solution is then a local optimum with respect to single-facility swaps.

The first-improvement strategy was chosen over best-improvement because it reduces the number of evaluations per iteration while still guaranteeing convergence to a local optimum. A solution is called 1-optimal if no single-facility swap can improve it; this is the termination criterion of the LS procedure.

In practice, the LS phase serves as a quality verification step: if GAS already produces a 1-optimal solution, the LS terminates immediately without performing any swap, confirming the quality of the constructive phase. This was observed consistently in the computational experiments reported in Section 5.

Time complexity: $O(p \cdot M \cdot N)$ per improvement iteration. In the worst case, $O(k \cdot p \cdot M \cdot N)$ where k is the number of improving swaps accepted before reaching a local optimum.

5. Computational Work

This section describes the computational experiments conducted to evaluate the performance of the proposed heuristics. All algorithms were implemented in Python and executed on randomly generated MCLP instances. The instance generator was designed to allow full control over problem parameters, enabling a systematic analysis of algorithm behavior across a range of instance sizes and configurations.

5.1 Instance Generation

Random instances were generated using the custom instance_generator.py module. Demand nodes and candidate facility locations were placed uniformly at random within a square region of side length L = 100. Demand weights were drawn proportionally from a uniform random distribution scaled to a fixed total demand, ensuring comparability across instances. The coverage relationship between nodes and candidates was determined by Euclidean distance: a candidate j covers demand node i if and only if dist(i, j) ≤ S.

Three sets of instances were generated, increasing in problem size and difficulty. The parameter values were chosen specifically to create instances where the coverage density is moderate — meaning S is small enough relative to the area that not all demand can be trivially covered — which allows meaningful differentiation between the heuristic methods. If S is too large, all methods achieve 100% coverage and no comparison is possible. The following parameter ranges were used:

Parameter	Description	Values Tested
N	Number of demand nodes	100, 150, 200
M	Number of candidate locations	60, 80, 100
p	Facilities to open	5, 6 (small) / 7 (medium) / 8 (large)
S	Service radius	15 (small) / 17 (medium) / 18 (large)
L	Area side length	100
Total demand	Sum of all node demands	5,000 – 8,000
Instances/set	Number of random instances per set	20 per set (60 total)

Each instance was generated with a fixed random seed to ensure reproducibility. Each set contains exactly 20 instances, for a total of 60 instances. The service radius S was deliberately chosen differently for each set — S=15 for small, S=17 for medium, S=18 for large — to maintain a moderate coverage density as N increases, ensuring that the problem remains challenging and that meaningful differences between methods can be observed.

5.2 Algorithm Implementation

The solution pipeline was implemented across five Python modules: instance_generator.py (random instance creation), data_loader.py (shared I/O utilities and coverage matrix computation), heuristics.py (GA and GAS), local_search.py (single-swap local search), and main.py (pipeline orchestrator). The pipeline is sequential: the generator writes a structured .txt file that is consumed by the heuristics, whose output is then passed as the starting point for the local search.

The coverage matrix is computed at load time from node and facility coordinates using Euclidean distance. This avoids recomputing distances at every algorithm step, reducing the computational bottleneck to simple list lookups during objective function evaluations.

5.3 Results

A total of 60 instances were tested across three sets of increasing problem size — 20 small (N=100), 20 medium (N=150), and 20 large (N=200). For each instance, the three methods (GA, GAS, GAS+LS) were executed and their coverage rates and CPU times recorded.

Set 1 – Small Instances (N=100, M=60, S=15)

Exp	N	p	Tot. Demand	GA %	GAS %	GAS+LS %	Improvement
1	100	5	5,000	46.94%	46.94%	46.94%	–
2	100	5	5,000	48.64%	48.64%	48.64%	–
3	100	5	5,000	56.88%	**57.88% ✓**	57.88%	**+1.00%**
4	100	5	5,000	51.60%	51.60%	51.60%	–

5	100	5	5,000	56.78%	56.78%	56.78%	–
6	100	5	5,000	53.60%	53.60%	53.60%	–
7	100	5	5,000	56.22%	56.22%	56.22%	–
8	100	5	5,000	50.30%	50.30%	50.30%	–
9	100	5	5,000	49.16%	**49.54% ✓**	49.54%	**+0.38%**
10	100	5	5,000	49.38%	**50.14% ✓**	50.14%	**+0.76%**
11	100	6	5,000	63.68%	63.68%	63.68%	–
12	100	6	5,000	65.48%	65.48%	65.48%	–
13	100	6	5,000	54.94%	54.94%	54.94%	–
14	100	6	5,000	60.32%	60.32%	60.32%	–
15	100	6	5,000	60.10%	60.10%	60.10%	–
16	100	6	5,000	59.56%	59.56%	59.56%	–
17	100	6	5,000	67.96%	67.96%	67.96%	–
18	100	6	5,000	54.62%	54.62%	54.62%	–
19	100	6	5,000	54.94%	54.94%	54.94%	–
20	100	6	5,000	57.84%	57.84%	57.84%	–

Set 2 – Medium Instances (N=150, M=80, p=7, S=17)

Exp	N	p	Tot. Demand	GA %	GAS %	GAS+LS %	Improvement
1	150	7	7,000	76.14%	76.14%	76.14%	–
2	150	7	7,000	69.54%	**69.76% ✓**	69.76%	**+0.22%**
3	150	7	7,000	73.56%	**73.60% ✓**	73.60%	**+0.04%**
4	150	7	7,000	70.79%	70.79%	70.79%	–
5	150	7	7,000	73.20%	**76.40% ✓**	76.40%	**+3.20%**
6	150	7	7,000	76.49%	76.49%	76.49%	–
7	150	7	7,000	71.97%	**73.30% ✓**	73.30%	**+1.33%**
8	150	7	7,000	74.44%	74.44%	74.44%	–

9	150	7	7,000	65.09%	**69.64% ✓**	69.64%	**+4.55%**
10	150	7	7,000	70.59%	**71.97% ✓**	71.97%	**+1.38%**
11	150	7	7,000	72.76%	72.76%	72.76%	–
12	150	7	7,000	67.33%	**67.39% ✓**	67.39%	**+0.06%**
13	150	7	7,000	71.61%	71.61%	71.61%	–
14	150	7	7,000	72.13%	**73.39% ✓**	73.39%	**+1.26%**
15	150	7	7,000	67.39%	67.39%	67.39%	–
16	150	7	7,000	69.33%	**71.53% ✓**	71.53%	**+2.20%**
17	150	7	7,000	67.64%	**68.64% ✓**	68.64%	**+1.00%**
18	150	7	7,000	75.67%	75.67%	75.67%	–
19	150	7	7,000	76.74%	76.74%	76.74%	–
20	150	7	7,000	73.36%	**74.63% ✓**	74.63%	**+1.27%**

Set 3 – Large Instances (N=200, M=100, p=8, S=18)

Exp	N	p	Tot. Demand	GA %	GAS %	GAS+LS %	Improvement
1	200	8	8,000	73.98%	**76.80% ✓**	76.80%	**+2.82%**
2	200	8	8,000	81.96%	**84.05% ✓**	84.05%	**+2.09%**
3	200	8	8,000	82.80%	**83.06% ✓**	83.06%	**+0.26%**
4	200	8	8,000	81.94%	**84.66% ✓**	84.66%	**+2.72%**
5	200	8	8,000	77.11%	77.11%	77.11%	–
6	200	8	8,000	83.20%	**84.12% ✓**	84.12%	**+0.92%**
7	200	8	8,000	80.42%	**81.88% ✓**	81.88%	**+1.46%**
8	200	8	8,000	81.76%	**85.04% ✓**	85.04%	**+3.28%**
9	200	8	8,000	81.51%	81.51%	81.51%	–
10	200	8	8,000	83.21%	**84.86% ✓**	84.86%	**+1.65%**
11	200	8	8,000	85.92%	**86.96% ✓**	86.96%	**+1.04%**

12	200	8	8,000	85.11%	**85.47% ✓**	85.47%	**+0.36%**
13	200	8	8,000	83.43%	83.43%	83.43%	–
14	200	8	8,000	80.38%	**80.69% ✓**	80.69%	**+0.31%**
15	200	8	8,000	86.85%	86.85%	86.85%	–
16	200	8	8,000	78.97%	**82.44% ✓**	82.44%	**+3.47%**
17	200	8	8,000	81.60%	81.60%	81.60%	–
18	200	8	8,000	83.08%	**83.56% ✓**	83.56%	**+0.48%**
19	200	8	8,000	81.19%	**81.55% ✓**	81.55%	**+0.36%**
20	200	8	8,000	81.73%	**83.36% ✓**	83.36%	**+1.63%**

Note: ✓ indicates instances where GAS achieved strictly higher coverage than GA. GAS+LS = GAS in all 60 instances.

5.3.1 Summary by Instance Set

The average results per set across all 20 instances are summarized below:

Set	Avg GA %	Avg GAS %	Avg Improvement	GAS > GA
Set 1 – Small (N=100, S=15)	55.95%	56.05%	+0.11%	3 / 20
Set 2 – Medium (N=150, S=17)	71.79%	72.61%	+0.83%	11 / 20
Set 3 – Large (N=200, S=18)	81.81%	82.95%	+1.14%	15 / 20
Overall Average (60 instances)	**69.85%**	**70.54%**	**+0.69%**	**29 / 60**

5.3.2 Analysis and Discussion

Several important observations can be drawn from the experimental results:

GAS improvement increases with instance size. The number of instances where GAS outperformed GA increased consistently across sets: 3/20 in small, 11/20 in medium, and 15/20 in large. This clear progression confirms that the substitution phase becomes increasingly effective as the problem grows, because larger instances have more candidate swaps and a higher probability that early greedy choices become suboptimal as more facilities are added.

Most significant improvements in large instances. In Set 3 (N=200, S=18), GAS improved upon GA in 15 out of 20 instances with an average gain of +1.14% and individual improvements reaching up to +3.47 percentage points. This confirms that the substitution phase has the most impact when the problem has high combinatorial complexity.

GAS solutions are already 1-optimal. In all 60 tested instances, the Local Search phase did not improve the GAS solution — GAS+LS always equaled GAS. This means every GAS solution was already a local optimum with respect to single-facility swaps, validating the quality of GAS construction. The substitution phase embedded within GAS effectively performs the local search role during construction.

Computational efficiency. All three methods executed in under 0.05 seconds for all 60 instances. GA ran in under 0.001s, GAS under 0.03s, and LS under 0.01s even for N=200. This confirms the practical viability of all methods for real-world applications.

Effect of coverage density on improvement. The small instances with S=15 showed minimal GAS advantage (+0.11% avg) because the sparse coverage makes the problem very constrained — most greedy choices are forced and have little room for substitution improvement. As S increases relative to N (medium and large sets), the problem develops richer structure with overlapping coverage regions, creating more opportunities for beneficial swaps in the GAS substitution phase.

6. Conclusions

This work addressed the Maximum Covering Location Problem through a structured three-stage computational pipeline: random instance generation, constructive heuristic solution, and local search improvement. The following conclusions can be drawn from the implementation and experimental results.

Both the Greedy Adding and Greedy Adding with Substitution algorithms proved to be effective and computationally inexpensive methods for obtaining good feasible solutions to the MCLP. Their polynomial time complexity makes them practical for real-world instances where exact optimization methods such as integer programming would be prohibitively slow [3]. On the 60 test instances, coverage rates ranged from 47% to 87%, with an overall average of 69.85% for GA and 70.54% for GAS, demonstrating the quality of greedy construction across diverse problem configurations.

The substitution phase in GAS represents a meaningful and consistent improvement over pure GA, particularly for larger and more complex instances. The results show a clear progression: GAS outperformed GA in 3/20 small instances (+0.11% avg), 11/20 medium instances (+0.83% avg), and 15/20 large instances (+1.14% avg). This increasing trend with problem size confirms that the substitution phase is most effective when the problem has higher combinatorial complexity, where early greedy decisions have a stronger cascading impact on solution quality [1, 2].

The Local Search phase confirmed that GAS solutions are already 1-optimal with respect to single-facility swaps in all 60 tested instances. No improving swap was found in any case, meaning GAS+LS always equaled GAS. This is a notable result: it validates that the substitution phase embedded within GAS effectively performs the role of local search during construction, making a separate post-processing LS phase redundant for MCLP instances of these sizes.

From a computational efficiency standpoint, all three methods ran in under 0.05 seconds even for the largest N=200 instances, confirming their suitability for real-time or large-scale decision-support applications. The marginal increase in runtime from GA to GAS is negligible compared to the improvement in solution quality.

Future work could explore more sophisticated metaheuristics such as Tabu Search, simulated annealing, or Iterated Local Search with random perturbations to escape local optima in harder instances. Additionally, exact branch-and-bound or Lagrangean relaxation methods [2, 3] could be used to establish optimality gaps for the heuristic solutions, providing a more complete evaluation of solution quality. Incorporating real-world geographic data and heterogeneous service radii would further extend the applicability of this framework to practical facility location problems.

References.

Articles in Scientific Journals

[1] Church, R. L., and ReVelle, C. S. The maximal covering location problem. *Papers of the Regional Science Association*, 32(1):101–118, 1974.

[2] Galvão, R. D., and ReVelle, C. S. A Lagrangean heuristic for the maximal covering location problem. *European Journal of Operational Research*, 88(1):114–123, 1996.

[3] Galvão, R. D., Acosta Espejo, L. G., and Boffey, B. A comparison of Lagrangean and surrogate relaxations for the maximal covering location problem. *European Journal of Operational Research*, 124(2):377–389, 2000.

[4] Megiddo, N., and Supowit, K. J. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13(1):182–196, 1984.

[5] Murray, A. T. Maximal covering location problem: A review. *International Regional Science Review*, 39(1):5–27, 2016.

[6] Pirkul, H., and Schilling, D. A. The maximal covering location problem with capacities on total workload. *Management Science*, 37(2):233–248, 1991.

Books

[7] Drezner, Z., and Hamacher, H. W. (Eds.). *Facility Location: Applications and Theory.* Springer, Berlin, Germany, 2002.

[8] Farahani, R. Z., Asgari, N., Heidari, N., Hosseininia, M., and Goh, M. Covering problems in facility location: A review. *Computers & Industrial Engineering*, 62(1):368–407, 2012.