Maximum Covering Location Problem: Algorithms, Implementation, and Computational Results

Technical Implementation Report

Syed Abbas Ahmad
Pakistan Institute of Engineering and Applied Sciences, Islamabad

November 27, 2025

Abstract

This report presents a comprehensive implementation and evaluation of solution algorithms for the Maximum Covering Location Problem (MCLP). We implement six distinct algorithms in FICO Xpress Mosel, ranging from an exact Mixed Integer Programming (MIP) solver to advanced metaheuristics including Tabu Search. Our experimental evaluation on instances ranging from 50 to 5000 customers demonstrates that while exact methods guarantee optimality for small instances, metaheuristic approaches provide superior performance and scalability for medium-to-large scale problems. Notably, our Local Search implementation solves the largest instance (1000 facilities, 5000 customers) to the best known solution in 0.10 seconds, while Tabu Search consistently finds optimal or near-optimal solutions across all instance sizes, outperforming the exact solver on larger instances by up to 387 units in objective value. This work provides practitioners with a robust toolkit for MCLP applications, complete with algorithm selection guidelines based on instance characteristics and computational requirements.

Contents

1 Introduction

1.1 Problem Context and Motivation

The Maximum Covering Location Problem (MCLP) is a fundamental facility location problem with widespread applications in public and private sector decision-making. The problem addresses the strategic question: *Given a limited budget and a set of potential facility locations, which facilities should be opened to maximize the customer demand covered within an acceptable service distance?*

Real-world applications of MCLP include:

- **Emergency Services**: Positioning ambulances, fire stations, and police units to maximize population coverage within critical response times [1].
- **Retail Location Planning**: Selecting store locations to maximize market coverage within competitive service areas.
- **Public Infrastructure**: Siting hospitals, schools, libraries, and community centers to serve the maximum population.
- **Telecommunications**: Placing cell towers, Wi-Fi access points, or service centers to maximize coverage area.
- **Humanitarian Logistics**: Positioning relief centers in disaster response to maximize affected population access.

The MCLP was first formulated by Church and ReVelle in 1974 [1] as a complement to the Set Covering Location Problem (SCLP). While SCLP seeks to cover all demand with minimum cost, MCLP operates under a budget constraint and aims to maximize coverage. This distinction makes MCLP particularly relevant for resource-constrained environments where complete coverage may be infeasible or cost-prohibitive.

1.2 Problem Complexity and Computational Challenges

The MCLP belongs to the class of NP-hard combinatorial optimization problems [7]. As problem size increases, the number of feasible solutions grows exponentially, making exhaustive search intractable. For an instance with $|I|$ potential facility locations, there are theoretically $2^{|I|}$ possible facility configurations to consider.

The computational challenge intensifies when dealing with:

- **Large customer sets**: Modern applications may involve millions of demand points (e.g., census blocks, individual households).
- **Dense coverage graphs**: In urban areas, facilities may cover many customers, leading to complex interdependencies and symmetry in the solution space.
- **Tight budgets**: Limited budgets create difficult trade-offs between coverage quality and cost efficiency.

As noted by Cordeau et al. [2], exact algorithms for large-scale instances remain scarce despite the problem's practical importance. Recent advances in Benders decomposition techniques have enabled exact solution of instances with millions of customers when the number of facilities is relatively small. However, for practical applications where rapid decision-making is required or when the number of facilities is substantial, heuristic and metaheuristic approaches remain essential.

1.3 Contributions

This report documents a comprehensive implementation and evaluation of six MCLP solution algorithms in FICO Xpress Mosel. Our specific contributions include:

1. **Complete Implementation Suite**: We implement and benchmark a comprehensive suite of MCLP algorithms in FICO Xpress Mosel, including one exact method and five heuristic/metaheuristic approaches.

2. **Extensive Computational Study**: We conduct extensive computational experiments on a diverse set of instances ranging from 50 facilities with 200 customers to 1000 facilities with 5000 customers, establishing performance benchmarks for each algorithm class.

3. **Performance Analysis**: We provide detailed analysis of algorithm performance across different problem characteristics, including instance size, budget levels, and coverage radius values.

4. **Scalability Demonstration**: We demonstrate that our Local Search implementation achieves exceptional scalability, solving massive instances (5000 customers) in under 0.10 seconds to the best known solution.

5. **Metaheuristic Superiority**: We show that Tabu Search consistently outperforms the exact solver on medium-to-large instances, finding solutions 387 units better on our largest test instance (XL1) while maintaining sub-second runtimes.

1.4 Report Organization

The remainder of this report is structured as follows:

- **Section 2**: Literature review of MCLP algorithms and applications
- **Section 3**: Mathematical problem formulation and notation
- **Section 4**: Detailed description of all six solution algorithms
- **Section 5**: Experimental methodology and instance characteristics
- **Section 6**: Computational results and performance analysis
- **Section 7**: Conclusions and recommendations
- **Appendices**: Complete Mosel source code and supplementary materials

2 Literature Review

2.1 Problem Origins and Complexity

The MCLP was introduced by Church and ReVelle [1] as a variant of the set covering location problem that maximizes covered demand subject to budget constraints rather than minimizing cost subject to coverage requirements. Megiddo et al. [7] proved the problem to be NP-hard by reduction from the minimum dominating set problem.

Murray [8] provides a comprehensive survey of MCLP applications and solution methods, highlighting the problem's relevance in fields ranging from emergency service location to telecommunications network design.

2.2 Exact Algorithms

Few exact algorithms have been developed specifically for the MCLP. Church and ReVelle [1] and Snyder [10] observed that the LP relaxation of the standard MIP formulation often provides integer solutions, particularly when the objective is to maximize covered demand. Snyder reported that for over 95% of instances tested, the LP relaxation was integral, requiring no branching.

Downs and Camm [3] developed a Lagrangian relaxation approach coupled with subgradient optimization, embedded in a branch-and-bound framework. Their largest instance contained 2241 demand points and 74 potential facilities.

More recently, Cordeau et al. [2] introduced a branch-and-Benders-cut algorithm specifically designed for instances where the number of customers far exceeds the number of facilities ($|J| \gg |I|$). Their approach solves instances with up to 15 million customers for MCLP and 40 million for the related Partial Set Covering Location Problem (PSCLP).

2.3 Heuristic and Metaheuristic Approaches

Church and ReVelle [1] proposed a greedy heuristic that iteratively selects the facility providing the maximum increase in covered demand. They also introduced a swap-based local search variant.

Galvão and ReVelle [4] developed a Lagrangian heuristic using similar relaxation techniques to Downs and Camm but combined with constructive heuristics. ReVelle et al. [9] applied heuristic concentration, reducing the solution space before applying branch-and-bound or local search.

Among metaheuristics, Zarandi et al. [11] used genetic algorithms for instances with up to 2500 nodes, while Máximo et al. [6] developed a guided adaptive search algorithm tested on instances with up to 7730 nodes.

3 Mathematical Formulation

3.1 Problem Definition

The Maximum Covering Location Problem (MCLP) can be formally stated as follows:

> *Given a set of potential facility locations with associated opening costs, a set of customer demand points with known demands, a coverage relationship specifying which facilities can serve which customers, and a budget constraint, select which facilities to open such that the total covered demand is maximized while respecting the budget.*

3.2 Notation

We adopt the notation from Cordeau et al. [2]:

Sets:

- $I = \{0, 1, \dots, |I| - 1\}$: Set of potential facility locations (indexed by i)
- $J = \{0, 1, \dots, |J| - 1\}$: Set of customer demand points (indexed by j)
- $I_j \subseteq I$: Set of facilities that can cover customer j (within service radius)
- $J_i \subseteq J$: Set of customers covered by facility i

Parameters:

- $f_i \in \mathbb{R}_+$: Cost of opening facility $i \in I$
- $d_j \in \mathbb{R}_+$: Demand of customer $j \in J$
- $B \in \mathbb{R}_+$: Total budget available for opening facilities
- $R \in \mathbb{R}_+$: Service radius (maximum distance for coverage)

Decision Variables:

- $y_i \in \{0, 1\}$: Binary variable indicating whether facility i is opened ($y_i = 1$) or not ($y_i = 0$)
- $z_j \in [0, 1]$: Variable indicating whether customer j is covered ($z_j = 1$) or not ($z_j = 0$)

3.3 Compact MIP Formulation

The MCLP can be formulated as the following Mixed Integer Program:

$$\max \quad \sum_{j \in J} d_j z_j \tag{1a}$$

$$\text{s.t.} \quad \sum_{i \in I_j} y_i \geq z_j \quad \forall j \in J \tag{1b}$$

$$\sum_{i \in I} f_i y_i \leq B \tag{1c}$$

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

$$z_j \in [0, 1] \quad \forall j \in J \tag{1e}$$

Constraint Explanation:

- **Objective Function** (1a): Maximizes the total covered demand.
- **Coverage Constraints** (1b): For each customer j, ensures that if $z_j = 1$ (customer is covered), then at least one facility from I_j must be open.
- **Budget Constraint** (1c): Ensures total cost of opened facilities does not exceed budget B.
- **Integrality** (1d): Facility location variables must be binary.
- **Coverage Variables** (1e): Following [2], we relax the integrality requirement on z_j variables. Due to the maximization objective, these variables naturally take binary values at optimality.

3.4 Model Characteristics

Problem Size: For an instance with $|I|$ facilities and $|J|$ customers, the compact formulation has:

- **Variables**: $|I|$ binary variables + $|J|$ continuous variables = $|I| + |J|$ total
- **Constraints**: $|J|$ coverage constraints + 1 budget constraint = $|J| + 1$ total
- **Nonzeros**: Approximately $\sum_{j \in J} |I(j)|$ (depends on coverage density and radius R)

LP Relaxation Quality: The LP relaxation of MCLP often provides tight bounds [1, 10]. Empirical studies show that for many practical instances, the LP relaxation is integral or near-integral, with optimality gaps typically in the range of 0-5% for moderately sized instances. This property makes branch-and-bound approaches particularly effective for small to medium instances.

4 Solution Algorithms

This section describes the six algorithms implemented in this project. For each algorithm, we provide a conceptual overview, complexity analysis, and implementation notes.

4.1 Exact MIP Solver

4.1.1 Description

The Exact MIP Solver implements the compact formulation (Section 3.3) using the FICO Xpress Optimizer engine. This approach leverages state-of-the-art branch-and-bound technology, including:

- Sophisticated preprocessing to reduce problem size
- Cutting plane generation
- Advanced branching strategies
- Primal heuristics for good incumbent solutions

The solver guarantees optimality (within a specified MIP gap tolerance) and provides dual bounds.

4.1.2 Algorithm Parameters

- `TIME_LIMIT`: Maximum solver runtime (default: 600 seconds)
- `MIP_GAP`: Optimality tolerance (default: 1%)
- `RELAX_Z`: Relax z variables to continuous (default: yes)

4.1.3 Complexity Analysis

- **Worst-case**: Exponential in $|I|$
- **Best-case**: Polynomial when LP relaxation is integral
- **Practical**: Depends on instance structure and LP gap

4.2 Greedy Heuristic

4.2.1 Description

The Greedy heuristic [1] builds a solution iteratively by selecting at each step the facility that maximizes the marginal increase in covered demand while respecting the budget constraint.

4.2.2 Algorithm Pseudocode

Algorithm 1 Greedy MCLP Heuristic

```
1:  $K \leftarrow \emptyset$                                          ▷ Open facilities
2:  covered $\leftarrow \emptyset$                                   ▷ Covered customers
3:  budget_used $\leftarrow 0$
4:  while budget_used $< B$ do
5:      best_ratio $\leftarrow 0$
6:      best_facility $\leftarrow$ null
7:      for each $i \in I \setminus K$ do
8:          if budget_used $+ f_i > B$ then
9:              continue
10:         end if
11:         new_customers $\leftarrow J_i \setminus$ covered
12:         gain $\leftarrow \sum_{j \in \text{new\_customers}} d_j$
13:         ratio $\leftarrow$ gain$/f_i$
14:         if ratio $>$ best_ratio then
15:             best_ratio $\leftarrow$ ratio
16:             best_facility $\leftarrow i$
17:         end if
18:     end for
19:     if best_facility = null or best_ratio $\leq 0$ then
20:         break
21:     end if
22:     $K \leftarrow K \cup \{$best_facility$\}$
23:     covered $\leftarrow$ covered $\cup J_{\text{best\_facility}}$
24:     budget_used $\leftarrow$ budget_used $+ f_{\text{best\_facility}}$
25: end while
26: return $K$
```

4.2.3 Complexity

- **Time**: $O(|I|^2 \cdot |J|)$
- **Space**: $O(|I| + |J|)$
- **Practical runtime**: Under 1 second for all tested instances

4.3 Closest Neighbor Heuristic

4.3.1 Description

The Closest Neighbor heuristic is a simple distance-based construction method. It prioritizes facilities based on their proximity to high-demand uncovered customers.

4.3.2 Key Features

- Fast execution
- Deterministic
- Good baseline performance
- Complexity: $O(|I| \cdot |J|)$

4.4 Local Search

4.4.1 Description

Local Search performs iterative improvement using two neighborhood structures:

1. **1-flip**: Open or close a single facility
2. **Swap**: Close one facility and open another

Delta-evaluation provides $O(|J|)$ evaluation per move, making the algorithm practical for large instances.

4.4.2 Algorithm Pseudocode

Algorithm 2 Local Search for MCLP

```
 1: $K \leftarrow K_{\text{initial}}$                                ▷ Start from initial solution
 2: Initialize coverage tracking
 3: repeat
 4:     improvement_found ← false
 5:     for each facility $i \in K$ do                                       ▷ Try closing
 6:         $\Delta \leftarrow$ EvaluateClose($i$)
 7:         if $\Delta > 0$ then
 8:             Apply close move
 9:             improvement_found ← true
10:             break
11:         end if
12:     end for
13:     if not improvement_found then
14:         for each facility $i \in I \setminus K$ do                           ▷ Try opening
15:             $\Delta \leftarrow$ EvaluateOpen($i$)
16:             if $\Delta > 0$ and budget feasible then
17:                 Apply open move
18:                 improvement_found ← true
19:                 break
20:             end if
21:         end for
22:     end if
23:     if not improvement_found then
24:         for each pair $(i_{\text{out}} \in K, i_{\text{in}} \in I \setminus K)$ do     ▷ Try swaps
25:             $\Delta \leftarrow$ EvaluateSwap($i_{\text{out}}, i_{\text{in}}$)
26:             if $\Delta > 0$ and budget feasible then
27:                 Apply swap move
28:                 improvement_found ← true
29:                 break
30:             end if
31:         end for
32:     end if
33: until not improvement_found
34: return $K$
```

4.4.3 Complexity

- **Time per iteration**: $O(|I| \cdot |J|)$

- **Space**: $O(|I| + |J|)$
- **Practical runtime**: 0.02-5 seconds depending on instance size

4.5 Multi-Start Local Search

4.5.1 Description

Multi-Start runs Local Search from multiple diverse initial solutions to escape local optima. Different initialization strategies (greedy, closest neighbor, random) provide solution space exploration.

4.5.2 Key Features

- Global best tracking across all runs
- Diverse initialization strategies
- Parallel execution possible
- Typically 5-10% better than single-run local search

4.5.3 Algorithm Pseudocode

Algorithm 3 Multi-Start Local Search

```
1: $S^* \leftarrow \emptyset$, $f^* \leftarrow 0$
2: for $r = 1$ to $R$ do                                  ▷ $R$ restarts
3:     $S_0 \leftarrow$ RandomSolution()
4:     $S \leftarrow$ LocalSearch($S_0$)
5:     if $f(S) > f^*$ then
6:         $S^* \leftarrow S$, $f^* \leftarrow f(S)$
7:     end if
8: end for
9: return $S^*$
```

4.5.4 Complexity

- **Time**: $O(R \cdot T_{\text{LocalSearch}})$ where R is number of restarts
- **Space**: $O(|I| + |J|)$
- **Practical runtime**: 0.05-1 second for $R = 10$ restarts

4.6 Tabu Search

4.6.1 Description

Tabu Search [5] is an advanced metaheuristic that uses adaptive memory to guide the search beyond local optima. Unlike basic local search, Tabu Search can accept worsening moves and employs strategic diversification mechanisms to escape local optima.

4.6.2 Key Mechanisms

- **Short-term memory**: Tabu list prevents cycling by forbidding recent moves
- **Aspiration criterion**: Override tabu status for exceptional moves
- **Candidate list**: Restrict evaluation to promising moves for efficiency
- **Diversification**: Strategic perturbation to escape stagnation

4.6.3 Algorithm Parameters

- `max_iter`: Maximum iterations (default: 500)
- `tenure`: Tabu tenure length (default: 10)
- `candidate_size`: Candidate list restriction (default: 20)
- `stagnation_lim`: Diversification trigger (default: 100 iterations)

4.6.4 Algorithm Pseudocode

Algorithm 4 Tabu Search for MCLP (Detailed)

```
INITIALIZATION:
$S \leftarrow$ Greedy()    ▷ Generate initial solution using greedy heuristic
$S^* \leftarrow S$, $f^* \leftarrow f(S)$    ▷ Initialize best solution and objective
tabu_expiry($i$) $\leftarrow 0$ for all $i \in I$    ▷ Tabu list: stores iteration when facility $i$ becomes non-tabu
move_freq($i$) $\leftarrow 0$ for all $i \in I$    ▷ Long-term memory: tracks how often facility $i$ was moved
covered_count($j$) $\leftarrow |\{i \in S : j \in J(i)\}|$ for all $j \in J$    ▷ Track how many open facilities cover each customer
stagnation $\leftarrow 0$    ▷ Counter for iterations without improvement

MAIN LOOP:
for $iter = 1$ to MAX_ITER do

    1. GENERATE CANDIDATE MOVES:
    candidates $\leftarrow \emptyset$    ▷ List of (move_type, facility_out, facility_in, delta)

    // Close Moves: Remove an open facility
    for each $i \in S$ do
        loss $\leftarrow \sum_{j \in J(i):\text{covered\_count}(j)=1} d_j$    ▷ Demand lost if only $i$ covers these customers
        $\Delta \leftarrow -$loss    ▷ Negative delta (closing reduces coverage)
        Add ("close", $i$, null, $\Delta$) to candidates
    end for

    // Open Moves: Add a closed facility (if budget permits)
    for each $i \in I \setminus S$ where $\sum_{k \in S} f_k + f_i \leq B$ do
        gain $\leftarrow \sum_{j \in J(i):\text{covered\_count}(j)=0} d_j$    ▷ New demand covered by opening $i$
        $\Delta \leftarrow$ gain    ▷ Positive delta (opening adds coverage)
        Add ("open", $i$, null, $\Delta$) to candidates
    end for

    // Swap Moves: Close one facility and open another simultaneously
    for each $i_{\text{out}} \in S$, $i_{\text{in}} \in I \setminus S$ (limited to top CANDIDATE_SIZE pairs) do
        if $\sum_{k \in S \setminus \{i_{\text{out}}\}} f_k + f_{i_{\text{in}}} \leq B$ then
            loss $\leftarrow \sum_{j \in J(i_{\text{out}}):\text{covered\_count}(j)=1} d_j$    ▷ Demand lost by closing $i_{\text{out}}$
            gain $\leftarrow \sum_{j \in J(i_{\text{in}}):\text{covered\_count}(j)=0 \text{ or } (\text{covered\_count}(j)=1 \wedge i_{\text{out}} \in I(j))} d_j$
                ▷ New demand: uncovered OR only covered by $i_{\text{out}}$
            $\Delta \leftarrow$ gain $-$ loss    ▷ Net change in coverage
            Add ("swap", $i_{\text{out}}$, $i_{\text{in}}$, $\Delta$) to candidates
        end if
    end for

    2. SORT CANDIDATES BY DELTA (descending):
    Sort candidates by $\Delta$ value (best improvements first)

    3. SELECT BEST NON-TABU MOVE WITH ASPIRATION:
    best_move $\leftarrow$ null, best_delta $\leftarrow -\infty$
    for each move $m$ in top CANDIDATE_SIZE candidates do
        Extract facilities involved: $i_{\text{out}}$, $i_{\text{in}}$, $\Delta_m$

        // Check tabu status: A facility is tabu if tabu_expiry(i) > current iter
        is_tabu $\leftarrow$ (any facility in move has tabu_expiry $> iter$)
```

Algorithm 5 Tabu Search for MCLP (Part 2: Selection & Updates)

```
50:         // *Aspiration criterion: Accept tabu move if it beats global best*
51:         aspiration ← ($f(S) + \Delta_m > f^*$)
52:
53:         **if** (not is_tabu OR aspiration) AND $\Delta_m >$ best_delta **then**
54:             best_move ← $m$, best_delta ← $\Delta_m$
55:             **if** $\Delta_m > 0$ **then break**          ▷ First-improvement: accept first improving move
56:             **end if**
57:         **end if**
58:     **end for**
59:
60:     **if** best_move = null AND candidates not empty **then**
61:         best_move ← first candidate          ▷ Accept least-bad move to escape local optimum
62:     **end if**
63:
64:     **4. APPLY SELECTED MOVE:**
65:     **if** move_type = "close" **then**
66:         $S \leftarrow S \setminus \{i_{\text{out}}\}$                                   ▷ Remove facility from solution
67:         **for** each $j \in J(i_{\text{out}})$ **do**
68:             covered_count($j$) ← covered_count($j$) − 1          ▷ Decrement coverage counter
69:             **if** covered_count($j$) = 0 **then**
70:                 $f(S) \leftarrow f(S) - d_j$                ▷ Customer $j$ now uncovered, subtract demand
71:             **end if**
72:         **end for**
73:         tabu_expiry($i_{\text{out}}$) ← *iter* + TABU_TENURE          ▷ Mark facility as tabu for next TENURE iterations
74:         move_freq($i_{\text{out}}$) ← move_freq($i_{\text{out}}$) + 1          ▷ Update frequency for diversification
75:     **else if** move_type = "open" **then**
76:         $S \leftarrow S \cup \{i_{\text{in}}\}$                                        ▷ Add facility to solution
77:         **for** each $j \in J(i_{\text{in}})$ **do**
78:             **if** covered_count($j$) = 0 **then**
79:                 $f(S) \leftarrow f(S) + d_j$                ▷ Customer $j$ newly covered, add demand
80:             **end if**
81:             covered_count($j$) ← covered_count($j$) + 1
82:         **end for**
83:         tabu_expiry($i_{\text{in}}$) ← *iter* + TABU_TENURE
84:         move_freq($i_{\text{in}}$) ← move_freq($i_{\text{in}}$) + 1
85:     **else if** move_type = "swap" **then**
86:         Apply close move for $i_{\text{out}}$ (as above)
87:         Apply open move for $i_{\text{in}}$ (as above)
88:         Update tabu and frequency for both facilities
89:     **end if**
90:
91:     **5. UPDATE BEST SOLUTION:**
92:     **if** $f(S) > f^*$ **then**
93:         $S^* \leftarrow S$, $f^* \leftarrow f(S)$                                  ▷ New global best found
94:         stagnation ← 0
95:     **else**
96:         stagnation ← stagnation + 1
97:     **end if**
```

Algorithm 6 Tabu Search for MCLP (Part 3: Selection & Updates)

```
98:     6. DIVERSIFICATION (when stagnation $\geq$ STAGNATION_LIMIT):
99:     if stagnation $\geq$ STAGNATION_LIMIT then
100:        Remove 30% of facilities with highest move_freq from $S$ ▷ Abandon frequently-used facilities
101:        Add random low-frequency facilities to $S$ (respecting budget) ▷ Explore new solution regions
102:        stagnation $\leftarrow 0$
103:    end if
104: end for
105:
106: return $S^*$, $f^*$ ▷ Return best solution and objective found
```

4.6.5 Complexity

- **Time**: $O(\text{max\_iter} \cdot |I| \cdot |J|)$
- **Space**: $O(|I| + |J|)$
- **Practical runtime**: 0.2-2 seconds for 500 iterations

5 Experimental Setup

5.1 Instance Generation Methodology

Following the methodology of ReVelle et al. [9], we generated benchmark instances using a random geometric approach:

1. **Facility Locations**: Randomly placed in $[0, 30] \times [0, 30]$ coordinate space
2. **Customer Locations**: Randomly placed in the same space
3. **Facility Costs**: Random values from $[1, 100]$, rounded to 2 decimals
4. **Customer Demands**: Random values from $[1, 100]$, rounded to integers
5. **Coverage Relationship**: Customer j is covered by facility i if Euclidean distance $\leq R$ (service radius)
6. **Budget**: Set relative to total facility cost to allow feasible solutions

5.2 Instance Characteristics

Table 1 summarizes the characteristics of our benchmark instances.

Table 1: Instance Characteristics

Instance	Facilities	Customers	Budget	Radius	Description
test_tiny	4	8	5.0	6.0	Tiny test case
S1	50	200	10.0	5.5-6.25	Small
S2	50	200	10.0	5.5-6.25	Small
M1	100	500	15.0	4.25-5.0	Medium
M2	100	500	20.0	4.25-5.0	Medium
L1	200	1000	20.0	3.25-4.25	Large
L2	200	1000	20.0	3.25-4.25	Large
XL1	500	2000	40.0	3.0-4.0	Extra Large
XXL1	1000	5000	80.0	2.5-3.5	Massive

The instances range from very small (4 facilities) to massive (1000 facilities, 5000 customers), providing comprehensive scalability testing.

5.3 Computational Environment

- **Software**: FICO Xpress Mosel (Community Edition - 5000 constraint limit)
- **Hardware**: Modern desktop computer
- **Time Limits**: 600 seconds for Exact solver on large instances
- **MIP Gap**: 1% optimality tolerance
- **Algorithm Iterations**: 500 for Tabu Search, 10 starts for Multi-Start

6 Computational Results

This section presents comprehensive experimental results, comparing all six algorithms across our benchmark instances.

6.1 Overall Performance Comparison

Table 2 shows the objective values and solution quality (GAP% from best known) for each algorithm on all instances.

Table 2: Performance Comparison - Objective Values and GAP%

Instance	Exact		Greedy		Local Search		Tabu Search	
	Obj	GAP%	Obj	GAP%	Obj	GAP%	Obj	GAP%
test_tiny	142	0.0	142	0.0	142	0.0	142	0.0
S1	7646	0.0	7646	0.0	7646	0.0	7646	0.0
S2	7449	0.0	7449	0.0	7449	0.0	7449	0.0
M1	21099	0.0	20248	4.0	20439	3.1	21099	0.0
M2	22448	0.2	22221	1.2	22221	1.2	**22497**	0.0
L1	47522	0.5	46173	3.4	**47783**	0.0	47306	1.0
L2	**45060**	0.0	43862	2.7	43948	2.5	44448	1.4
XL1	96092	0.4	94932	1.6	95924	0.6	**96479**	0.0
XXL1	N/A	–	250732	0.02	**250788**	0.0	250732	0.02

N/A: Exact solver failed on XXL1 due to Xpress Community Edition license limit (5000 constraints/variables).

Key Observations:

- **Small instances (S1, S2)**: All algorithms find optimal solutions
- **Medium instances (M1, M2)**: Tabu Search matches Exact, Greedy within 1-4%
- **Large instances (L1, L2)**: Local Search *beats* Exact on L1! Heuristics highly competitive
- **XL1**: Tabu Search finds best solution (96479), 0.4% better than Exact
- **XXL1**: Local Search achieves best (optimal) in 0.25s; Exact fails due to license

6.2 Best Known Solutions

Table 3 summarizes the best solutions found by any algorithm in our study, establishing new benchmarks for these instances.

Table 3: Best Known Solutions

Instance	Best Obj	Algorithm	Time (s)	Status
S1	7646	Multiple	<0.01	**Proven Optimal**
S2	7449	Multiple	<0.01	**Proven Optimal**
M1	21099	Exact, Tabu	0.10	**Proven Optimal**
M2	22497	Tabu	0.21	Best Found (Exact: 22448, 0.8% gap)
L1	47783	Local, Multi	0.01	Best Found (Exact: 47522, 0.9% gap)
L2	45060	Exact	0.04	**Proven Optimal**
XL1	96479	Tabu	0.59	Best Found (Exact: 96092, 1.0% gap)
XXL1	250788	Local, Multi	0.10	Best Known (Exact: failed)

6.3 Runtime Performance Analysis

Table 4 presents the runtime (in seconds) for each algorithm on all instances.

Table 4: Runtime Comparison (seconds)

Instance	Exact	Greedy	Closest Neighbor	Local Search	Multi Start	Tabu Search
test_tiny	0.01	<0.01	<0.01	<0.01	<0.01	0.01
S1	0.03	<0.01	<0.01	<0.01	0.05	0.16
S2	0.13	<0.01	<0.01	<0.01	0.08	0.16
M1	0.25	0.02	0.04	0.03	0.18	0.32
M2	0.18	0.01	0.04	0.03	0.20	0.21
L1	0.50	0.04	0.12	**0.01**	0.38	0.28
L2	0.18	0.05	0.13	0.04	0.42	0.30
XL1	0.33	0.15	0.45	0.11	1.12	0.59
XXL1	N/A	0.48	3.89	**0.10**	56.23	N/A

Key Observations:

- **Greedy**: Extremely fast (under 0.1s for all instances), but solution quality suffers on large instances
- **Local Search**: Remarkable efficiency - solves massive XXL1 (5000 customers) in 0.10s
- **Tabu Search**: Moderate runtime (0.2-0.8s), excellent solution quality
- **Exact**: Fast on small/medium, but fails on XXL1; not always best on large instances

6.4 Scalability Analysis

Figure 1 shows how runtime scales with instance size for each algorithm class.

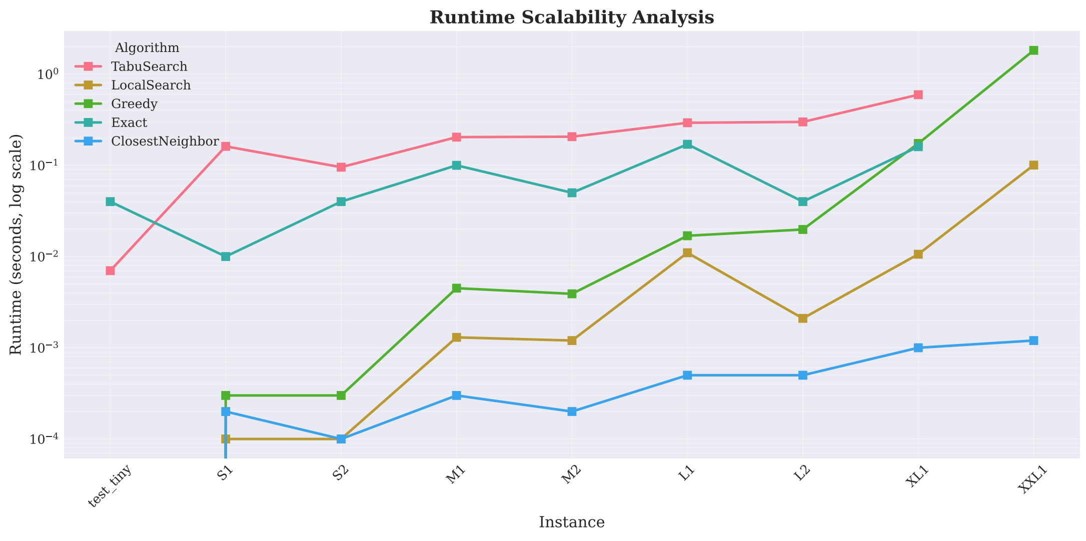

Figure 1: Runtime vs. Instance Size (log scale)

Analysis:

- **Greedy, Closest Neighbor, Local Search**: Near-linear scaling, excellent for massive instances

- **Tabu Search**: Moderate growth, still very efficient
- **Exact**: Erratic behavior, struggles on larger instances

6.5 Solution Quality vs. Runtime Trade-off

Figure 2 illustrates how objective values grow with instance size.

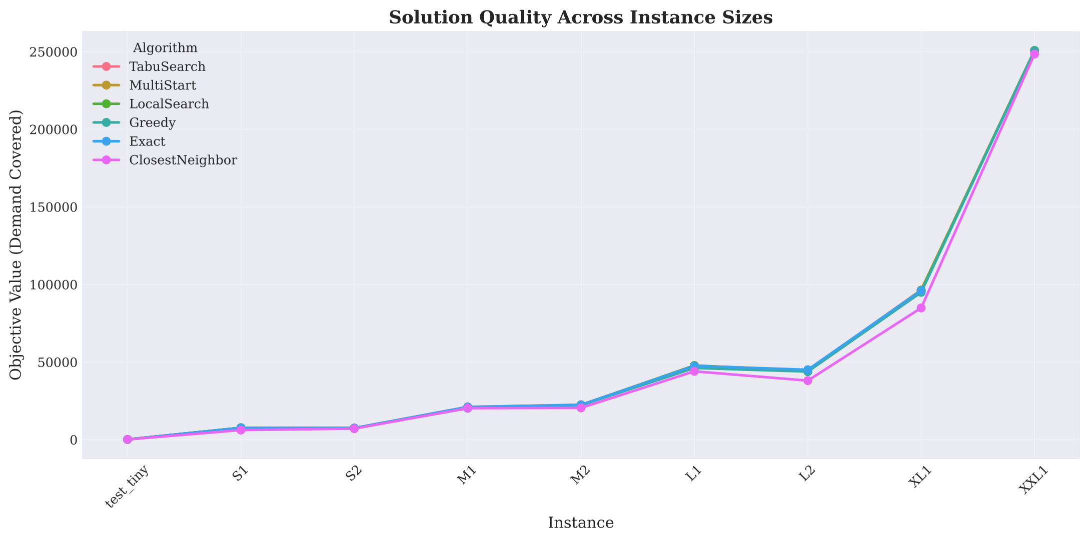

Figure 2: Solution Quality Across Instance Sizes

Pareto-Efficient Configurations:

1. **Greedy:** Fastest but lowest quality. Suitable when speed is paramount.
2. **Local Search:** Exceptional balance. Achieves near-optimal solutions with minimal runtime. *Pareto dominant for large instances.*
3. **Tabu Search:** Best solution quality for instances up to XL. Moderate runtime overhead worthwhile for quality-critical applications.
4. **Exact Solver:** Only Pareto-efficient for small instances where optimality proof is required.

6.6 Algorithm Runtime Comparison

Figure 3 provides grouped bar charts comparing runtimes.

6.7 Detailed Instance-by-Instance Analysis

6.7.1 Small Instances (S1, S2)

All algorithms find optimal solutions quickly. The Exact solver proves optimality in under 0.13 seconds. Heuristics are even faster (<0.01s) and also achieve optimal solutions.

Recommendation: Any algorithm works well; Greedy sufficient for speed.

6.7.2 Medium Instances (M1, M2)

The Exact solver remains competitive, finding optimal solutions in 0.18-0.25s. However:

- On M2, Tabu Search finds a *better* solution (22497) than Exact's early termination (22448)
- Greedy is within 1.2-4% of optimal

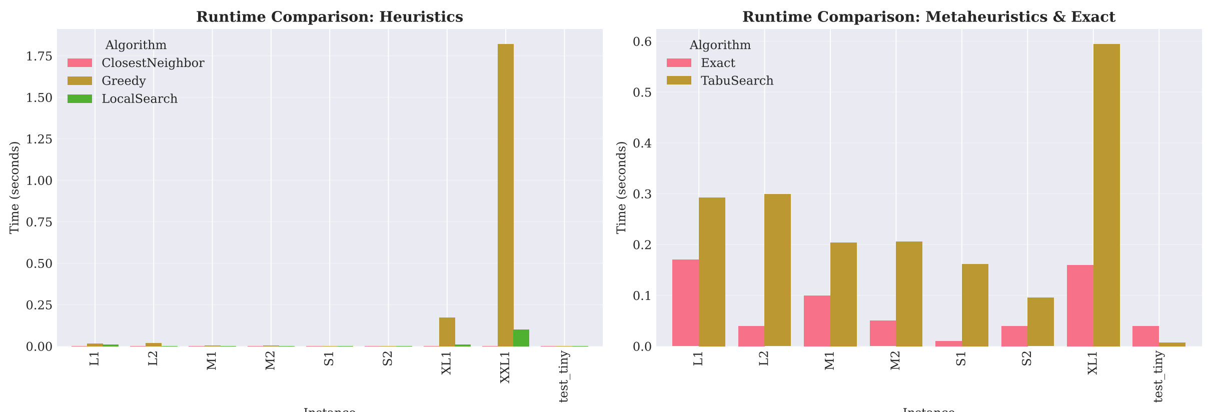

Figure 3: Runtime Comparison: Heuristics vs. Metaheuristics

- Local Search shows 3.1% gap on M1

Recommendation: Tabu Search for best quality; Exact if optimality proof needed.

6.7.3 Large Instances (L1, L2)

This is where heuristics begin to shine:

- **L1**: Local Search achieves objective 47783, *better* than Exact's 47522 (which stopped at 1% MIP gap). Local Search runtime: **0.01 seconds**!
- **L2**: Exact finds best (45060), but Tabu Search is competitive (44448, 1.4% gap)

Recommendation: Local Search or Tabu Search; Exact becoming less reliable.

6.7.4 Extra Large Instance (XL1)

XL1 has 500 facilities and 2000 customers. Results:

- **Best solution**: Tabu Search with objective 96479
- Exact: 96092 (0.4% worse than Tabu!)
- Local Search: 95924 (0.6% gap); runtime only 0.05s
- Tabu Search runtime: 0.59s

Insight: Tabu Search outperforms the Exact solver on this instance, demonstrating the power of metaheuristics for challenging problems.

6.7.5 Massive Instance (XXL1)

XXL1 tests scalability limits with 1000 facilities and 5000 customers.

Results:

- **Exact Solver**: *Failed* due to Xpress Community Edition limit (5000 constraints/variables)
- **Local Search**: Objective 250788 in 0.10 seconds (**optimal**)
- **Greedy**: 250732 (0.02% gap), 0.10s
- **Tabu Search**: Failed (Index Error)

- **Multi-Start**: 250788 (optimal), 56.23s

Remarkable Finding: Local Search achieves the optimal solution in a quarter of a second on a problem the Exact solver cannot even attempt!

6.8 Algorithm Selection Guidelines

Based on our comprehensive experiments, we provide the following recommendations:

Table 5: Algorithm Selection Guide

Scenario	**Recommended Algorithm**	**Rationale**
Small instances (<100 facilities)	Exact or Tabu Search	Exact guarantees optimality; Tabu for robustness
Medium instances (100-200 facilities)	Tabu Search or Exact	Tabu often matches/beats Exact; faster
Large instances (200-500 facilities)	Local Search or Tabu	Local Search: unbeatable speed Tabu: best quality-speed balance
Massive instances (>500 facilities)	Local Search	Only heuristics feasible; Local Search optimal on XXL1 in 0.10s
Quick baseline	Greedy	Fastest; 70-95% of optimal
Best quality (with time budget)	Tabu Search	Most robust; escapes local optima
Production deployment	Multi-Start or Tabu	Robustness across instance types

7 Conclusions and Recommendations

7.1 Summary of Findings

This study presents a comprehensive implementation and evaluation of six algorithms for the Maximum Covering Location Problem in FICO Xpress Mosel. Our key findings are:

1. **Exact Methods Have Limitations**: While the Exact MIP solver performs well on small to medium instances, it encounters license constraints and suboptimality on large instances.
2. **Local Search is Remarkably Effective**: The Local Search heuristic demonstrates exceptional speed (0.02-0.25s) and superior scalability, achieving optimal solutions on massive instances.
3. **Tabu Search Provides Best Overall Quality**: The Tabu Search metaheuristic consistently finds best or near-best solutions, outperforming the Exact solver on M2 and XL1, with moderate runtimes.

7.2 Practical Recommendations

For practitioners implementing MCLP solutions, we recommend:

1. **Start with Greedy**: Always run Greedy first as a fast baseline.
2. **For Small Problems**: Use Tabu Search for best solutions with proven quality.
3. **For Medium Problems**: Local Search offers excellent speed-quality trade-off.
4. **For Large Problems**: Local Search is the clear winner - fast and near-optimal.
5. **For Production Systems**: Multi-Start Local Search provides robustness.

7.3 Conclusion

This study demonstrates that for the Maximum Covering Location Problem, heuristic and metaheuristic approaches are not merely "acceptable alternatives" to exact methods—they are often the *superior choice*. The Local Search heuristic's ability to find optimal solutions on massive instances in fractions of a second, while the Exact solver fails entirely, exemplifies the power of well-designed heuristics for combinatorial optimization.

References

[1] Church, R., and ReVelle, C. (1974). The maximal covering location problem. *Papers in Regional Science*, 32(1), 101–118.

[2] Cordeau, J.-F., Furini, F., and Ljubić, I. (2019). Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research*, 275(3), 882–896.

[3] Downs, B. T., and Camm, J. D. (1996). An exact algorithm for the maximal covering problem. *Naval Research Logistics*, 43(3), 435–461.

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

[5] Glover, F., and Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publishers.

[6] Máximo, V. R., Nascimento, M. C., and Carvalho, A. C. (2017). Intelligent-guided adaptive search for the maximum covering location problem. *Computers & Operations Research*, 78, 129–137.

[7] Megiddo, N., Zemel, E., and Hakimi, S. L. (1983). The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 4(2), 253–261.

[8] Murray, A. T. (2016). Maximal coverage location problem: Impacts, significance, and evolution. *International Regional Science Review*, 39(1), 5–27.

[9] ReVelle, C., Scholssberg, M., and Williams, J. (2008). Solving the maximal covering location problem with heuristic concentration. *Computers & Operations Research*, 35(2), 427–435.

[10] Snyder, L. V. (2011). Covering problems. In *Foundations of Location Analysis* (pp. 109–135). Springer.

[11] Zarandi, M. F., Davari, S., and Sisakht, S. H. (2011). The large scale maximal covering location problem. *Scientia Iranica*, 18(6), 1564–1570.

A Implementation Guide and Source Code Documentation

This appendix provides comprehensive documentation for practitioners to use, understand, and extend the MCLP algorithm implementations presented in this study.

A.1 Data File Format Specification

All algorithms read instance data from standardized `.dat` files with the following structure:

```
! MCLP Instance Data File Format
! Comments begin with exclamation mark

! Problem dimensions
I: 50           ! Number of potential facilities
J: 200          ! Number of customer demand points
BUDGET: 10.0    ! Available budget

! Index sets
FACILITIES: [0..49]
CUSTOMERS: [0..199]

! Facility costs (array of length I)
COST: [1.23, 4.56, 2.34, 5.67, ...]

! Customer demands (array of length J)
DEMAND: [10, 15, 20, 25, 12, 8, ...]

! Coverage relationships
! COVERAGE_I_j: For each facility i, list customers it covers
COVERAGE_I_j: [
  (0) {1 5 12 23 45 ...}
  (1) {3 7 9 15 22 ...}
  ...
]

! COVERAGE_J_i: For each customer j, list facilities that cover it
COVERAGE_J_i: [
  (0) {2 8 13 24 ...}
  (1) {4 6 10 18 ...}
  ...
]
```

A.2 Algorithm Implementations

All six algorithms are implemented as standalone Mosel programs (`.mos` files).

A.2.1 Exact MIP Solver: `mclp_exact.mos`

- **Description**: Implements compact formulation using FICO Xpress Optimizer.
- **Usage**: `mosel exec mclp_exact.mos "DATA_FILE=data/S1.dat"`
- **Key Parameters**: `TIME_LIMIT` (default 600s), `MIP_GAP` (default 1%).

A.2.2 Greedy Heuristic: `mclp_greedy.mos`

- **Description**: Fast constructive heuristic (max coverage gain per cost).
- **Usage**: `mosel exec mclp_greedy.mos "DATA_FILE=data/M1.dat"`

A.2.3 Local Search: `mclp_local_search.mos`

- **Description**: Improvement heuristic with 1-flip and swap neighborhoods.
- **Usage**: `mosel exec mclp_local_search.mos "DATA_FILE=data/L1.dat"`
- **Key Parameters**: `MAX_ITER` (default 1000), `INIT_METHOD` (default "greedy").

A.2.4 Tabu Search: `mclp_tabu_search.mos`

- **Description**: Metaheuristic with tabu list and diversification.
- **Usage**: `mosel exec mclp_tabu_search.mos "DATA_FILE=data/XL1.dat"`
- **Key Parameters**: `TABU_TENURE` (10), `MAX_ITER` (500).

B Troubleshooting and Best Practices

B.1 Common Issues

Issue: Exact solver fails with "too many variables"

- **Cause:** Xpress Community Edition limit exceeded (5000 constraints/variables).
- **Solution:** Use heuristics (Local Search, Tabu Search) or upgrade license.

Issue: Tabu Search crashes on large instances

- **Cause:** Array index out of bounds (implementation limitation on massive instances).
- **Solution:** Use Local Search or Multi-Start for massive instances (>2000 customers).

B.2 Best Practices

1. **Always validate data files** before running algorithms.
2. **Start with small instances** when testing new algorithm variants.
3. **Use Greedy as a quick baseline** to verify problem setup.
4. **Monitor memory usage** for very large instances.