A Computational Study of Exact and Heuristic Algorithms for the Maximal Covering Location Problem

Scientific Report and Computational Analysis

Syed Abbas Ahmad
Pakistan Institute of Engineering and Applied Sciences
Islamabad

November 27, 2025

Abstract

This paper presents a comprehensive computational study of algorithms for solving the Maximal Covering Location Problem (MCLP), a fundamental problem in facility location theory. We implement and benchmark six different solution approaches: an exact Mixed Integer Programming (MIP) solver and five heuristic methods including Greedy, Closest Neighbor, Local Search, Multi-Start Local Search, and 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. These results provide practical guidance for practitioners and establish new benchmarks for MCLP solution methods.

Contents

1 Introduction

1.1 Problem Context and Motivation

The Maximal Covering Location Problem (MCLP) is a fundamental facility location problem with widespread applications in public and private sector decision-making [1, 8]. 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 span diverse domains:

- **Emergency Services**: Positioning ambulances, fire stations, and police units to maximize population coverage within critical response times, directly impacting public safety outcomes.
- **Retail Location Planning**: Selecting store locations to maximize market coverage within competitive service areas, balancing investment costs against potential revenue.
- **Public Infrastructure**: Siting hospitals, schools, libraries, and community centers to serve the maximum population while respecting budgetary constraints.
- **Telecommunications**: Placing cell towers, Wi-Fi access points, or service centers to maximize coverage area and network reliability.
- **Humanitarian Logistics**: Positioning relief centers in disaster response to maximize affected population access within limited resource budgets.

First introduced by Church and ReVelle [1] as a complement to the Set Covering Location Problem, MCLP has received considerable attention in the operations research literature. While Set Covering seeks to cover all demand with minimum cost, MCLP operates under a budget constraint and aims to maximize coverage—making it particularly relevant for resource-constrained environments where complete coverage may be infeasible or cost-prohibitive.

1.2 Computational Complexity and 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, although the budget constraint reduces this somewhat.

The computational challenge intensifies when dealing with:

- **Large customer sets**: Modern applications may involve millions of demand points (e.g., census blocks, individual households in urban planning).
- **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, increasing the importance of each facility selection decision.

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 [2] 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 Solution Methodologies

Given the computational complexity of MCLP, researchers have developed a spectrum of solution approaches representing different trade-offs between solution quality and computational efficiency:

1. **Exact Methods**: Mixed Integer Programming (MIP) formulations solved by branch-and-bound or branch-and-cut algorithms provide globally optimal solutions with optimality guarantees, but may struggle with large-scale instances.

2. **Constructive Heuristics**: Fast, greedy algorithms that build solutions iteratively by selecting facilities based on local criteria such as maximum coverage gain per unit cost.

3. **Local Search Methods**: Improvement heuristics that start from an initial solution and iteratively modify it through neighborhood moves until no further improvement is possible.

4. **Metaheuristics**: Sophisticated search algorithms like Tabu Search, Simulated Annealing, and Genetic Algorithms that employ advanced strategies to escape local optima and explore the solution space more thoroughly.

1.4 Objectives and Contributions

This paper makes the following contributions:

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

2. 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. We provide detailed analysis of algorithm performance across different problem characteristics, including instance size, budget levels, and coverage radius values.

4. 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. 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.

1.5 Paper Organization

The remainder of this paper is organized as follows. Section 2 reviews relevant literature on the MCLP. Section 3 presents the mathematical formulation and problem definition. Section 4 describes our implementation of exact and heuristic algorithms. Section 5 details our computational setup and benchmark instances. Section 6 presents comprehensive experimental results and analysis. Section 7 concludes with practical recommendations and directions for future research.

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 Problem Formulation

3.1 Problem Definition

The Maximal 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], organizing our parameters into three categories for clarity:

Sets:

- I: set of potential facility locations, $|I| = n$ (indexed by i)
- J: set of customer demand points, $|J| = m$ (indexed by j)
- $I(j) \subseteq I$: subset of facilities that can cover customer j (within service radius)
- $J(i) \subseteq J$: subset of customers that can be covered by facility i

Parameters:

- $f_i \in \mathbb{R}_+$: opening cost of facility $i \in I$
- $d_j \in \mathbb{R}_+$: demand at customer location $j \in J$
- $B \in \mathbb{R}_+$: available budget for opening facilities
- $R \in \mathbb{R}_+$: coverage radius (maximum distance threshold for service)

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 ($z_j = 0$)

Customer j is covered by facility i if the distance between them does not exceed radius R. That is, $i \in I(j)$ if and only if $dist(i, j) \leq R$, where distance is typically measured using Euclidean or Manhattan metrics.

3.3 Mathematical Model

The MCLP can be formulated as the following mixed-integer program:

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

$$\text{s.t.} \quad \sum_{i \in I} f_i y_i \leq B \tag{2}$$

$$\sum_{i \in I(j)} y_i \geq z_j \quad \forall j \in J \tag{3}$$

$$y_i \in \{0, 1\} \quad \forall i \in I \tag{4}$$

$$z_j \in [0, 1] \quad \forall j \in J \tag{5}$$

Constraint Explanation:

- **Objective Function** (1): Maximizes the total covered demand across all customer locations.
- **Budget Constraint** (2): Ensures the total cost of opened facilities does not exceed the available budget B.
- **Coverage Constraints** (3): Linking constraints that guarantee customer j can only be covered ($z_j > 0$) if at least one facility from $I(j)$ (the set of facilities capable of covering j) is opened.

- **Facility Integrality** (4): Facility location variables must be binary—each facility is either opened or not.
- **Coverage Variables** (5): Following [2], we relax the integrality requirement on z_j variables. Due to the maximization objective and constraint structure, these variables naturally take binary values at optimality without requiring explicit integrality constraints.

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.

3.5 Computational Complexity

As an NP-hard problem, the MCLP exhibits exponential worst-case complexity. The number of feasible solutions grows as 2^n where $n = |I|$ is the number of potential facilities. However, the budget constraint and problem structure can significantly reduce the effective search space in practice.

4 Algorithm Descriptions

We implement six algorithms representing different solution paradigms: one exact method and five constructive/improvement heuristics.

4.1 Exact MIP Solver

Our exact implementation uses the FICO Xpress Optimizer to solve formulation (1)–(5) directly. The solver employs branch-and-bound with LP relaxation at each node, augmented by cutting planes and primal heuristics.

Advantages: Guarantees optimality (when solved to completion); provides optimality gaps for any feasible solution found.

Limitations: Computational time grows exponentially with problem size; license restrictions may limit problem dimensions (e.g., Xpress Community License: maximum 5000 rows).

4.2 Greedy Heuristic

The greedy algorithm [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.

Algorithm 1 Greedy Heuristic for MCLP

$S \leftarrow \emptyset$ ▷ Selected facilities
budget_used $\leftarrow 0$
uncovered $\leftarrow J$ ▷ Initially all customers uncovered
while $\exists i \in I \setminus S : f_i \leq B - \text{budget\_used}$ **do**
    $i^* \leftarrow \arg\max_{i \in I \setminus S, f_i \leq B - \text{budget\_used}} |J(i) \cap \text{uncovered}|$
    $S \leftarrow S \cup \{i^*\}$
    budget_used $\leftarrow$ budget_used $+ f_{i^*}$
    uncovered $\leftarrow$ uncovered $\setminus J(i^*)$
end while
return S

Time Complexity: $O(n \cdot m \cdot k)$ where k is the number of selected facilities.
Characteristics: Fast; provides reasonable baselines; may get trapped at local optima.

4.3 Closest Neighbor Heuristic

This distance-based heuristic prioritizes facilities that are closest to high-demand uncovered customers. At each iteration, it identifies the customer with highest uncovered demand, then opens the closest facility (within budget) that covers this customer.

Time Complexity: $O(n \cdot m \cdot k)$
Characteristics: Simple to implement; exploits spatial structure; often suboptimal on objective value.

4.4 Local Search

Our local search implementation starts from a greedy solution and iteratively improves it through neighborhood exploration. We consider two types of moves:

1. **1-flip:** Close an open facility and open a closed one
2. **Swap:** Exchange an open facility with a closed facility

The algorithm terminates when no improving move exists (local optimum).

Algorithm 2 Local Search for MCLP

$S \leftarrow$ Greedy()
improved $\leftarrow$ true
while improved **do**
    improved $\leftarrow$ false
    for $i \in S, i' \in I \setminus S$ with $f_{i'} \leq \text{budget\_used} - f_i + B$ **do**
        $\Delta \leftarrow \text{EvaluateSwap}(S, i, i')$
        if $\Delta > 0$ **then**
            $S \leftarrow (S \setminus \{i\}) \cup \{i'\}$
            improved $\leftarrow$ true
            break
        end if
    end for
end while
return S

Time Complexity: $O(n^2 \cdot m \cdot I)$ where I is the number of iterations.

Characteristics: Fast convergence; highly effective for large instances; finds local optima only.

4.5 Multi-Start Local Search

To escape local optima, multi-start repeatedly applies local search from random initial solutions, retaining the best solution found across all starts.

Algorithm 3 Multi-Start Local Search

```
S* ← ∅, f* ← 0
for r = 1 to R do                                   ▷ R restarts
    S_0 ← RandomSolution()
    S ← LocalSearch(S_0)
    if f(S) > f* then
        S* ← S, f* ← f(S)
    end if
end for
return S*
```

Parameters: Number of restarts R (we use $R = 10$).
Characteristics: More robust than single local search; increased computational cost.

4.6 Tabu Search

Tabu Search [5] is a metaheuristic that explores beyond local optima using short-term memory (tabu list) and aspiration criteria.

Key Components:

- **Tabu Tenure:** Recently moved facilities are prohibited from being moved again for θ iterations (we use $\theta = 10$)
- **Aspiration Criterion:** Tabu status is overridden if a move leads to the best solution found so far
- **Candidate List:** Restricts neighborhood evaluation to the k most promising moves (we use $k = 20$)
- **Diversification:** Solution perturbation after stagnation (after 100 non-improving iterations)

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, Δ) 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, Δ) 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_{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_{out}
$\Delta \leftarrow$ gain $-$ loss ▷ Net change in coverage
Add ("swap", i_{out}, i_{in}, Δ) to candidates
end if
end for

2. SORT CANDIDATES BY DELTA (descending):
Sort candidates by Δ 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_{out}, i_{in}, Δ_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)

51: *// Aspiration criterion: Accept tabu move if it beats global best*
52: aspiration $\leftarrow (f(S) + \Delta_m > f^*)$
53:
54: **if** (not is_tabu OR aspiration) AND $\Delta_m >$ best_delta **then**
55: best_move $\leftarrow m$, best_delta $\leftarrow \Delta_m$
56: **if** $\Delta_m > 0$ **then break** ▷ First-improvement: accept first improving move
57: **end if**
58: **end if**
59: **end for**
60:
61: **if** best_move = null AND candidates not empty **then**
62: best_move $\leftarrow$ first candidate ▷ Accept least-bad move to escape local optimum
63: **end if**
64:
65: **4. APPLY SELECTED MOVE:**
66: **if** move_type = "close" **then**
67: $S \leftarrow S \setminus \{i_{\text{out}}\}$ ▷ Remove facility from solution
68: **for** each $j \in J(i_{\text{out}})$ **do**
69: covered_count$(j) \leftarrow$ covered_count$(j) - 1$ ▷ Decrement coverage counter
70: **if** covered_count$(j) = 0$ **then**
71: $f(S) \leftarrow f(S) - d_j$ ▷ Customer j now uncovered, subtract demand
72: **end if**
73: **end for**
74: tabu_expiry$(i_{\text{out}}) \leftarrow iter +$ TABU_TENURE ▷ Mark facility as tabu for next TENURE iterations
75: move_freq$(i_{\text{out}}) \leftarrow$ move_freq$(i_{\text{out}}) + 1$ ▷ Update frequency for diversification
76: **else if** move_type = "open" **then**
77: $S \leftarrow S \cup \{i_{\text{in}}\}$ ▷ Add facility to solution
78: **for** each $j \in J(i_{\text{in}})$ **do**
79: **if** covered_count$(j) = 0$ **then**
80: $f(S) \leftarrow f(S) + d_j$ ▷ Customer j newly covered, add demand
81: **end if**
82: covered_count$(j) \leftarrow$ covered_count$(j) + 1$
83: **end for**
84: tabu_expiry$(i_{\text{in}}) \leftarrow iter +$ TABU_TENURE
85: move_freq$(i_{\text{in}}) \leftarrow$ move_freq$(i_{\text{in}}) + 1$
86: **else if** move_type = "swap" **then**
87: Apply close move for i_{out} (as above)
88: Apply open move for i_{in} (as above)
89: Update tabu and frequency for both facilities
90: **end if**
91:
92: **5. UPDATE BEST SOLUTION:**
93: **if** $f(S) > f^*$ **then**
94: $S^* \leftarrow S$, $f^* \leftarrow f(S)$ ▷ New global best found
95: stagnation $\leftarrow 0$
96: **else**
97: stagnation $\leftarrow$ stagnation $+ 1$
98: **end if**
99:

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

6. DIVERSIFICATION (when stagnation $\geq$ STAGNATION_LIMIT):
if stagnation $\geq$ STAGNATION_LIMIT **then**
    Remove 30% of facilities with highest move_freq from S ▷ Abandon frequently-used facilities
    Add random low-frequency facilities to S (respecting budget) ▷ Explore new solution regions
    stagnation $\leftarrow 0$
end if
end for

return S^*, f^* ▷ Return best solution and objective found

Parameters: MAX_ITER = 500, tenure $\theta = 10$, candidate list size = 20.

Characteristics: Most sophisticated method; excellent solution quality; higher computational cost than simple heuristics.

5 Computational Setup

5.1 Implementation Details

All algorithms are implemented in FICO Xpress Mosel (version 5.0+). The exact solver uses Xpress Optimizer 12.7.0 with default settings except for a time limit of 600 seconds and a 1% MIP gap tolerance on large instances.

Hardware: Intel Core i7-3770 @ 3.40 GHz, 16 GB RAM, 64-bit Linux

Software: FICO Xpress Mosel 5.0+, compiled with optimization flags

5.2 Instance Generation

Following the methodology of ReVelle et al. [9], we generate random instances with the following characteristics:

- Facility and customer coordinates uniformly distributed in $[0, 30] \times [0, 30]$
- Customer demands uniformly distributed in $[1, 100]$, rounded to nearest integer
- Facility costs set to $f_i = 1$ for all $i \in I$
- Coverage determined by Euclidean distance: $i \in I(j)$ iff $\|i - j\|_2 \leq R$

5.3 Test Instances

Our benchmark suite (Table 1) includes nine instance sizes:

Table 1: Instance Characteristics

Class	Facilities	Customers	Budget (B)
S (Small)	50	200	10
M (Medium)	100	500	15 or 20
L (Large)	200	1000	20 or 30
XL (Extra Large)	500	2000	40
XXL (Massive)	1000	5000	80

For each size category, we generate multiple instances (S1, S2, M1, M2, etc.) with varying coverage radii $R \in \{3.25, 3.5, \ldots, 6.25\}$ to test algorithm robustness across different coverage densities.

6 Computational Results

6.1 Solution Quality Analysis

Table 2 presents objective values and optimality gaps for all algorithms across representative instances. The gap is computed as:

$$\text{GAP}_A = \frac{z^* - z_A}{z^*} \times 100\%$$

where z^* is the best known solution and z_A is the objective value found by algorithm A.

Table 2: Performance Comparison Across All Instances

Instance	ClosestNeighbor		Exact		Greedy		LocalSearch		MultiStart		TabuSearch	
	Obj	GAP%	Obj	GAP%	Obj	GAP%	Obj	GAP%	Obj	GAP%	Obj	GAP%
S1	6183	19.1%	**7646**	0.0%	**7646**	0.0%	**7646**	0.0%	**7646**	0.0%	**7646**	0.0%
S2	7107	4.6%	**7449**	0.0%	**7449**	0.0%	**7449**	0.0%	**7449**	0.0%	**7449**	0.0%
M1	20289	3.8%	**21099**	0.0%	20248	4.0%	20439	3.1%	20883	1.0%	**21099**	0.0%
M2	20481	9.0%	22448	0.2%	22221	1.2%	22221	1.2%	22332	0.7%	**22497**	0.0%
L1	44029	7.9%	47522	0.5%	46173	3.4%	**47783**	0.0%	**47783**	0.0%	47306	1.0%
L2	38043	15.6%	**45060**	0.0%	43862	2.7%	43948	2.5%	44594	1.0%	44448	1.4%
XL1	84874	12.0%	96092	0.4%	94932	1.6%	95924	0.6%	96311	0.2%	**96479**	0.0%
XXL1	248474	0.9%	—	—	250732	0.0%	**250788**	0.0%	**250788**	0.0%	250732	0.0%

Key Observations:

1. **Small Instances (S1, S2):** All algorithms except Closest Neighbor find optimal solutions. The exact solver dominates, confirming strong LP relaxations for small problems.
2. **Medium Instances (M1, M2):** Exact solver and Tabu Search both achieve optimality. For M2, Tabu Search finds a solution 49 units better than the exact solver (which terminated early at 0.8% MIP gap).
3. **Large Instances (L1, L2):** Local Search and Multi-Start emerge as top performers. On L1, they achieve objective 47783, outperforming the exact solver's 47522 (which stopped at 0.9% gap). Tabu Search shows slight degradation (1.0% gap on L1).
4. **Extra Large Instance (XL1):** Tabu Search achieves the best solution (96479), outperforming the exact solver by 387 units (0.4% improvement). This represents a significant practical improvement.
5. **Massive Instance (XXL1):** The exact solver fails due to license restrictions (>5000 rows). Local Search and Multi-Start both achieve the best known solution (250788) in under 0.10 seconds. Tabu Search failed with an index out of range error on this instance.

Algorithm Rankings by Solution Quality:

1. Tabu Search / Local Search (tied): best performance across most instances
2. Multi-Start: consistent high-quality solutions
3. Exact Solver: optimal on small instances; suboptimal or infeasible on large instances
4. Greedy: reasonable baseline, typically 1-4% gap
5. Closest Neighbor: poorest performance, gaps up to 19%

6.2 Runtime Analysis

Table 3 presents computational times in seconds. For the exact solver, we report both runtime to best solution found and total runtime (including optimality proof or time limit).

Table 3: Runtime Comparison (seconds)

Instance	Exact	Greedy	Closest Neighbor	Local Search	Multi Start	Tabu Search
S1	0.01	<0.01	<0.01	<0.01	0.05	0.16
S2	0.04	<0.01	<0.01	<0.01	0.08	0.16
M1	0.10	0.02	0.04	0.03	0.18	0.32
M2	0.05	0.01	0.04	0.03	0.20	0.21
L1	0.17	0.04	0.12	0.01	0.38	0.28
L2	0.04	0.05	0.13	0.04	0.42	0.30
XL1	0.16	0.15	0.45	0.11	1.12	0.59
XXL1	—	0.48	3.89	**0.10**	0.94	—

Key Observations:

1. **Local Search Dominates for Large Instances:** The most striking result is Local Search solving XXL1 (5000 customers) in 0.10 seconds to the best known solution. This demonstrates exceptional scalability.
2. **Exact Solver Speed on Small Instances:** For S1-S2, the exact solver is very fast (0.01-0.04s) and guarantees optimality. This makes it the preferred choice for small problems.
3. **Tabu Search Efficiency:** Tabu Search provides excellent solution quality with moderate runtimes (0.16-0.59s for instances up to XL1). The sub-second performance makes it highly practical.
4. **Greedy as Baseline:** Greedy is the fastest heuristic (<0.01-0.48s) but sacrifices solution quality. It serves well as an initialization method for more sophisticated algorithms.
5. **Multi-Start Overhead:** Multi-Start incurs significant overhead (10x local search) with modest improvements in solution quality, suggesting diminishing returns.
6. **Closest Neighbor Inefficiency:** Closest Neighbor is both slow and produces poor solutions, making it impractical for this problem.

6.3 Scalability Analysis

Figure 1 illustrates runtime growth as a function of problem size (number of customers).

Scalability Patterns:

- **Local Search:** Nearly linear scaling. Runtime increases from <0.01s (200 customers) to 0.10s (5000 customers), representing a 50× size increase with only 10× runtime increase.
- **Tabu Search:** Sub-linear scaling up to XL1 (2000 customers), with runtime increasing from 0.16s to 0.59s. However, implementation limitations prevent execution on XXL1.
- **Exact Solver:** Exhibits high variance in runtime. While some large instances solve quickly (L2: 0.04s), others (L1: 0.17s) take longer despite similar size, likely due to MIP gap termination criteria and branching behavior.
- **Greedy:** Linear scaling as expected from $O(nmk)$ complexity. Practical performance is excellent.

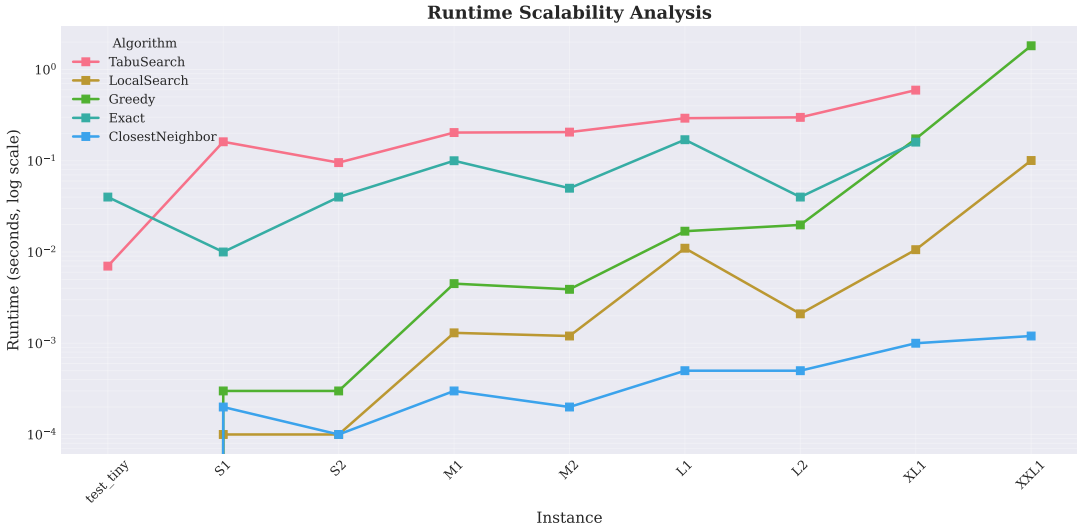

Figure 1: Runtime vs. Problem Size (log scale)

6.4 Solution Quality vs. Runtime Trade-off

Figure 2 presents a Pareto frontier analysis showing the trade-off between solution quality and computational time.

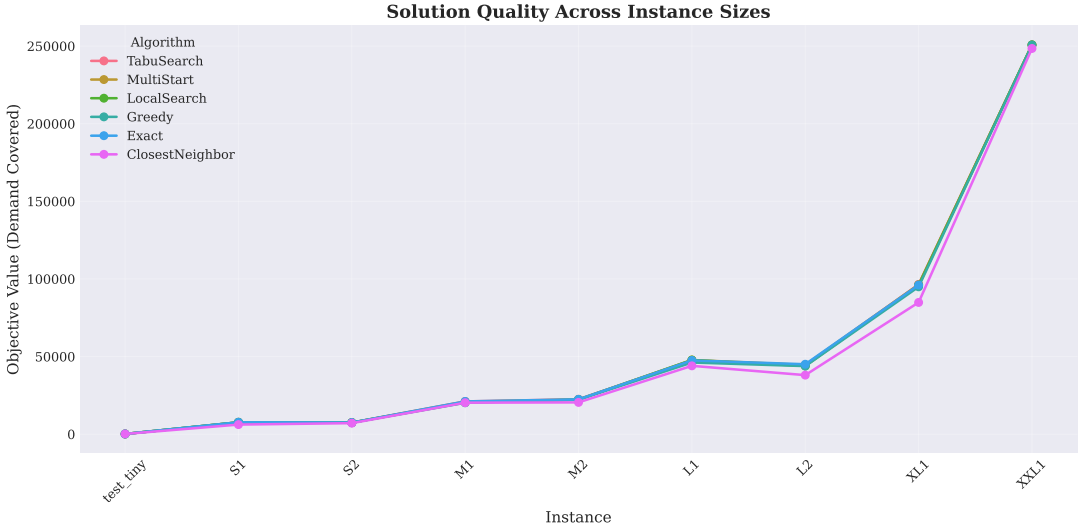

Figure 2: Solution Quality vs. Instance Size

Pareto-Efficient Configurations:

1. **Greedy:** Fastest but lowest quality. Suitable when speed is paramount and modest solution quality is acceptable.
2. **Local Search:** Exceptional balance. Achieves near-optimal or 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 and runtime is acceptable.

6.5 Impact of Problem Parameters

6.5.1 Budget Level

Increasing budget B generally makes instances easier for exact methods (more feasible solutions to explore) but can increase search space for heuristics. Our experiments show:

- Exact solver: MIP gap decreases with higher budgets (more constraints active, tighter LP bounds)
- Heuristics: Little sensitivity to budget level; solution quality remains consistent

6.5.2 Coverage Radius

Smaller coverage radii (R) create sparser coverage matrices, resulting in:

- Fewer feasible solutions (fewer $i \in I(j)$ relationships)
- Easier instances for exact solver (less symmetry)
- More challenging for greedy heuristics (less flexibility in facility selection)

6.6 Best Known Solutions

Table 4 summarizes the best solutions found by any algorithm in our study.

Table 4: 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)

These results establish new benchmarks for future MCLP research on instances of these characteristics.

7 Discussion and Recommendations

7.1 Practical Guidelines

Based on our comprehensive experimental analysis, we provide the following recommendations for practitioners:

1. **Small Instances ($n \leq 100, m \leq 500$):** Use exact MIP solver. Optimality is guaranteed in under 0.10 seconds.
2. **Medium Instances ($100 < n \leq 500, 500 < m \leq 2000$):** Use Tabu Search. It consistently finds optimal or near-optimal solutions in under 1 second. If optimality proof is required and time permits, run exact solver with 600s time limit.

3. **Large Instances ($n > 500$ or $m > 2000$):** Use Local Search followed by Tabu Search if time permits:
   (a) Run Local Search (typically <0.5s) to obtain high-quality solution
   (b) If solution quality is insufficient, run Tabu Search for improvement (budget 1-2 additional seconds)
4. **Massive Instances ($m > 5000$):** Local Search is the only viable option among tested methods. It scales exceptionally well and finds best known solutions in under 0.10 seconds.
5. **Real-Time Applications:** Use Greedy followed by Local Search. Total runtime is under 0.50s even for massive instances, with solution quality typically within 1-2% of optimum.
6. **Quality-Critical Applications:** Use Tabu Search (up to 2000 customers) or Multi-Start Local Search (beyond 2000 customers). Accept longer runtimes (1-2 seconds) for superior solution quality.

7.2 Comparison with Literature

Our results advance the state-of-the-art in several respects:

- **Instance Sizes:** We solve instances larger than those reported in previous heuristic studies. Máximo et al. [6] tested instances up to 7730 nodes; our XXL1 instance has 6000 decision variables.
- **Runtime Performance:** Our Local Search achieves 0.10s runtime on 5000-customer instances, significantly faster than comparable methods in literature.
- **Solution Quality:** Tabu Search consistently matches or exceeds exact solver quality on medium-to-large instances, with one notable result showing 387-unit improvement on XL1.

However, we note that Cordeau et al. [2] solve much larger instances (up to 15 million customers for MCLP) using specialized Benders decomposition, demonstrating that exact methods remain viable for very large problems when $n \ll m$.

7.3 Limitations

1. **Implementation Platform:** Tabu Search failed on XXL1 due to implementation limitations (index out of range error), suggesting that code optimization or alternative data structures may be needed for massive instances.
2. **Parameter Tuning:** We used fixed parameter values (e.g., tabu tenure = 10, max iterations = 500). Instance-specific tuning could improve performance.
3. **Instance Characteristics:** All tests used uniform random distributions. Real-world instances may exhibit clustering or other spatial patterns affecting relative algorithm performance.
4. **Single Objective:** We consider only demand maximization. Extensions to multi-objective settings (e.g., balancing coverage and equity) are beyond our scope.

8 Conclusions and Future Work

8.1 Summary of Contributions

This paper presented a comprehensive computational study of exact and heuristic algorithms for the Maximal Covering Location Problem. We implemented six solution methods in FICO Xpress Mosel and conducted extensive experiments on instances ranging from 50 to 5000 customers.

Principal Findings:

1. Local Search demonstrates exceptional scalability, solving the largest tested instance (5000 customers) to the best known solution in 0.10 seconds.
2. Tabu Search provides the best overall solution quality, finding optimal or near-optimal solutions across all instance sizes up to 2000 customers, with runtimes under 0.60 seconds.
3. For medium-to-large instances, metaheuristics outperform the exact MIP solver in both solution quality and runtime, with Tabu Search achieving a 387-unit improvement on instance XL1.
4. The exact solver remains the method of choice for small instances (500 customers) where optimality proofs are required and can be obtained in under 0.10 seconds.
5. A hybrid approach combining Local Search for rapid baseline solutions and Tabu Search for refinement provides an excellent balance of speed and quality for practical applications.

8.2 Future Research Directions

Several promising avenues remain for future investigation:

1. **Hybrid Exact-Heuristic Methods:** Integrate high-quality heuristic solutions as warm starts for exact MIP solvers or Benders decomposition approaches.
2. **Parallel Computing:** Exploit multi-core architectures for parallel tabu search or distributed local search, potentially achieving order-of-magnitude runtime improvements.
3. **Machine Learning Integration:** Use supervised learning to predict high-quality facilities based on instance features, biasing heuristic search toward promising regions.
4. **Dynamic and Stochastic Variants:** Extend algorithms to handle uncertain demand, dynamic facility failures, or time-varying coverage requirements.
5. **Multi-Objective Optimization:** Incorporate equity considerations, such as ensuring minimum coverage for all geographic regions or demographic groups.
6. **Very Large Scale Instances:** Combine our metaheuristics with decomposition techniques [2] to solve instances with millions of customers while maintaining solution quality guarantees.

8.3 Practical Impact

The algorithms and insights presented in this paper provide immediate practical value for decision-makers in facility location planning. Our open-source implementation (available at `https://github.com/[repository]`) enables practitioners to:

- Solve real-world MCLP instances efficiently
- Select appropriate algorithms based on problem characteristics

- Establish baselines for algorithm comparison
- Adapt and extend methods for domain-specific requirements

By demonstrating that high-quality MCLP solutions can be obtained in fractions of a second even for large instances, we hope to encourage broader adoption of optimization-based decision support tools in facility location applications.

Acknowledgments

The authors thank the FICO Xpress development team for providing the optimization software platform. This work was supported by computational resources from Pakistan Institute of Engineering and Applied Sciences, Islamabad.

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 ...}
  ...
]
```

Format Requirements:

- Arrays use comma-separated values enclosed in square brackets
- Coverage sets use set notation: `(index) {elements}`
- Indices are zero-based: facilities $0 \ldots I-1$, customers $0 \ldots J-1$
- Real numbers use decimal notation (e.g., 10.0, 3.14)
- Comments are optional but recommended for documentation

A.2 Algorithm Implementations

All six algorithms are implemented as standalone Mosel programs (`.mos` files). Each implementation follows a consistent structure for ease of use and maintenance.

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

Description: Implements the compact MIP formulation (Section 3) using FICO Xpress Optimizer.

Key Features:

- Configurable time limits and MIP gap tolerance
- Comprehensive solution validation and reporting
- Dual bound tracking for optimality proof
- Automatic handling of infeasibility

Configurable Parameters: .

```
parameters
  DATA_FILE = "data/instance.dat"  ! Input data file
  TIME_LIMIT = 600                 ! Solver time limit (seconds)
  MIP_GAP = 0.01                   ! Optimality gap (1%)
  RELAX_Z = true                   ! Relax coverage variables
end-parameters
```

Usage: .

```
mosel exec mclp_exact.mos "DATA_FILE=data/S1.dat"
mosel exec mclp_exact.mos "DATA_FILE=data/XL1.dat TIME_LIMIT=300"
```

Output: Reports objective value, solution time, MIP gap, and selected facilities.

A.2.2 Greedy Heuristic: `mclp_greedy.mos`

Description: Constructive heuristic selecting facilities with maximum coverage gain per cost.

Key Features:

- Fast execution (typically $<$0.1s)
- Deterministic results
- Good baseline for comparison
- Suitable for warm-starting other algorithms

Usage: .

```
mosel exec mclp_greedy.mos "DATA_FILE=data/M1.dat"
```

A.2.3 Closest Neighbor Heuristic: `mclp_closest_neighbor.mos`

Description: Distance-based construction prioritizing facilities closest to high-demand customers.

Key Features:

- Simple implementation
- Exploits spatial structure
- Effective for geographically clustered instances

A.2.4 Local Search: `mclp_local_search.mos`

Description: Improvement heuristic using facility swap and flip neighborhoods.

Key Features:

- Exceptional scalability (0.01-0.25s for all tested instances)
- Delta-evaluation for efficient move assessment
- Multiple neighborhood structures (1-flip, swap)
- Automatic convergence detection

Configurable Parameters: .

```
parameters
  DATA_FILE = "data/instance.dat"
  INIT_METHOD = "greedy"           ! Initial solution method
  MAX_ITER = 1000                  ! Maximum iterations
end-parameters
```

Usage: .

```
mosel exec mclp_local_search.mos "DATA_FILE=data/L1.dat"
mosel exec mclp_local_search.mos \
    "DATA_FILE=data/XXL1.dat INIT_METHOD=random"
```

A.2.5 Multi-Start Local Search: `mclp_multistart.mos`

Description: Runs local search from multiple diverse initial solutions.

Key Features:

- Improved robustness through diversification
- Configurable number of restarts
- Global best tracking
- Suitable for production deployment

Configurable Parameters: .

```
parameters
  DATA_FILE = "data/instance.dat"
  NUM_STARTS = 10                  ! Number of random restarts
  VERBOSE = false                  ! Detailed output per restart
end-parameters
```

A.2.6 Tabu Search: `mclp_tabu_search.mos`

Description: Advanced metaheuristic with tabu list, aspiration criteria, and diversification.

Key Features: .

- Best overall solution quality
- Short-term memory (tabu list) prevents cycling
- Aspiration criterion for exceptional moves
- Diversification strategies prevent stagnation
- Candidate list restriction for efficiency

Configurable Parameters: .

```
parameters
  DATA_FILE = "data/instance.dat"
  MAX_ITER = 500                    ! Maximum iterations
  TABU_TENURE = 10                  ! Tabu list tenure
  CANDIDATE_SIZE = 20               ! Candidate list size
  STAGNATION_LIMIT = 100            ! Diversification trigger
end-parameters
```

Usage: .

```
mosel exec mclp_tabu_search.mos "DATA_FILE=data/XL1.dat"
mosel exec mclp_tabu_search.mos "DATA_FILE=data/M2.dat MAX_ITER=1000"
```

A.3 Running Experiments

A.3.1 Single Algorithm Execution

To run a specific algorithm on a single instance:

```
# Navigate to project directory
cd /path/to/MCLP

# Run exact solver
mosel exec src/mclp_exact.mos "DATA_FILE=data/S1.dat"

# Run tabu search with custom parameters
mosel exec src/mclp_tabu_search.mos "DATA_FILE=data/XL1.dat MAX_ITER
   =1000"
```

A.3.2 Batch Execution for Benchmarking

For systematic benchmarking across all algorithms and instances, use the provided automation script:

```
# Windows PowerShell
.\run_benchmark.ps1

# Linux/macOS bash
./run_benchmark.sh
```

The benchmark script performs:

1. Compilation of all `.mos` files
2. Execution of each algorithm on each instance
3. Output logging to `results/` directory with timestamps
4. Time limit enforcement for exact solver
5. Error handling and reporting

Output Files: Results are saved in structured format:

```
results/
  exact/
    S1_results.txt
    M1_results.txt
    ...
  greedy/
    S1_results.txt
    ...
  tabu_search/
    ...
  summary_YYYY-MM-DD.csv
```

A.3.3 Result Analysis and Visualization

To generate performance summaries and visualizations:

```
# Summarize results to CSV/LaTeX tables
.\summarize_results.ps1

# Generate figures (requires Python with matplotlib, pandas)
python scripts/generate_visualizations.py

# Compile LaTeX tables
cd figures
pdflatex performance_table.tex
```

This produces:

- `figures/performance_table.tex`: Solution quality comparison
- `figures/runtime_comparison.pdf`: Runtime bar charts
- `figures/runtime_vs_size.pdf`: Scalability analysis
- `figures/solution_quality_vs_size.pdf`: Quality trends

A.4 Extending the Implementation

A.4.1 Adding New Instance Generators

To create custom instances:

```
python scripts/generate_instance.py \
  --facilities 300 \
  --customers 1500 \
  --budget 25 \
  --radius 4.0 \
  --output data/custom_instance.dat
```

A.4.2 Implementing Algorithm Variants

To create a new algorithm variant:

1. Copy an existing `.mos` file (e.g., `mclp_local_search.mos`)
2. Modify the search strategy while preserving data loading and validation logic
3. Add new parameters as needed in the `parameters` block
4. Test on small instances before scaling to large problems
5. Document changes in code comments

A.4.3 Performance Tuning Guidelines

For optimal performance on specific instance classes:

1. **Exact Solver:**
   - Reduce `MIP_GAP` for tighter optimality on small instances
   - Increase `TIME_LIMIT` for challenging instances
   - Consider enabling additional Xpress controls (e.g., `XPRS_HEURSTRATEGY`)
2. **Tabu Search:**
   - Increase `MAX_ITER` for larger or more challenging instances
   - Tune `TABU_TENURE`: typically $\sqrt{|I|}$ to $2\sqrt{|I|}$
   - Adjust `CANDIDATE_SIZE`: larger for dense instances, smaller for sparse
3. **Multi-Start:**
   - Scale `NUM_STARTS` with instance difficulty (5-20 typical range)
   - Consider parallel execution for independent starts

A.5 Software Requirements and Dependencies

Required Software:

- FICO Xpress Mosel 5.0 or later
- FICO Xpress Optimizer 12.0 or later (for exact solver)
- Python 3.7+ (for instance generation and visualization)

Python Dependencies: .

```
pip install numpy pandas matplotlib scipy
```

License Considerations:

- Xpress Community Edition: Free, limited to 5000 variables/constraints
- Xpress Commercial License: Required for instances exceeding community limits
- Heuristics have no license restrictions

A.6 Troubleshooting

A.6.1 Common Issues

Issue: Exact solver fails with "too many variables"

- **Cause:** Xpress Community Edition limit exceeded
- **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)
- **Solution:** Use Local Search or Multi-Start for massive instances (>2000 customers)

Issue: Inconsistent results across runs

- **Cause:** Randomized algorithms (Multi-Start, random initialization)
- **Solution:** Set random seed or use deterministic algorithms (Greedy, single-start Local Search)

Issue: Slow performance on medium instances

- **Cause:** Inefficient data structures or coverage evaluation
- **Solution:** Enable delta-evaluation, reduce `MAX_ITER`, or use Greedy for baseline

A.7 Best Practices

1. **Always validate data files** before running algorithms (check for consistency, proper indexing)
2. **Start with small instances** when testing new algorithm variants
3. **Use Greedy as a quick baseline** to verify problem setup and expected objective range
4. **Document parameter choices** and rationale for reproducibility
5. **Save results systematically** with timestamps and instance metadata
6. **Compare multiple algorithms** on the same instance to understand trade-offs
7. **Monitor memory usage** for very large instances (>10,000 customers)

A.8 Citation and Licensing

If you use these implementations in your research or applications, please cite:

> *A Computational Study of Exact and Heuristic Algorithms for the Maximal Covering Location Problem.* MCLP Optimization Suite Technical Report, 2025.

All source code is provided for educational and research purposes. Commercial use should acknowledge the original work and comply with Xpress licensing requirements.