Maximizing Coverage Profit for EV Charging Stations

Chapter 1 Problem Background and Description

1.1 Problem Scenario

We consider the following combinatorial optimization problem. In a city, there are several residential buildings. Let the total number of buildings be $n \in \mathbb{Z}_+$, and denote the building index set by $\mathcal{I} = \{1, 2, \ldots, n\}$. For each building $i \in \mathcal{I}$, the potential number of users who need to use EV charging piles is given by $D_i \geq 0$. Serving one user from building i generates a unit profit $p_i \geq 0$, which may vary across buildings.

The planning department provides m candidate areas for installing charging stations, with index set $\mathcal{J} = \{1, 2, \ldots, m\}$. For each area $j \in \mathcal{J}$, the set of buildings it can serve is known and represented by a coverage matrix $a_{ij} \in \{0, 1\}$. Different areas can have overlapping coverage, i.e., some buildings may be covered by multiple areas.

If we decide to build a station in area j, we must pay a fixed construction cost $c_j \geq 0$, which may include civil works, power connection, equipment installation, permits, and other related expenses. Within area j, several charging piles can be installed; let the number of piles be x_j. Due to physical, planning, or capacity limitations, there is an upper bound $U_j \in \mathbb{Z}_+$ on the number of piles that can be installed in area j, beyond which construction is infeasible.

For simplicity, we assume each charging pile can serve one “unit user” (or equivalently, the service capability of each pile has been normalized into a single user unit). Therefore, the total number of users that area j can serve is bounded above by x_j.

Our goal is, under budget and capacity constraints, to decide **which areas to build, how many piles to install in each built area, and how to assign users from buildings to areas under coverage constraints**, so that the **net profit** (total profit from served users minus total construction cost) is maximized.

Chapter 2 Mathematical Modeling

2.1 Parameter Definitions

Sets and indices:

- $\mathcal{I} = \{1, 2, \ldots, n\}$: index set of buildings, where n is the total number of buildings.
- $\mathcal{J} = \{1, 2, \ldots, m\}$: index set of areas, where m is the total number of candidate areas.

Input parameters:

- $D_i \geq 0$: potential number of users in building i.
- $p_i \geq 0$: profit generated by serving one user from building i.
- $c_j \geq 0$: fixed construction cost if area j is built.
- $U_j \in \mathbb{Z}_+$: upper bound on the number of charging piles in area j.
- $a_{ij} \in \{0, 1\}$: coverage matrix, where $a_{ij} = 1$ means area j can cover building i.

2.2 Decision Variables

To describe the decision process, we introduce the following decision variables:

1. Construction decision variables

Let binary decision variable z_j denote whether to build a station in area j. $z_j \in \{0, 1\}$.

2. Charging pile configuration variables

Let x_j denote the number of charging piles installed in area j. To reflect the discreteness of piles, we require $x_j \in \mathbb{Z}_+$.

3. User assignment variables

Let y_{ij} denote the number of users from building i who are assigned to area j for charging. For every building–area pair (i, j), we impose $y_{ij} \geq 0$.

2.3 Constraints

1. Coverage and construction logic constraints

Only when area j covers building i and area j is built can users from building i be assigned to area j. This logic can be captured by the following constraint:

$$y_{ij} \leq D_i \, a_{ij} \, z_j, \quad \forall i \in \mathcal{I}, \; \forall j \in \mathcal{J}.$$

When $a_{ij} = 0$ or $z_j = 0$, the above constraint forces $y_{ij} = 0$, which forbids assigning users from building i to an area that does not cover it or has not been built.

2. User demand constraints

The total number of users assigned from each building cannot exceed its potential demand. For each $i \in \mathcal{I}$:

$$\sum_{j \in \mathcal{J}} y_{ij} \leq D_i,$$

which means that the total number of assigned users from building i cannot exceed its potential demand D_i.

3. Area capacity constraints

The service capacity of area j is limited by the number of piles installed. Since each pile can serve one unit user, the total number of users area j can serve is bounded by x_j. Thus, for each $j \in \mathcal{J}$, we have the capacity constraint

$$\sum_{i \in \mathcal{I}} y_{ij} \leq x_j.$$

4. Coupling constraints for the number of piles

The number of piles is also limited by the upper bound U_j, and piles can only be installed in areas that are actually built. Therefore, for each $j \in \mathcal{J}$, we introduce the following coupling constraint:

$$0 \leq x_j \leq U_j \, z_j.$$

When $z_j = 0$, this constraint forces $x_j = 0$, indicating that no piles can be installed in an unbuilt area. When $z_j = 1$, x_j is allowed to take any value in $[0, U_j]$.

5. Variable domain constraints

$$z_j \in \{0,1\}, \quad x_j \geq 0, \quad y_{ij} \geq 0, \quad \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}.$$

2.4 Objective Function

1. Total revenue

The total revenue from all successfully served users can be written as

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i \, y_{ij}.$$

2. Total construction cost

The total fixed construction cost for building all selected areas is

$$\sum_{j \in \mathcal{J}} c_j \, z_j.$$

3. Net profit objective

We seek to maximize the net profit, i.e., total revenue minus total construction cost:

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i \, y_{ij} \; - \; \sum_{j \in \mathcal{J}} c_j \, z_j.$$

2.5 Complete Mathematical Programming Model

Combining the objective function and all constraints, we obtain the mathematical programming model for the charging station location and configuration problem:

$$\begin{aligned}
\max_{z,x,y} \quad & \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i \, y_{ij} \; - \; \sum_{j \in \mathcal{J}} c_j \, z_j, & \\
\text{s.t.} \quad & y_{ij} \leq D_i \, a_{ij} \, z_j, & \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}, \\
& \sum_{j \in \mathcal{J}} y_{ij} \leq D_i, & \forall i \in \mathcal{I}, \\
& \sum_{i \in \mathcal{I}} y_{ij} \leq x_j, & \forall j \in \mathcal{J}, \\
& 0 \leq x_j \leq U_j \, z_j, & \forall j \in \mathcal{J}, \\
& z_j \in \{0,1\},\ x_j \geq 0,\ y_{ij} \geq 0, & \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}.
\end{aligned}$$

This model captures, under overlapping coverage, construction costs, and capacity upper bounds, **which areas to build**, **how many piles to install** in each built area, and **how to assign users from different buildings to areas**. The objective is to maximize the overall economic profit. In essence, this is a **mixed-integer linear programming (MILP)** problem.

Chapter 3 Complexity Analysis and Reduction Proof

3.1 Formalization of the Decision Problem

Based on the mathematical model above, we now analyze the problem from the perspective of computational complexity. We show that the decision version of this charging station location and configuration problem belongs to NP, and prove via a polynomial-time reduction from the Set Cover problem that it is NP-hard. Hence, its decision version is NP-complete and

the optimization version is NP-hard.

We first rewrite the original **optimization problem** as a **decision problem**. Given:

- Building set $\mathcal{I} = \{1, \ldots, n\}$ and area set $\mathcal{J} = \{1, \ldots, m\}$,
- Potential demands $D_i \geq 0$ and unit profits $p_i \geq 0$,
- Area construction costs $c_j \geq 0$ and upper bounds $U_j \in \mathbb{Z}_+$,
- Coverage matrix $a_{ij} \in \{0, 1\}$,
- And a target threshold $K \in \mathbb{R}$,

we consider the following decision problem:

> **Charging Station Location Decision Problem**:
> Does there exist a set of decision variables (z, x, y) satisfying all constraints
>
> $$\begin{aligned} & y_{ij} \leq D_i a_{ij} z_j, && \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}, \\ & \sum_{j \in \mathcal{J}} y_{ij} \leq D_i, && \forall i \in \mathcal{I}, \\ & \sum_{i \in \mathcal{I}} y_{ij} \leq x_j, && \forall j \in \mathcal{J}, \\ & 0 \leq x_j \leq U_j z_j, && \forall j \in \mathcal{J}, \\ & z_j \in \{0, 1\},\ x_j \geq 0,\ y_{ij} \geq 0, && \forall i \in \mathcal{I},\ \forall j \in \mathcal{J}, \end{aligned}$$
>
> such that the objective value satisfies
>
> $$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i\, y_{ij} - \sum_{j \in \mathcal{J}} c_j\, z_j \ \geq K\ ?$$

3.2 Membership in NP

Given a candidate solution (z, x, y), we can verify whether it is a valid certificate in polynomial time:

1. Check linear constraints

For each pair (i, j), check
$y_{ij} \leq D_i a_{ij} z_j$;
for each i, check $\sum_j y_{ij} \leq D_i$;
for each j, check $\sum_i y_{ij} \leq x_j$ and $0 \leq x_j \leq U_j z_j$.
There are $\mathcal{O}(nm + n + m)$ linear inequalities, and computing and comparing each left-hand side and right-hand side can be done in polynomial time.

2. Check variable domains

For each j, check $z_j \in \{0, 1\}$; for each j, check $x_j \geq 0$; for each (i, j), check $y_{ij} \geq 0$. The number of inequalities is $\mathcal{O}(nm)$, again polynomial.

3. Compute the objective and compare with the threshold

In $\mathcal{O}(nm + m)$ time we can compute

$$\sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i\, y_{ij} - \sum_{j \in \mathcal{J}} c_j\, z_j$$

and compare it to the threshold K.

Assuming all input parameters ($D_i, p_i, c_j, U_j, a_{ij}, K$, etc.) are encoded in binary, the total verification time is polynomial in the input size. Therefore, the decision problem belongs to class NP: once a candidate solution is given, we can check feasibility and whether the objective reaches K in polynomial time.

3.3 Reduction from Set Cover

To prove that this problem is NP-hard, we construct a polynomial-time reduction from the classical Set Cover problem.

3.3.1 Review of the Set Cover Problem

The Set Cover decision problem is defined as follows:

- Given a finite universe

$$U = \{e_1, e_2, \ldots, e_n\},$$

- A family of subsets

$$\mathcal{S} = \{S_1, S_2, \ldots, S_m\}, \quad S_j \subseteq U,$$

- And a positive integer k,

the question is: does there exist an index set $J^\star \subseteq \{1, \ldots, m\}$, such that $|J^\star| \leq k$ and

$$\bigcup_{j \in J^\star} S_j = U \ ?$$

This problem is a well-known NP-complete problem. We will show that, for any Set Cover instance $(U, \mathcal{S}, k)$, we can construct in polynomial time a charging station location decision instance such that the Set Cover instance has a cover of size at most k if and only if the constructed instance admits a feasible solution with objective at least a threshold K.

3.3.2 Steps of the Reduction

Given a Set Cover instance $(U, \mathcal{S}, k)$ with

$$U = \{e_1, \ldots, e_n\}, \quad \mathcal{S} = \{S_1, \ldots, S_m\},$$

we construct a charging station instance as follows.

Step 1: Map buildings to elements

Map each element $e_i \in U$ to a building i. The building set is

$$\mathcal{I} = \{1, \ldots, n\}, \quad n = |U|.$$

Set the potential demand of each building to

$$D_i = 1, \quad \forall i \in \mathcal{I},$$

meaning each element needs to be "served" at most once. For simplicity, set the profit of serving each user to the same constant:

$$p_i = M, \quad \forall i \in \mathcal{I},$$

where the constant M is chosen as

$$M = m + 1.$$

Step 2: Map areas to subsets

Map each subset S_j to a candidate area j. The area set is

$$\mathcal{J} = \{1, \ldots, m\}, \quad m = |\mathcal{S}|.$$

Define the coverage matrix a_{ij} by

$$a_{ij} = \begin{cases} 1, & \text{if } e_i \in S_j, \\ 0, & \text{otherwise}, \end{cases} \qquad \forall i \in \mathcal{I},\ \forall j \in \mathcal{J},$$

so that area j covers exactly those buildings corresponding to elements in S_j.

Step 3: Capacities and construction costs

To avoid capacity constraints limiting coverage, set the upper bound on the number of piles in each area to

$$U_j = n, \quad \forall j \in \mathcal{J},$$

so that x_j can be large enough to serve all buildings it covers (since each building has at most 1 user). Set the construction cost to a constant

$$c_j = 1, \quad \forall j \in \mathcal{J},$$

so selecting each area costs 1. With large capacities, whenever $z_j = 1$, area j can fully serve all buildings it covers.

Step 4: Setting the threshold

The objective in the charging problem is

$$\text{Profit}(z, x, y) = \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} p_i \, y_{ij} - \sum_{j \in \mathcal{J}} c_j \, z_j.$$

In this construction, since $D_i = 1$, each building i is either fully served by some area (contributing profit M), or not served at all (contributing 0). Let the set of covered buildings be

$$C(J) = \{i \in \mathcal{I} : \text{there exists } j \in \mathcal{J} \text{ with } y_{ij} = 1\},$$

and the set of selected areas be

$$J = \{j \in \mathcal{J} : z_j = 1\}.$$

Then the objective can be rewritten as

$$\text{Profit}(J) = M \cdot |C(J)| - |J|.$$

We choose the threshold as

$$K = Mn - k.$$

Step 5: Size and time of the construction

Constructing $D_i, p_i, c_j, U_j, a_{ij}, K$ from $(U, \mathcal{S}, k)$ only requires scanning all elements and subsets to compute a_{ij}, with time complexity $\mathcal{O}(nm)$, which is polynomial. Thus, the reduction is polynomial-time.

3.4 Proof of Correctness of the Reduction

We prove the following equivalence:

> **Proposition**:
> The original Set Cover instance has a set cover of size at most k
> if and only if
> the constructed charging station decision instance has a feasible solution with
>
> $$\text{Profit}(z, x, y) \geq K = Mn - k.$$

3.4.1 Sufficiency

If there exists a set cover, then there exists a solution with profit at least K.

Assume that in the Set Cover instance there is an index set $J^\star \subseteq \{1, \ldots, m\}$ such that

$$|J^\star| \leq k, \quad \bigcup_{j \in J^\star} S_j = U.$$

In the charging instance, construct a solution as follows:

- For $j \in J^\star$, set $z_j = 1$; for $j \notin J^\star$, set $z_j = 0$.
- For each selected area $j \in J^\star$, choose sufficiently many piles, e.g., $x_j = n$, which clearly satisfies $x_j \leq U_j z_j$.
- Since $J^\star$ is a set cover, each building i is covered by at least one area $j \in J^\star$ with $a_{ij} = 1$. Choose one such area and set $y_{ij} = 1$, and $y_{ij'} = 0$ for all other j'. Then for all i we have $\sum_j y_{ij} = 1 \leq D_i$, and for each j:

  $$\sum_i y_{ij} \leq n = x_j.$$

  All constraints are satisfied.

In this case, all buildings are covered, i.e., $C(J^\star) = \mathcal{I}$, so $|C(J^\star)| = n$. The objective is

$$\text{Profit}(J^\star) = Mn - |J^\star| \geq Mn - k = K.$$

Thus, if the Set Cover instance has a cover of size at most k, then the charging instance admits a feasible solution with objective at least K.

3.4.2 Necessity

If there exists a solution with profit at least K, then there exists a set cover of size at most k.

Conversely, suppose in the charging instance there exists a feasible solution (z, x, y) whose corresponding area set

$$J = \{j \in \mathcal{J} : z_j = 1\}$$

satisfies

$$\text{Profit}(J) = M|C(J)| - |J| \geq K = Mn - k.$$

We first show that this solution must cover all buildings, i.e., $|C(J)| = n$.

If there exists some building i that is not served, then $i \notin C(J)$ and hence $|C(J)| \leq n - 1$. Since $|J| \geq 0$, we have

$$\text{Profit}(J) = M|C(J)| - |J| \leq M(n-1) - 0 = M(n-1).$$

On the other hand,

$$K = Mn - k \geq Mn - m.$$

Substituting $M = m + 1$ gives

$$Mn - m = (m+1)n - m = (m+1)(n-1) + (m+1-m) = M(n-1) + 1.$$

Thus,

$$K \geq M(n-1) + 1 > M(n-1).$$

This contradicts $\text{Profit}(J) \leq M(n-1)$ and $\text{Profit}(J) \geq K$. Therefore, any feasible solution with $\text{Profit}(J) \geq K$ must satisfy $|C(J)| = n$, i.e., all buildings are covered.

When $|C(J)| = n$, the objective simplifies to

$$\text{Profit}(J) = Mn - |J|.$$

The condition $\text{Profit}(J) \geq K = Mn - k$ is equivalent to

$$Mn - |J| \geq Mn - k \quad \Longleftrightarrow \quad |J| \leq k.$$

Since $|C(J)| = n$, every building i is covered by some $j \in J$ with $a_{ij} = 1$, which implies

$$\{e_i : i \in \mathcal{I}\} \subseteq \bigcup_{j \in J} S_j.$$

From the one-to-one correspondence between buildings and elements, we know

$$\bigcup_{j \in J} S_j = U,$$

with $|J| \leq k$. Hence, the set J gives a set cover of size at most k for the original Set Cover instance.

In summary, if there exists a feasible solution to the charging instance with profit at least K, then there exists a set cover of size at most k in the Set Cover instance.

3.5 Complexity Conclusion

The above construction is polynomial in the input size and satisfies:

- The Set Cover instance is a “Yes” instance if and only if
- The corresponding charging station decision instance is also a “Yes” instance (there exists a solution with objective $\geq K$).

Therefore, we have a polynomial-time reduction from the Set Cover decision problem to the charging station location decision problem. Since Set Cover is NP-complete and we have already shown that our decision problem is in NP, it follows that:

- **The charging station location decision problem is NP-complete**;
- **The corresponding optimization problem (maximizing profit) is NP-hard**.

This means that, in general, unless $\text{P} = \text{NP}$, there is no polynomial-time algorithm that always finds a global optimum. From a complexity perspective, this is a typical hard combinatorial optimization problem.

Chapter 4 Algorithm Design

4.1 Unified Optimal Assignment Strategy: Min-Cost Max-Flow Method

All algorithms use a shared subproblem strategy: once the location decision z is fixed, the number of piles x is set directly to the capacity limit ($x_j = U_j \cdot z_j$), and then a **min-cost max-flow algorithm** is used to compute the optimal user assignment y exactly.

4.1.1 Core Idea and Theoretical Basis

Key observation: Since adding more piles does not incur additional marginal cost (only the fixed construction cost c_j matters), it is always optimal to set the number of piles in each chosen area to its upper bound. Hence, the subproblem reduces to: given a construction plan z and $x_j = U_j \cdot z_j$, how do we optimally assign users y?

Theoretical basis:

- With fixed z and x, the user assignment subproblem can be modeled as a min-cost max-flow problem.
- Min-cost max-flow can be solved exactly in polynomial time, with complexity $O((n+m)^2 \times \log(n+m) \times C)$.
- Since the constraint matrix in network flow problems is totally unimodular, the algorithm automatically returns integer solutions, ensuring $y_{ij} \in \mathbb{Z}_+$.

4.1.2 Network Flow Modeling

We transform the user assignment problem into a min-cost max-flow problem.

Network structure:

- **Node set**: $\{s, t\} \cup \mathcal{I} \cup S$, where $S = \{j \in \mathcal{J} : z_j = 1\}$ is the set of selected areas:
  - Source node s,
  - Building nodes i (n nodes),
  - Area nodes j (only those $j \in S$),
  - Sink node t.

Directed edges with capacities and costs:

1. **Source to buildings**: $s \to i$ with capacity D_i and cost 0.
2. **Buildings to areas**: $i \to j$ (if $a_{ij} = 1$ and $z_j = 1$) with capacity D_i and cost $-p_i$.
3. **Areas to sink**: $j \to t$ (for $j \in S$) with capacity $U_j \cdot z_j = U_j$ and cost 0.
4. **Buildings directly to sink**: $i \to t$ with capacity D_i and cost 0 (representing unserved users).

Flow balance:

- Total supply at the source s: $\sum_i D_i$.
- Total demand at the sink t: $\sum_i D_i$.

4.1.3 Solution Method and Objective

We use the `max_flow_min_cost` algorithm (e.g., from the NetworkX library) to solve the min-cost max-flow problem. This algorithm is typically based on successive shortest path or cost-scaling methods.

Relation to the original objective:

- The total cost of the flow is:

$$\text{Total Cost} = \sum_{i,j} (-p_i) \cdot y_{ij} = -\sum_{i,j} p_i y_{ij}.$$

- Minimizing total cost is equivalent to maximizing total revenue $\sum_{i,j} p_i y_{ij} = -\min \text{Total Cost}$.

4.1.4 Complexity and Optimality Guarantees

Time complexity:

- Number of nodes: $O(n+m)$ (actually $n + |S| + 2$, where $|S| \leq m$).
- Number of edges: $O(n \times m)$ (worst case when every building connects to every area).
- **Overall time complexity**: $O((n+m)^2 \times \log(n+m) \times C)$, where $C = \max\{D_i, U_j\}$ is the maximum capacity. This is polynomial.

Space complexity:

- Storing the network graph: $O(n \times m)$ (edges).
- Storing flow results: $O(n \times m)$.
- **Total space complexity**: $O(n \times m)$.

Optimality guarantee: The min-cost max-flow algorithm finds an optimal solution (for fixed z and x) for the user assignment y. This ensures that all solvers using this method achieve optimal assignment given the chosen locations, which makes algorithm comparisons fair.

4.2 Brute-Force Enumeration

4.2.1 Algorithm Idea

1. **Outer enumeration**: Enumerate all 2^m possible construction plans $z \in \{0,1\}^m$.
2. **Subproblem solution**: For each z:
   - Set the number of piles to $x_j = U_j \cdot z_j$ (fully installed).
   - Use the **min-cost max-flow algorithm** (Section 4.1) to compute the optimal user assignment y.
   - Compute the objective value $\sum_i \sum_j p_i y_{ij} - \sum_j c_j z_j$.
3. **Select the best solution**: Keep track of the (z, x, y) with the highest objective value.
4. Return the global optimum.

4.2.2 Complexity Analysis

Time complexity:

- Number of enumerations: 2^m.
- Min-cost max-flow per enumeration: $O((n+m)^2 \times \log(n+m) \times C)$, where $C = \max\{D_i, U_j\}$.
- **Overall**: $O(2^m \times (n+m)^2 \times \log(n+m) \times C)$.

Space complexity:

- Store the current best solution: $O(n \times m)$ for the y matrix.
- Min-cost max-flow: $O(n+m)$ nodes and $O(n \times m)$ edges.
- **Overall**: $O(n \times m)$.

Applicable size: Feasible when $m \leq 15$; for $m > 20$, exponential blow-up makes this impractical.

4.2.3 Approximation Ratio

Brute-force enumeration finds the global optimum, so the approximation ratio is $\rho = 1$ with no approximation error.

4.2.4 Implementation Details

- Supports a time limit parameter (`time_limit`); if exceeded, returns the best solution found so far.
- For $m > 15$, we may only enumerate the first 1000 solutions to avoid excessive runtime.
- Uses `itertools.product` to efficiently generate all $(0, 1)$ combinations.

4.3 Greedy Construction

4.3.1 Algorithm Idea

Phase A: Construction

1. Initialization: set $z_j = 0$, $x_j = 0$, and $y_{ij} = 0$ for all i, j.
2. Compute a "benefit-to-cost" score for each candidate area j:
   - Potential profit:

$$\text{region\_profits}_j = \bar{p}_j \times \min\left\{\sum_{i:a_{ij}=1} D_i, U_j\right\},$$

     where $\bar{p}_j$ is the average unit profit of buildings covered by area j.
   - Score:

$$\text{score}_j = \frac{\text{region\_profits}_j}{c_j}.$$

3. Sort areas in descending order of score_j.
4. Iteratively consider adding each area j in this order:
   - Temporarily set $z_j = 1$ and $x_j = U_j$.

- Use the **min-cost max-flow algorithm** (Section 4.1) to compute the optimal assignment y and objective value.
- If the objective improves, keep this area; otherwise, revert z_j to 0.

5. For the final set of selected areas, run the min-cost max-flow algorithm once more to compute the final optimal assignment.

Note: Unlike traditional greedy methods that use a simple local assignment rule, this implementation uses min-cost max-flow for user assignment, ensuring that the user assignment is optimal for a given location plan.

4.3.2 Complexity Analysis

Time complexity:

- Score computation: $O(n \times m)$.
- Sorting: $O(m \log m)$.
- Greedy iterations: up to m iterations, each involving one min-cost max-flow call.
- **Overall**: $O(m \times (n+m)^2 \times \log(n+m) \times C)$.

Space complexity:

- Store the current solution: $O(n \times m)$.
- Min-cost max-flow: $O(n \times m)$.
- **Overall**: $O(n \times m)$.

4.3.3 Approximation Ratio

This problem can be viewed as a mixture of "coverage with penalties" and "knapsack-like capacity". Borrowing from Set Cover greedy analysis, in theory:

- $\rho \leq H_n + 1$, where $H_n = 1 + 1/2 + \ldots + 1/n \approx \ln n$ is the n-th harmonic number.
- If all $p_i = 1$, the problem reduces to Set Cover with penalties, and $\rho \leq H_n$.
- For heterogeneous profits, $\rho \leq H_n \cdot (p_{\max}/p_{\min})$.

Empirical performance: Because we use min-cost max-flow to assign users, performance is typically better than classic greedy approaches. Experiments show an empirical approximation ratio around $\rho \approx 1.1 \sim 1.3$.

4.3.4 Advantages and Disadvantages

Advantages:

- Very fast runtime, suitable for large-scale problems.
- Simple to implement and easy to understand.
- Uses min-cost max-flow to guarantee optimal user assignment for a given location plan.

Disadvantages:

- No guarantee of global optimality (location decisions are greedy).
- Can get stuck in local optima.

4.4 Exact MILP-Based Algorithm

4.4.1 Algorithm Idea

1. Directly model the Chapter 2 formulation as a mixed-integer linear program (MILP):
   - Decision variables: $z_j \in \{0, 1\}$ (binary), $x_j \in \mathbb{Z}_+$ (integer), $y_{ij} \in \mathbb{Z}_+$ (integer).
   - Objective: $\max \sum_i \sum_j p_i y_{ij} - \sum_j c_j z_j$.
   - Constraints: coverage constraints, demand constraints, capacity constraints, plus the key constraint $x_j = U_j \cdot z_j$.
2. Use the CBC solver (via PuLP) or other MILP solvers to solve the problem using branch-and-bound.
3. Extract and return the optimal solution.

Note: The MILP method optimizes all variables jointly within the model without explicitly calling the min-cost max-flow algorithm. The constraint $x_j = U_j \cdot z_j$ ensures consistency with the unified strategy.

4.4.2 Complexity Analysis

Time complexity:

- Number of variables: m (z) + m (x) + $n \times m$ (y) = $O(n \times m)$.
- Number of constraints: $O(n \times m)$ (coverage) + n (demand) + m (capacity) + m (upper bounds) = $O(n \times m)$.
- Branch-and-bound worst-case nodes: $O(2^m)$ (branching mainly on z variables).
- LP relaxation per node: $O((n \times m)^{3.5})$ (worst-case complexity of interior-point methods).
- **Overall**: Exponential in the worst case, roughly $O(2^m \times (n \times m)^{3.5})$. In practice, due to pruning and preprocessing, performance is usually better.

Space complexity:

- Store the MILP model: $O(n \times m)$ (constraint matrix).
- Branch-and-bound tree: worst case $O(2^m)$ nodes.
- **Overall**: $O(n \times m + 2^m)$, often much smaller in practice thanks to pruning.

Applicable size: Medium-scale problems (e.g., $m \leq 30$, $n \leq 50$), depending on the solver.

4.4.3 Approximation Ratio

When the MILP solver finds the global optimum, the approximation ratio is $\rho = 1$. With time limits, it returns the best feasible solution found so far along with a bound on the optimal value (gap).

4.4.4 Implementation Details

- We use PuLP for modeling, with CBC as the default open-source solver.
- The model can be switched to other solvers (e.g., Gurobi, CPLEX) for better performance.
- All variables x and y are treated as integer (no continuous relaxation).
- The key constraint $x_j = U_j \cdot z_j$ ensures that once an area is chosen, the number of piles is set to its upper bound.

4.5 Genetic Algorithm (GA)

4.5.1 Encoding

- **Chromosome representation**: a binary string $\mathrm{ch} \in \{0,1\}^m$ representing the construction decisions z.
- **Decoding procedure**: for a given chromosome z:
  1. Set $x_j = U_j \cdot z_j$.
  2. Use the **min-cost max-flow algorithm** (Section 4.1) to compute the optimal user assignment y.
  3. Compute the objective value $\sum_i \sum_j p_i y_{ij} - \sum_j c_j z_j$.

4.5.2 Fitness

The fitness function is the objective value:
$\mathrm{fitness}(\mathrm{ch}) = \sum_i \sum_j p_i y_{ij} - \sum_j c_j z_j$.

Note: Because min-cost max-flow is used for decoding, the assignment y is optimal for a given z, so the fitness is exact.

4.5.3 Genetic Operators

- **Initialization**: generate the initial population randomly, each gene in the chromosome set to 1 with probability 0.3 (bias toward fewer selected areas).
- **Selection**: roulette-wheel selection (fitness-proportionate), so individuals with higher fitness are more likely to be chosen.
- **Crossover**: single-point crossover with probability $p_c = 0.8$.
- **Mutation**: bit-flip mutation with probability $p_m = 0.1$ (often chosen as $1/m$).
- **Elitism**: retain the top 10% of individuals as elites into the next generation.

4.5.4 Parameters and Stopping Criteria

- Population size $N = 50$ (default).
- Maximum number of generations $G = 100$ (default).
- **Early stopping**: if there is no improvement for `early_stop_patience = 20` successive generations, terminate early.
- Other parameters: crossover rate 0.8, mutation rate 0.1, elite fraction 0.1.

4.5.5 Complexity Analysis

Time complexity:

- Initialization: $O(N \times m)$.
- Fitness evaluation per generation: N decodings, each with one min-cost max-flow call.
- Selection, crossover, mutation: $O(N \times m)$ per generation.
- **Overall**: $O(G \times N \times (n+m)^2 \times \log(n+m) \times C)$.

Space complexity:

- Population storage: $O(N \times m)$.

- Min-cost max-flow: $O(n \times m)$.
- **Overall**: $O(N \times m + n \times m)$.

4.5.6 Approximation Ratio and Performance

As a metaheuristic, the genetic algorithm does **not** offer a theoretical approximation guarantee, but performs well in practice:

- **Compared with brute-force**: for $m \leq 20$, it usually finds the optimal or near-optimal solution (gap $\leq 1\%$).
- **Compared with greedy**: typically yields 3–8% improvement on average, especially for larger instances.
- **Convergence**: the early stopping mechanism ensures the algorithm does not run unnecessarily long; in practice, 20–50 generations are often sufficient.

4.5.7 Advantages and Disadvantages

Advantages:

- Can escape local optima and is suitable for large-scale problems.
- Uses min-cost max-flow to ensure an optimal assignment for each chromosome.
- Early stopping improves efficiency.
- Parameters are tunable and the method is flexible.

Disadvantages:

- No guarantee of global optimality.
- Sensitive to parameter choices.
- May require a relatively large number of iterations.

4.6 Algorithm Summary and Comparison

Algorithm	Optimality Guarantee	Applicable Scale	Time Complexity	Space Complexity	Main Advantages	Main Disadvantages
Brute-force	✓ Global optimum	$m \leq 15$	$O(2^m \times \text{MCF})$	$O(n \times m)$	Benchmark method; results are reliable	Exponential blow-up; only feasible for small m
Greedy	✗ Theoretical H_n-approx	Any scale	$O(m \times \text{MCF})$	$O(n \times m)$	Very fast; easy to implement; uses min-cost max-flow	May get stuck in local optima
MILP	✓ Global optimum	Medium scale ($m \leq 30$)	Exponential (branch-and-bound)	$O(n \times m + 2^m)$	Guaranteed optimality; good for medium instances	Runtime may become large for big instances
GA	✗ Metaheuristic	Large scale ($m \leq 500$)	$O(G \times N \times \text{MCF})$	$O(N \times m + n \times m)$	Easily parallelized; uses min-cost max-flow; early stopping	Parameter tuning; no theoretical guarantees

Here,

- $\text{MCF} = O((n+m)^2 \times \log(n+m) \times C)$ is the complexity of min-cost max-flow (polynomial time),
- G is the number of generations, N is the population size,

- $C = \max\{D_i, U_j\}$ is the maximum capacity.

Key design features:

- All algorithms share the unified strategy from Section 4.1: once locations are fixed, the number of piles is set to the upper bound, and min-cost max-flow is used to optimally assign users.
- This ensures that, for any given location plan, each algorithm attains the best possible user assignment, making comparisons fair.
- The min-cost max-flow algorithm provides an efficient and exact polynomial-time method for the large linear subproblem, which is more efficient and accurate than traditional LP or simple greedy assignments.

Chapter 5 Experimental Results and Analysis

This chapter analyzes the performance of the four methods using experimental data from 50 problem instances, covering five scales: $n \in \{10, 15, 20, 25, 30\}$ and $m \in \{5, 8, 10, 12, 15\}$.

5.1 Experimental Statistics

The table below reports, for different problem sizes, the **number of times each method reaches the optimal solution** (out of 10 instances) and the **average runtime**:

Problem size	Method	Times reaching optimum	Probability of optimum	Avg runtime
$n = 10, m = 5$	Greedy	9/10	90%	15.8ms
	MILP solver	10/10	**100%**	12.8ms
	Brute-force	10/10	**100%**	32.6ms
	GA	10/10	**100%**	1.221s
$n = 15, m = 8$	Greedy	7/10	70%	14.2ms
	MILP solver	10/10	**100%**	21.0ms
	Brute-force	10/10	**100%**	387.0ms
	GA	10/10	**100%**	2.263s
$n = 20, m = 10$	Greedy	7/10	70%	24.0ms
	MILP solver	10/10	**100%**	45.9ms
	Brute-force	10/10	**100%**	2.112s
	GA	10/10	**100%**	3.452s

Problem size	Method	Times reaching optimum	Probability of optimum	Avg runtime
$n = 25, m = 12$	Greedy	7/10	70%	37.9ms
	MILP solver	10/10	**100%**	45.8ms
	Brute-force	10/10	**100%**	11.145s
	GA	10/10	**100%**	5.798s
$n = 30, m = 15$	Greedy	7/10	70%	61.4ms
	MILP solver	10/10	**100%**	92.2ms
	Brute-force	0/10	**0%**	60.0s (timeout)
	GA	8/10	80%	8.284s

5.2 Result Analysis

1. Solution optimality

- **MILP solver** achieves a 100% optimality rate across all tested scales, confirming its reliability as an exact method.
- **Brute-force** performs perfectly (100% optimality) for $m \leq 12$, but fails to finish within the time limit when $m = 15$ due to exponential complexity.
- **Genetic algorithm (GA)** achieves 100% optimality for small to medium instances ($m \leq 12$), but drops to 80% when $m = 15$.
- **Greedy algorithm** maintains an optimality probability between 70–90%, with about one quarter of instances failing to reach the optimum.

2. Runtime performance

- **Greedy algorithm** is the fastest method across all scales (15–61ms), and its runtime grows slowly with problem size, making it suitable for real-time decisions.
- **MILP solver** is also fast for small and medium instances (12–46ms), and averages about 92ms at $m = 15$ (with some instances taking longer).
- **Brute-force** shows exponential growth in runtime: 32ms at $m = 5$, over 11 seconds at $m = 12$, and reaches the time limit at $m = 15$.
- **Genetic algorithm** has longer runtimes (seconds), but the runtime grows relatively smoothly and is acceptable for larger problems.

3. Method characteristics

Method	Main advantages	Main disadvantages	Suitable scenarios
Greedy	Fastest (millisecond-level); simple implementation	Limited solution quality (70–90% optimality)	Real-time decisions; large-scale preliminary planning
MILP solver	Guaranteed optimal solutions (100%); fast on small/medium problems	May be slow for large m; depends on solver	Exact optimization for medium-sized instances
Brute-force	Guaranteed optimality for small m; good for validation	Exponential complexity; infeasible for $m > 12$	Small-scale problems ($m \leq 12$); algorithm benchmarking
Genetic algorithm	High-quality solutions (80–100% optimality); scalable	Longer runtime (seconds); no theoretical guarantee	Near-optimal solutions for large-scale problems

5.3 Recommendations for Algorithm Selection

- $m \leq 10$: Prefer **MILP solver** (guarantees optimality with fast runtime).
- $10 < m \leq 15$: Prefer **MILP solver**; if time is limited, consider **GA** or **Greedy**.
- $m > 15$: Prefer **GA** for high-quality solutions; use **Greedy** when fast responses are required.
- **Real-time decision-making**: Use the **Greedy algorithm** (fastest).
- **When global optimality is required**: Use the **MILP solver**, the only method that consistently guarantees 100% optimality across all tested scales.