Facility Location Optimization Problem

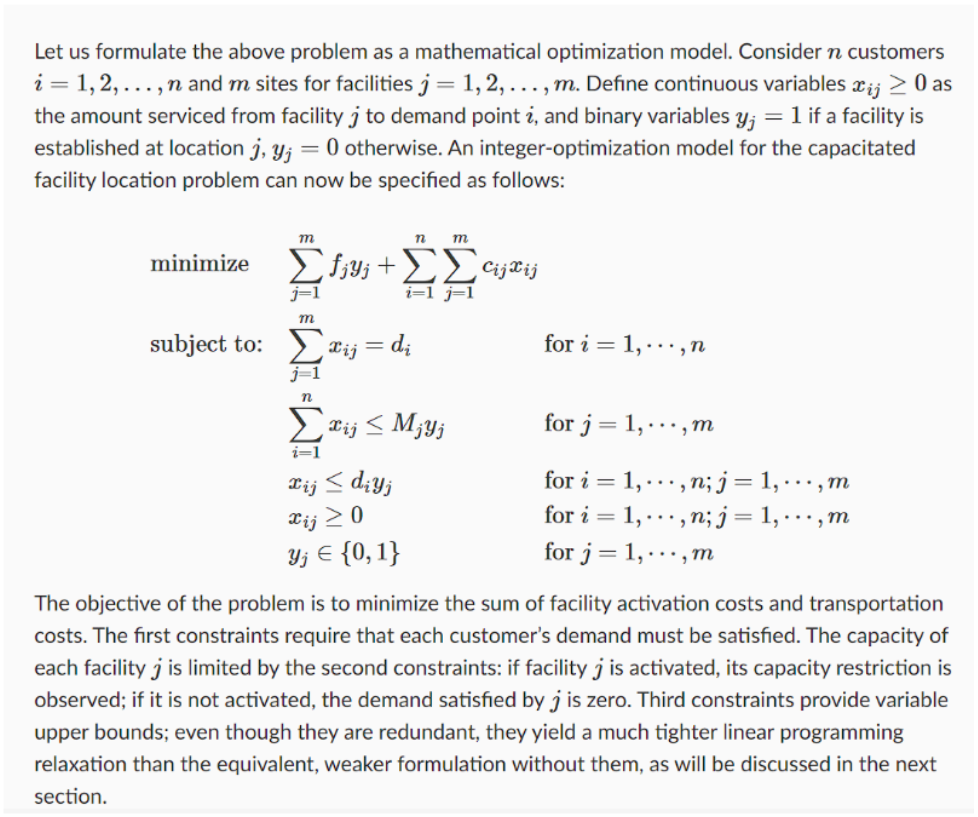
1. Introduction

   The Facility Location Optimization problem is a fundamental decision-making issue faced by businesses that need to distribute products or services to a dispersed customer base. The goal is to determine the most cost-effective locations for facilities such that customer demand is met and the overall cost, including facility activation and transportation, is minimized. This problem is modeled using a set of n customers and m potential facility sites, with constraints ensuring that customer demands are satisfied and facility capacities are not exceeded. Solutions to this problem have significant implications for operational efficiency and service quality. In this report, we explore a novel approach to solving the Facility Location Optimization problem by applying a genetic algorithm, offering a balance between solution quality and computational efficiency.

2. Methods

   2.1 Analysis of the problem

   Let's take a better look at the problem:



Let us formulate the above problem as a mathematical optimization model. Consider n customers $i = 1, 2, \ldots, n$ and m sites for facilities $j = 1, 2, \ldots, m$. Define continuous variables $x_{ij} \geq 0$ as the amount serviced from facility j to demand point i, and binary variables $y_j = 1$ if a facility is established at location j, $y_j = 0$ otherwise. An integer-optimization model for the capacitated facility location problem can now be specified as follows:

$$
\begin{array}{lll}
\text{minimize} & \displaystyle\sum_{j=1}^{m} f_j y_j + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij} x_{ij} & \\
\text{subject to:} & \displaystyle\sum_{j=1}^{m} x_{ij} = d_i & \text{for } i = 1, \cdots, n \\
 & \displaystyle\sum_{i=1}^{n} x_{ij} \leq M_j y_j & \text{for } j = 1, \cdots, m \\
 & x_{ij} \leq d_i y_j & \text{for } i = 1, \cdots, n; j = 1, \cdots, m \\
 & x_{ij} \geq 0 & \text{for } i = 1, \cdots, n; j = 1, \cdots, m \\
 & y_j \in \{0, 1\} & \text{for } j = 1, \cdots, m
\end{array}
$$

The objective of the problem is to minimize the sum of facility activation costs and transportation costs. The first constraints require that each customer's demand must be satisfied. The capacity of each facility j is limited by the second constraints: if facility j is activated, its capacity restriction is observed; if it is not activated, the demand satisfied by j is zero. Third constraints provide variable upper bounds; even though they are redundant, they yield a much tighter linear programming relaxation than the equivalent, weaker formulation without them, as will be discussed in the next section.

As described, the problem is a mixed-integer linear programming (MILP) problem. It is "mixed" because it includes both continuous variables $x_{ij} \geq 0$ (representing the amount serviced from facility j to demand point i) and integer variables $y_i \in \{0, 1\}$ (binary variables indicating whether a facility is established at location j or not). The objective function and constraints are linear with respect to these variables.

In this scenario, there are two related spaces our algorithm must explore, the continuous amount service per facility to the customer and the integer binary variable representing whether a facility is closed or not. Let's call the last space $y_i \in \{0, 1\}$ the First Space and $x_{ij} \geq 0$ the Second Space. The proposed algorithm shall explore both these spaces in a hierarchical manner.

2.3 Genetic Algorithms

2.3.1 Structure

The algorithm proposed follows the following structure:

1. Exploration of First Space: First explore the First Space using a proper genetic algorithm. The individuals are binary strings representing the close-open state of each facility. The fitness function of each individual is performed using a second genetic algorithm which will explore the Second Space just a bit, the fitness will be the overall score of this brief exploration.

2. Exploitation of Second Space: After this initial exploration is performed, the best binary individual will be chosen and now we will run the second algorithm on the Second Space more thoroughly. We will use the same exact algorithm used in the fitness of the first algorithm but now using a bigger population and running it for more time, thus exploiting the Second Space.

2.3.2 First Space algorithm

- Individuals: Are binary strings, for example, [0,1,0,0,1,1,1,0,1]

- Fitness function: The overall performance of another genetic algorithm running on the second space

- Selection: Tournament selection with size 3, that is, 3 individuals will be randomly chosen from the population and compete.

- Crossover: splitting and recombination. A crossover point will be selected which will cut the parents of the individuals and recombine them to form the children. This is a common way of performing a crossover on binary strings.

- Mutation: each element has a random chance of flipping (that is, a facility might get closed of opened)

2.3.3 Second Space algorithm

- Individuals:

The individuals of the second space algorithm deserve more attention. Let's revise the objective of this space. We want to determine the amount of units each facility must serve each demand point, with the constraint that each demand point must receive an exact number of units and each facility does not send more than the amount it can send.

Let's take this output example:

Facility A is open.
Facility B is open.
Facility C is open.

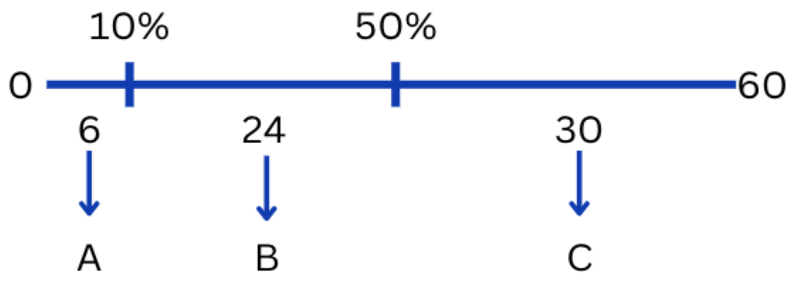
Facility A serves 60 units to demand point 3.
Facility B serves 40 units to demand point 4.
Facility C serves 60 units to demand point 1.
Facility C serves 50 units to demand point 2.

We could represent this distribution as the following matrix:

facility / Demand points	1	2	3	4
A	0	0	60	0
B	0	0	0	40
C	60	50	0	0

The important thing to notice is that the sum of each column must be exactly the demanded units, so it does not matter the distribution per column, as long as the sum is constant. If we are thinking of representing individuals in a genetic algorithm, we must find a way to uniquely represent a distribution like this one. This leads to the question: How can we represent a distribution of units across a single column? The idea I developed is using a percentage partition. For example: Let's say we want to distribute 60 units to facilities A, B and C. I am going to choose two numbers x1 and x2 that should "cut" the number 60 into three slices, which will be how much each facility will receive. For example:

x1 = 10% and x2 = 50%. This split means 10% of the units will be sent from facility A, 40% to facility B and 50% to facility C. Here is an image representing the split:



The question now is how to represent all columns. The answer is simply to extend this idea to more columns. Instead of having "slicers" for a single column we can have slicers for all demand points. Here is an example of an individual:

0.13	0.45	0.90	0
0.67	0.79	1	1

Then the number of columns must be exactly the same as the number of demand points and the number of rows must be one less than the number of facilities.

This representation has a couple of advantages and drawbacks. The positive aspect is the unique representation and easy mutation methods. The drawback is that it does not account for the facility capacity constraint (this will be dealt with later) and furthermore the crossover might be a bit difficult to deal with.

- Fitness function:

The fitness will be the formula in the description of the problem. Here we will use the information of transportation costs and whether a facility is opened or not.

Furthermore, a penalty term was included to stimulate the algorithm to search only for spaces where the individuals are valid according to the facility capacity constraint. The penalty is 10 times the exceeding values of each facility. Say facility A was assigned to send 40 more units then it's capable of, then the penalty will be 400.

- Selection: Tournament selection with size 3, that is, 3 individuals will be randomly chosen from the population and compete.

- Crossover: As every individual is a matrix, we can randomly choose its columns to create two new offsprings.

- Mutation: a gaussian noise is added to 33% of the numbers on the matrix. Clipping was added if the number exceeds 1 or gets below 0.