Computational Experience with Heuristics for the p-Median Problem

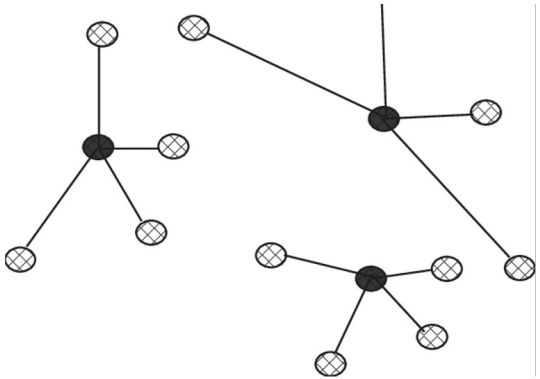
Team D
Marcelo Treviño, Luis F. Medellin, Sebastian Mayorga, Jesús R. Gómez

Selected Topics on Optimization
Roger Z. Rios, Ph.D.

What is the p-Median Problem?

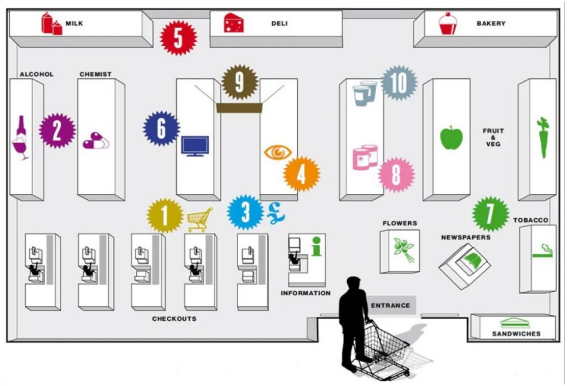
Definition:

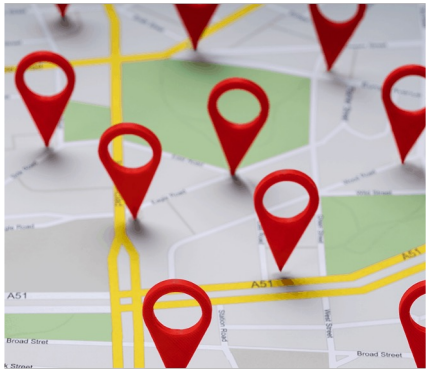
The p-Median Problem is a classic facility location problem where the objective is to choose p facilities (medians) such that the sum of the distances from each demand point to its nearest facility is minimized.



Introduction to the p-Median problem

Applications in real world scenarios:





Problem Description

The p-median problem involves making strategic decisions regarding the location of facilities to efficiently serve a dispersed population. This problem can be defined by four fundamental components:

1. Data

The entry consists of:

- The location of the possible locations of the facilities.

- The spatial distribution of the population or demand points in which these facilities are located

destined to serve.

- The costs or distances associated with the provision of the service from each installation site to each

demand point.

2. Decisions

The decision involves selecting p locations out of the potential facility sites where facilities will be placed.

3. Optimization

The objective is to minimize the total cost or distance required to provide service from the chosen facility locations to all demand points. Mathematically, this can be represented as minimizing the sum of the distances (or costs) between each demand point and its assigned facility, where the assignment is based on proximity.

4. Constraints

The feasible solution is defined by the requirement that each demand point must be assigned to exactly one facility. Additionally, exactly p facilities must be selected from the potential facility sites.

Mathematical formulation

Data/Parameters:

- *n*: Number of demand points.
- *m*: Number of potential facility locations.
- *p*: Number of facilities to be located.
- *dij*: Distance between demand point *i* and potential facility location *j*.

Decision variables:

- *xj*: Binary variable, *xj* =1 if a facility is located at site *j*, otherwise *xj* = 0.
- *yij*: Binary variable, *yij* = 1 if demand point *i* is assigned to facility at site *j*, otherwise *yij* = 0.

Mathematical formulation

Objective:

- The objective is to minimize the total distance between demand points and their assigned facilities.

Constraints:

- Each demand point must be assigned to exactly one facility.
- A facility can only serve a demand point if it is located there.
- Exactly *p* facilities must be located.
- Decision variables must be binary

Constructive Heuristic

- The Genetic Algorithm for the p-Median problem employs principles of natural evolution, such as selection, crossover, and mutation, to evolve a population of solutions over multiple generations. Initially, a population of random solutions is generated, each representing a set of p facilities.
- The fitness of each solution is evaluated based on the total distance to the demand points.

This process iterates over several generations, with the goal of converging to an optimal or near-optimal solution. While more computationally intensive than the Greedy Heuristic, the Genetic Algorithm can explore a broader solution space and potentially find better solutions.

Example of the p-Median Problem

Suppose we have a set of 6 potential locations where we can place facilities and a set of 6 demand points. The goal is to choose 2 facility locations (p = 2) such that the sum of distances from each demand point to the nearest facility is minimized

- Possible clinic locations (facilities): A, B, C, D, E, F

- Demand points (households): 1, 2, 3, 4, 5, 6

- Distance matrix (in kilometers) between the possible locations of the clinics and the demand points:

	A	B	C	D	E	F
1	4	3	1	4	6	8
2	2	6	5	2	4	3
3	5	2	3	3	2	1
4	4	5	4	3	3	2
5	6	4	3	2	1	3
6	8	3	2	1	4	5

Initialization

1. Population size: 4
2. Chromosomes: Each chromosome represents a set of facility locations.
3. Initial population (randomly selected sets of 4 facilities out of A-F):
    1. Chromosome 1: {A, B, C, D}
    2. Chromosome 2: {A, C, F, D}
    3. Chromosome 3: {B, D, C, E}
    4. Chromosome 4: {C, E, A, F}

	A	B	C	D	E	F
1	4	3	1	4	6	8
2	2	6	5	2	4	3
3	5	2	3	3	2	1
4	4	5	4	3	3	2
5	6	4	3	2	1	3
6	8	3	2	1	4	5

Fitness and Crossover

1. Chromosome 1: {A, B, C, D} = 11 km
2. Chromosome 2: {A, C, F, D} = 9 km
3. Chromosome 3: {B, D, C, E} = 10 km
4. Chromosome 4: {C, E, A, F} = 9 km

Let's apply crossover and mutation to produce new offspring.

- Crossover between Chromosome 2 ({A, C, F, D}) and Chromosome 4 ({C, E, A, F}):
  - Possible offspring after crossover: {A, C, F, E} and {C, F, D, A}
- Mutation (randomly changing one gene in the offspring):

	A	B	C	D	E	F
1	4	3	1	4	6	8
2	2	6	5	2	4	3
3	5	2	3	3	2	1
4	4	5	4	3	3	2
5	6	4	3	2	1	3
6	8	3	2	1	4	5

Pseudocode

```
GeneticAlgorithm_pMedian(n, m, p, distMatrix, popSize=50, gens=100, mutRate=0.1):
    pop <- initPopulation(m, p, popSize)
    bestSol <- selectBest(pop, distMatrix, n)
    bestFit <- calcFitness(bestSol, distMatrix, n)
    For gen from 1 to gens:
        newPop <- []
        For i from 1 to (popSize / 2):
            parents <- selectParents(pop)
            child1, child2 <- crossover(parents[0], parents[1], p)
            newPop <- newPop + [mutate(child1, m, p, mutRate), mutate(child2, m, p, mutRate)]
        pop <- newPop
        currentBest <- selectBest(pop, distMatrix, n)
        If calcFitness(currentBest, distMatrix, n) < bestFit:
            bestSol, bestFit <- currentBest, calcFitness(currentBest, distMatrix, n)
    Return bestSol, bestFit
initPopulation(m, p, popSize):
    Return [selectRandom(p, m) for i in 1 to popSize]
selectRandom(p, m):
    Return p unique locations from 0 to m-1
calcFitness(sol, distMatrix, n):
    Return sum(min(distMatrix[i][j] for j in sol) for i in 0 to n-1)
selectBest(pop, distMatrix, n):
    Return min(pop, key=lambda sol: calcFitness(sol, distMatrix, n))
selectParents(pop):
    Return 2 random solutions from pop
crossover(parent1, parent2, p):
    point <- randomNumber(1, p-1)
    Return parent1[:point] + [g for g in parent2 if g not in parent1[:point]], parent2[:point] + [g for g in parent1 if g not in parent2[:point]]
mutate(sol, m, p, mutRate):
    If randomNumber(0, 1) < mutRate:
        sol[randomNumber(0, p-1)] <- randomNumber(0, m-1)
    Return sol
```

Experiment 1 (Without Local Search)

Instance	Size of "n"	Size of "m"	Size of "p"	Genetic Algoritm Heuristic	Time
Data1	1000	10	3	7903.4	35.34s
Data2	1000	10	3	7601.9	34.84s
Data3	1000	10	3	7207.1	35.34s
Data4	1000	10	3	8072.1	33.40s
Data5	1000	10	3	7605.5	34.50s
Data6	1000	10	3	8166.1	33.82s
Data7	1000	10	3	7451.7	37.34s
Data8	1000	10	3	7812.6	34.49s
Data9	1000	10	3	7351.0	33.07s
Data10	1000	10	3	7236.9	34.01s
Data11	1000	10	3	7672.1	33.19s
Data12	1000	10	3	7208.3	33.66s
Data13	1000	10	3	8779.8	33.63s
Data14	1000	10	3	7588.8	34.00s
Data15	1000	10	3	7662.5	34.07s
Data16	1000	10	3	7398.3	33.83s
Data17	1000	10	3	8543.4	33.24s
Data18	1000	10	3	7484.4	33.69s
Data19	1000	10	3	7800.2	33.75s
Data20	1000	10	3	8039.3	31.17s

Experiment 2 (Without Local Search)

Instance	Size of "n"	Size of "m"	Size of "p"	Genetic Algoritm Heuristic	Time S
Data1	10 000	25	5	116121.4	161.20s
Data2	10 000	25	5	121180.7	163.64s
Data3	10 000	25	5	104109.8	173.08s
Data4	10 000	25	5	97387.5	158.22s
Data5	10 000	25	5	100827.7	167.77s
Data6	10 000	25	5	112131.8	176.98s
Data7	10 000	25	5	102062.3	165.75s
Data8	10 000	25	5	112476.4	175.30s
Data9	10 000	25	5	109332.4	158.51s
Data10	10 000	25	5	104273.0	182.33s
Data11	10 000	25	5	102693.9	176.25s
Data12	10 000	25	5	125373.3	189.28s
Data13	10 000	25	5	107250.2	153.39s
Data14	10 000	25	5	118078.3	183.43s
Data15	10 000	25	5	126643.8	186.03s
Data16	10 000	25	5	130483.2	170.55s
Data17	10 000	25	5	106921.5	169.89s
Data18	10 000	25	5	111971.0	30.573s
Data19	10 000	25	5	104941.1	147.60s
Data20	10 000	25	5	104947.9	190.87s

Experiment 3 (Without Local Search)

Instance	Size of "n"	Size of "m"	Size of "p"	Genetic Algoritm Heuristic	Time S
Data1	20 000	50	10	230504.4	279.96s
Data2	20 000	50	10	231840.9	231.16s
Data3	20 000	50	10	228598.7	238.02s
Data4	20 000	50	10	248845.8	254.57s
Data5	20 000	50	10	219550.5	235.21s
Data6	20 000	50	10	223935.4	260.71s
Data7	20 000	50	10	199164.8	283.47s
Data8	20 000	50	10	217348.5	245.76s
Data9	20 000	50	10	250451.9	229.12s
Data10	20 000	50	10	212493.1	250.24s
Data11	20 000	50	10	203685.4	246.56s
Data12	20 000	50	10	205393.7	241.94s
Data13	20 000	50	10	200296.8	251.51s
Data14	20 000	50	10	221142.6	259.92s
Data15	20 000	50	10	227078.5	252.70s
Data16	20 000	50	10	223813.7	253.79s
Data17	20 000	50	10	227050.2	249.230s
Data18	20 000	50	10	256901.0	239.13s
Data19	20 000	50	10	206865.8	218.44s
Data20	20 000	50	10	215691.0	257.43s

Local Search Heuristic

Local search can be used in conjunction with the Genetic Algorithm to improve the solutions generated during the evolutionary process. This hybrid approach, often referred to as a memetic algorithm, combines the global search capability of the Genetic Algorithm with the fine-tuning ability of local search to enhance solution quality.

The **move** used by the local search is to iterate through each of the solution's indexes and change each one of the facility points to another feasible facility (not already in used in the solution), and close the original one.

In our case, the local search is set to iterate a total of 100 times, and keep the best solution found in those iterations.

Experiment 1 (With Local Search)

Instance	Size of "n"	Size of "n"	Size of "m"	Genetic Algoritm Heuristic	Time S	Local Search	Time S	% Improve
Data1	1000	10	3	7903.4	35.34s	7432.9	0.06s	5.95%
Data2	1000	10	3	7601.9	34.84s	7217.7	0.47s	5.05%
Data3	1000	10	3	7207.1	35.34s	7169.5	0.062s	0.52%
Data4	1000	10	3	8072.1	33.40s	7743.5	0.18s	4.07%
Data5	1000	10	3	7605.5	34.50s	7328.6	0.26s	3.64%
Data6	1000	10	3	8166.1	33.82s	8001.7	0.20s	2.01%
Data7	1000	10	3	7451.7	37.34s	7300.2	0.51s	2.03%
Data8	1000	10	3	7812.6	34.49s	7675.6	0.26s	1.75%
Data9	1000	10	3	7351.0	33.07s	7217.7	0.14s	1.81%
Data10	1000	10	3	7236.9	34.01s	7080.3	0.25s	2.16%
Data11	1000	10	3	7672.1	33.19s	7487.2	0.16s	2.41%
Data12	1000	10	3	7208.3	33.66s	6972.9	0.23s	3.27%
Data13	1000	10	3	8779.8	33.63s	8563.8	0.19s	2.46%
Data14	1000	10	3	7588.8	34.00s	7270.6	0.24s	4.19%
Data15	1000	10	3	7662.5	34.07s	7440.9	0.28s	2.89%
Data16	1000	10	3	7398.3	33.83s	7292.6	0.35s	1.43%
Data17	1000	10	3	8543.4	33.24s	8076.9	0.16s	5.46%
Data18	1000	10	3	7484.4	33.69s	7305.4	0.15s	2.39%
Data19	1000	10	3	7800.2	33.75s	7643.7	0.27s	2.01%
Data20	1000	10	3	8039.3	31.17s	7423.2	0.39s	7.66%
							Avarage	**3.16%**

Experiment 2 (With Local Search)

Instance	Size of "n"	Size of "n"	Size of "m"	Genetic Algoritm Heuristic	Time S	Local Search	Time S	% Improve
Data1	10 000	25	5	116121.4	161.20s	113840.7969	1.71s	1.96%
Data2	10 000	25	5	121180.7	163.64s	114845.1753	0.74s	5.23%
Data3	10 000	25	5	104109.8	173.08s	103996.1201	0.62s	0.11%
Data4	10 000	25	5	97387.5	158.22s	97046.39614	0.51s	0.35%
Data5	10 000	25	5	100827.7	167.77s	97508.52762	1.62s	3.29%
Data6	10 000	25	5	112131.8	176.98s	107481.1094	1.14s	4.15%
Data7	10 000	25	5	102062.3	165.75s	95290.90191	0.60s	6.63%
Data8	10 000	25	5	112476.4	175.30s	111048.1283	0.7s	1.27%
Data9	10 000	25	5	109332.4	158.51s	107316.319	1.11s	1.84%
Data10	10 000	25	5	104273.0	182.33s	102767.2235	1.31s	1.44%
Data11	10 000	25	5	102693.9	176.25s	96259.09259	0.98s	6.27%
Data12	10 000	25	5	125373.3	189.28s	125373.3227	0.05s	0.00%
Data13	10 000	25	5	107250.2	153.39s	104788.4007	4.60S	2.30%
Data14	10 000	25	5	118078.3	183.43s	116671.5507	0.40s	1.19%
Data15	10 000	25	5	126643.8	186.03s	126643.7816	0.40s	0.00%
Data16	10 000	25	5	130483.2	170.55s	129491.1095	0.83s	0.76%
Data17	10 000	25	5	106921.5	169.89s	104384.1236	0.44s	2.37%
Data18	10 000	25	5	111971.0	30.573s	111697.0195	33.35s	0.24%
Data19	10 000	25	5	104941.1	147.60s	102998.6932	2.76s	1.85%
Data20	10 000	25	5	104947.9	190.87s	104090.0588	0.27s	0.82%
							Avarage	**2.10%**

Experiment 3 (With Local Search)

Instance	Size of "n"	Size of "n"	Size of "m"	Genetic Algoritm Heuristic	Time S	Local Search	Time S	% Improve
Data1	20 000	50	10	230504.4	279.96s	228369.0	0.749s	0.93%
Data2	20 000	50	10	231840.9	231.16s	212154.7	0.82s	8.49%
Data3	20 000	50	10	228598.7	238.02s	212984.9	0.65s	6.83%
Data4	20 000	50	10	248845.8	254.57s	240957.9	1.33s	3.17%
Data5	20 000	50	10	219550.5	235.21s	211147.2	0.85s	3.83%
Data6	20 000	50	10	223935.4	260.71s	222942.9	0.98s	0.44%
Data7	20 000	50	10	199164.8	283.47s	198282.6	1.03s	0.44%
Data8	20 000	50	10	217348.5	245.76s	207469.6	0.51s	4.55%
Data9	20 000	50	10	250451.9	229.12s	241901.1	4.95s	3.41%
Data10	20 000	50	10	212493.1	250.24s	200871.6	1.78s	5.47%
Data11	20 000	50	10	203685.4	246.56s	200207.4	1.27s	1.71%
Data12	20 000	50	10	205393.7	241.94s	194296.1	0.68s	5.40%
Data13	20 000	50	10	200296.8	251.51s	196449.3	1.79s	1.92%
Data14	20 000	50	10	221142.6	259.92s	215806.9	1.36s	2.41%
Data15	20 000	50	10	227078.5	252.70s	226958.0	0.55s	0.05%
Data16	20 000	50	10	223813.7	253.79s	214820.1	1.190s	4.02%
Data17	20 000	50	10	227050.2	249.230s	225961.5	0.764s	0.48%
Data18	20 000	50	10	256901.0	239.13s	239232.5	1.34s	6.88%
Data19	20 000	50	10	206865.8	218.44s	188734.2	1.47s	8.76%
Data20	20 000	50	10	215691.0	257.43s	208860.4	0.86s	3.17%
							Avarage	**3.62%**

Experiment 3 (With Local Search)

Key takeaways from our exploration of the p-Median problem include:

1. **Versatility and Practicality**: The p-Median problem's versatility makes it applicable to diverse fields, from strategic placement of public amenities and emergency services in urban planning to optimizing distribution networks in logistics and ensuring equitable access to healthcare facilities.
2. **Mathematical Rigor and Computational Techniques**: The problem can be rigorously formulated as an integer linear programming (ILP) model, enabling precise mathematical solutions. However, due to the computational complexity of solving ILP models for large instances, heuristic and metaheuristic approaches, such as Genetic Algorithms and Local Search, play a crucial role in finding near-optimal solutions efficiently.
3. **Hybrid Heuristic Approaches**: Combining global search techniques like Genetic Algorithms with local optimization methods, such as Local Search, can significantly enhance the solution quality and convergence speed. This hybrid approach leverages the strengths of both methods, balancing exploration and exploitation to achieve superior outcomes.
4. **Impact on Decision-Making**: The insights derived from solving the p-Median problem inform critical decision-making processes in various sectors. For instance, in urban planning, it aids in alleviating congestion and improving accessibility, while in healthcare, it ensures better resource allocation and patient outcomes.