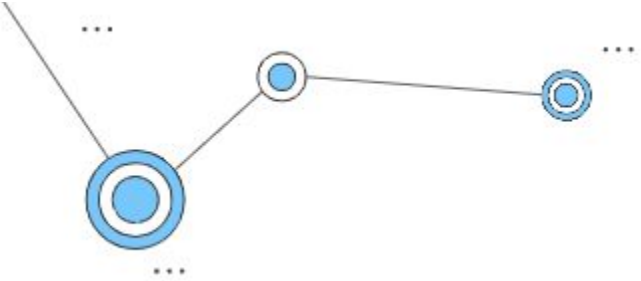
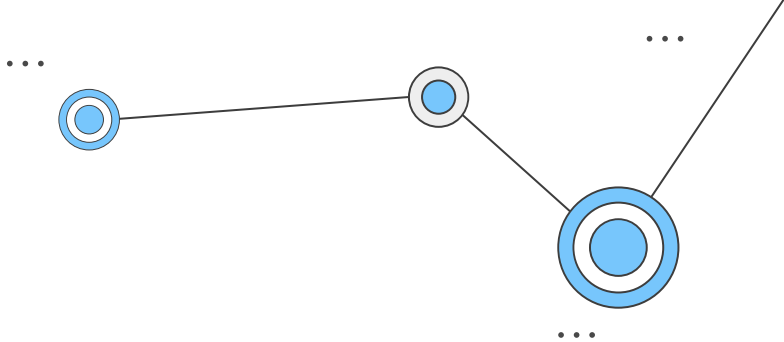
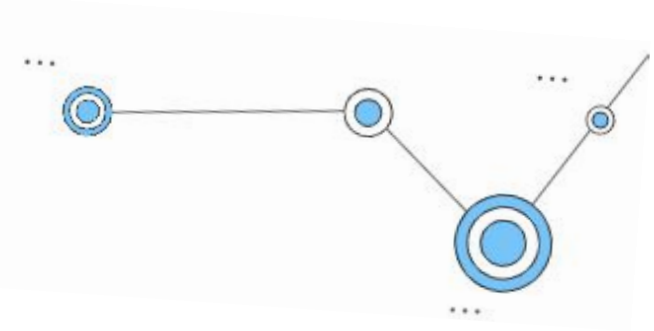


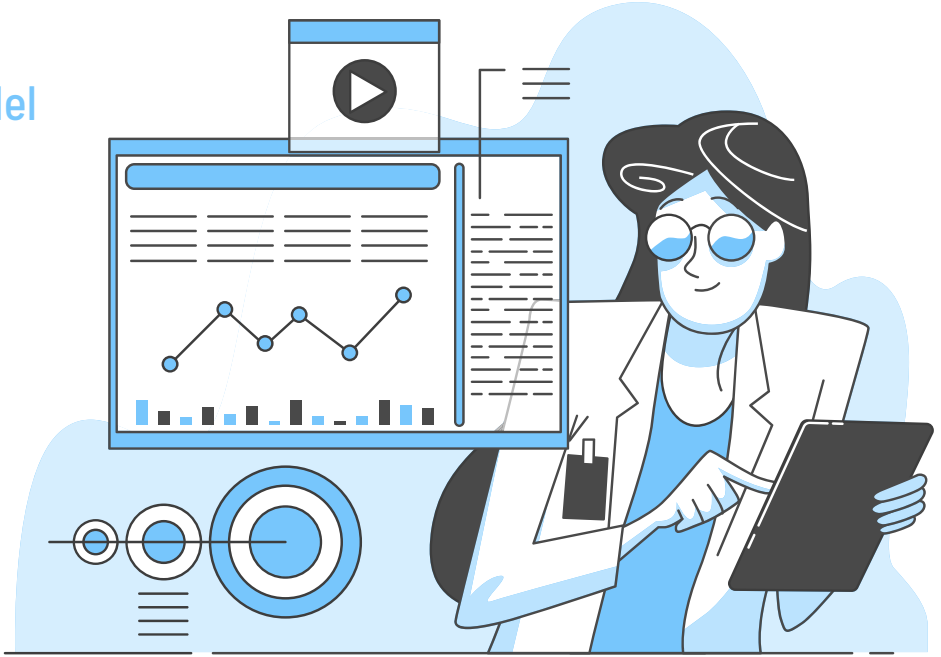


Maximum Covering Problem



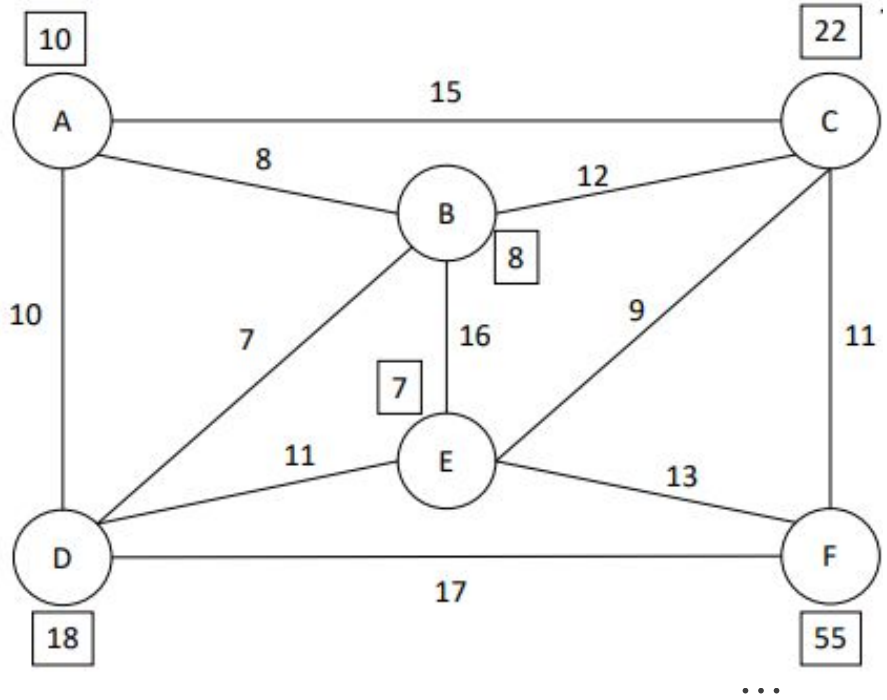
Jiaxue Li
Denisse Lopez

Table of Contents



01
Problem Presentation

Max Covering Problems



What we have:

- **6** Nodes
- **6** number of demand
- **11** Routes
- **11** Possible Coverage distance

The objective:

By fixed number of facilities to be located (as an input), maximize the number of demand covered

02

Linear Programming Model

1

Declare the Model for Max Covering

```
!pip install pulp
```

```
Requirement already satisfied: pulp in c:\users\cheri\anaconda3\lib\site-packages (2.
6.0)
```

```
from pulp import *
```

```
#declare the model
model = LpProblem('Maximum', LpMaximize)
```

```
#make the list of nodes
zones = ['A', 'B', 'C', 'D', 'E', 'F']
zones2 = ['A', 'B', 'C', 'D', 'E', 'F']
P = 4
K = 11
#demand of each nodes
demand = [10,8,22,18,7,55]
zone_demand = dict(zip(zones, demand))
zone_demand
```

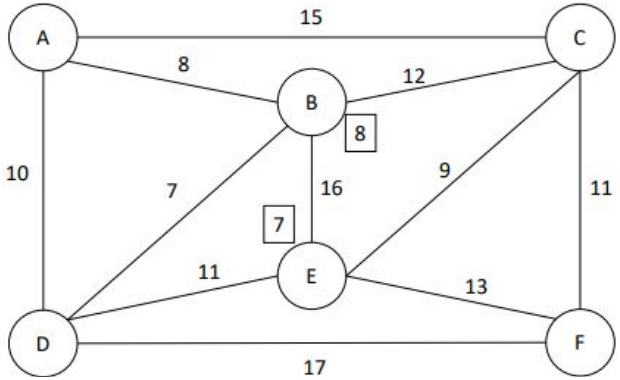
```
{'A': 10, 'B': 8, 'C': 22, 'D': 18, 'E': 7, 'F': 55}
```

Inputs/Parameters:

- **Nodes A,B,C,D,E,F**
- **4 Number of facilities to locate**
- **11 Possible Coverage distance**
- **Demand of each nodes**
- **Route distance of each (Next Page)**

2

Route distance of each




```
# keys / index for decision variables / parameters
keys = [(1,12) for 1 in zones for 12 in zones2]
length = [0, 8, 15, 10, 1000, 1000,
          8, 0, 12, 7, 16, 1000,
          15, 12, 0, 1000, 9, 11,
          10, 7, 1000, 0, 11, 17,
          1000, 16, 9, 11, 0, 13,
          1000, 1000, 11, 17, 13, 0]
keys
distance = dict(zip(keys, length))
distance
```

```
{('A', 'A'): 0,
 ('A', 'B'): 8,
 ('A', 'C'): 15,
 ('A', 'D'): 10,
 ('A', 'E'): 1000,
 ('A', 'F'): 1000,
 ('B', 'A'): 8,
 ('B', 'B'): 0,
 ('B', 'C'): 12,
 ('B', 'D'): 7,
 ('B', 'E'): 16,
 ('B', 'F'): 1000,
 ('C', 'A'): 15,
 ('C', 'B'): 12,
 ('C', 'C'): 0,
 ('C', 'D'): 1000,
 ('C', 'E'): 9,
 ('C', 'F'): 11,
 ('D', 'A'): 10,
 ('D', 'B'): 7,
 ('D', 'C'): 1000,
 ('D', 'D'): 0,
 ('D', 'E'): 11,
 ('D', 'F'): 17,
 ('E', 'A'): 1000,
 ('E', 'B'): 16,
 ('E', 'C'): 9,
 ('E', 'D'): 11,
 ('E', 'E'): 0,
 ('E', 'F'): 13,
 ('F', 'A'): 1000,
 ('F', 'B'): 1000,
 ('F', 'C'): 11,
 ('F', 'D'): 17,
 ('F', 'E'): 13,
 ('F', 'F'): 0}
```

3

Declare the decision variables

```
# declare the decision variables
#(name, keys, bounds (0, inf), cat = integer, continuous, binary)
k = LpVariable.dicts('counted', keys, cat = 'Binary')
x = LpVariable.dicts('facility_location', zones, cat = 'Binary')
z = LpVariable.dicts('covered', zones, cat = 'Binary')
```

Decision variables:

- **X =>**
  1 if a facility is located at candidate site *l* ∈ *zones*
  0, otherwise

- **Z =>**
  1 if node *l2* ∈ *zones is covered*
  0, otherwise

3

Declare the decision variables

```
# declare the decision variables
#(name, keys, bounds (0, inf), cat = integer, continuous, binary)
k = LpVariable.dicts('counted', keys, cat = 'Binary')
x = LpVariable.dicts('facility_location', zones, cat = 'Binary')
z = LpVariable.dicts('covered', zones, cat = 'Binary')
```

Decision variables:

- **K =>**
  1 if a candidate site *l ∈ zones has already been considered/counted*
  0, otherwise

MAXIMIZE

(Covered Demands) $10Z_A + 8Z_B + 22Z_C + 18Z_D + 7Z_E + 55Z_F$

SUBJECT TO

(Node A coverage)	X_A	$+X_B$		$+X_D$			$\geq Z_A$
(Node B coverage)	X_A	$+X_B$		$+X_D$			$\geq Z_B$
(Node C coverage)			$+X_C$		$+X_E$	$+X_F$	$\geq Z_C$
(Node D coverage)	X_A	$+X_B$		$+X_D$	$+X_E$		$\geq Z_B$
(Node E coverage)			X_C	$+X_D$	$+X_E$		$\geq Z_E$
(Node F coverage)			X_C			$+X_F$	$\geq Z_F$
(No. to locate)	X_A	$+X_B$	$+X_C$	$+X_D$	$+X_E$	$+X_F$	≤ 1
Integrality	$X_A,$	$X_B,$	$X_C,$	$X_D,$	$X_E,$	X_F	$\in \{0,1\}$
	$Z_A,$	$Z_B,$	$Z_C,$	$Z_D,$	$Z_E,$	Z_F	$\in \{0,1\}$

3

Declare the decision variables

```
# declare the decision variables
#(name, keys, bounds (0, inf), cat = integer, continuous, binary)
k = LpVariable.dicts('counted', keys, cat = 'Binary')
x = LpVariable.dicts('facility_location', zones, cat = 'Binary')
z = LpVariable.dicts('covered', zones, cat = 'Binary')
```

Decision variables:

- **K =>**
  1 if a candidate site *l ∈ zones has already been considered/counted*
  0, otherwise

```
for l in zones:
    for l2 in zones2:
        if distance [(l,l2)] <=11:
            k[(l,l2)] = 1
        else:
            k[(l,l2)] = 0
```

3

Declare the decision variables

Decision variables:
- K =>

Decision variables:
- X =>

Decision variables:
- Z =>

```
{('A', 'A'): counted_('A',_'A'),
 ('A', 'B'): counted_('A',_'B'),
 ('A', 'C'): counted_('A',_'C'),
 ('A', 'D'): counted_('A',_'D'),
 ('A', 'E'): counted_('A',_'E'),
 ('A', 'F'): counted_('A',_'F'),
 ('B', 'A'): counted_('B',_'A'),
 ('B', 'B'): counted_('B',_'B'),
 ('B', 'C'): counted_('B',_'C'),
 ('B', 'D'): counted_('B',_'D'),
 ('B', 'E'): counted_('B',_'E'),
 ('B', 'F'): counted_('B',_'F'),
 ('C', 'A'): counted_('C',_'A'),
 ('C', 'B'): counted_('C',_'B'),
 ('C', 'C'): counted_('C',_'C'),
 ('C', 'D'): counted_('C',_'D'),
 ('C', 'E'): counted_('C',_'E'),
 ('C', 'F'): counted_('C',_'F'),
 ('D', 'A'): counted_('D',_'A'),
 ('D', 'B'): counted_('D',_'B'),
 ('D', 'C'): counted_('D',_'C'),
 ('D', 'D'): counted_('D',_'D'),
 ('D', 'E'): counted_('D',_'E'),
 ('D', 'F'): counted_('D',_'F'),
 ('E', 'A'): counted_('E',_'A'),
 ('E', 'B'): counted_('E',_'B'),
 ('E', 'C'): counted_('E',_'C'),
 ('E', 'D'): counted_('E',_'D'),
 ('E', 'E'): counted_('E',_'E'),
 ('E', 'F'): counted_('E',_'F'),
 ('F', 'A'): counted_('F',_'A'),
 ('F', 'B'): counted_('F',_'B'),
 ('F', 'C'): counted_('F',_'C'),
 ('F', 'D'): counted_('F',_'D'),
 ('F', 'E'): counted_('F',_'E'),
 ('F', 'F'): counted_('F',_'F')}
```

```
{'A': facility_location_A,
 'B': facility_location_B,
 'C': facility_location_C,
 'D': facility_location_D,
 'E': facility_location_E,
 'F': facility_location_F}
```

```
{'A': covered_A,
 'B': covered_B,
 'C': covered_C,
 'D': covered_D,
 'E': covered_E,
 'F': covered_F}
```

4

Objective Function

$$\max \sum_{i \in I} h_i Z_i$$

Maximize the number of demands covered

```
# objective function
model += lpSum(zone_demand[(l)]*z[(l)] for l in zones)
```

MAXIMIZE
(Covered Demands)

$$10Z_A + 8Z_B + 22Z_C + 18Z_D + 7Z_E + 55Z_F$$

5

Constraints

$$Z_i \leq \sum_{j \in J} a_{ij} X_j \, , \forall i \in I$$

Demand at node $i \in I$ can't be covered unless at least one of the facility sites that cover the node is selected

```
#constraint 1
for 12 in zones2:
    model += lpSum(x[(1)]*k[(1,12)]for 1 in zones) >= z[(1)]
```

(Node A coverage)	X_A	$+X_B$		$+X_D$			$\geq Z_A$
(Node B coverage)	X_A	$+X_B$		$+X_D$			$\geq Z_B$
(Node C coverage)			$+X_C$		$+X_E$	$+X_F$	$\geq Z_C$
(Node D coverage)	X_A	$+X_B$		$+X_D$	$+X_E$		$\geq Z_B$
(Node E coverage)			X_C	$+X_D$	$+X_E$		$\geq Z_E$
(Node F coverage)			X_C			$+X_F$	$\geq Z_F$

5

Constraints

$$\sum_{j \in J} X_j \leq P$$

We locate no more than *P* facilities

```
# constraint 2
model += lpSum(x[(l)] for l in zones) <= P
```

$$X_A + X_B + X_C + X_D + X_E + X_F \leq 4$$

6

Solve the model

```
model.solve()
```

```
1
```

```
value(model.objective)
```

```
120.0
```

demand = [10, 8, 22, 18, 7, 55]

10+8+22+18+7+55=120

```
# get the solution
for i in z:
    print('{} covered {}'.format(i, z[i].varValue))
```

```
A covered 1.0
B covered 1.0
C covered 1.0
D covered 1.0
E covered 1.0
F covered 1.0
```

```
# get the solution
for i in x:
    print('{} located {}'.format(i, x[i].varValue))
```

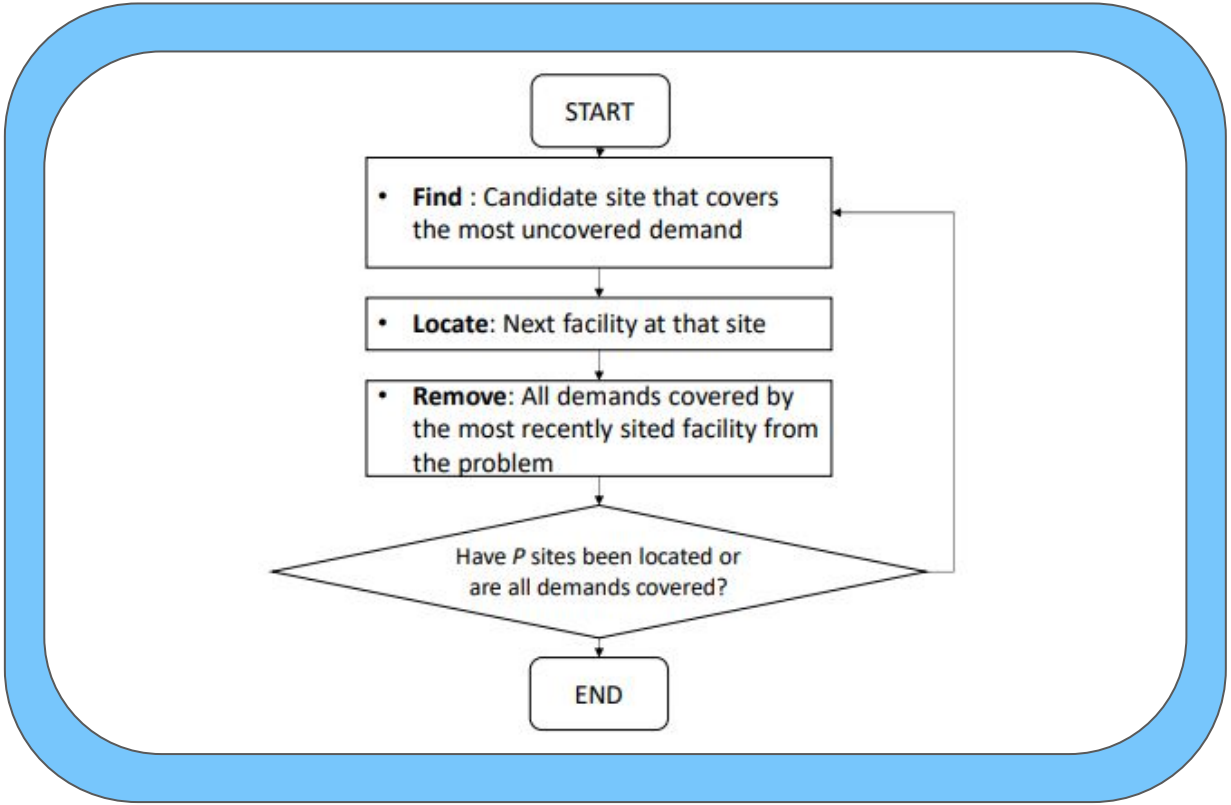
```
A located 1.0
B located 0.0
C located 1.0
D located 0.0
E located 0.0
F located 0.0
```

03

Using Greedy Algorithm

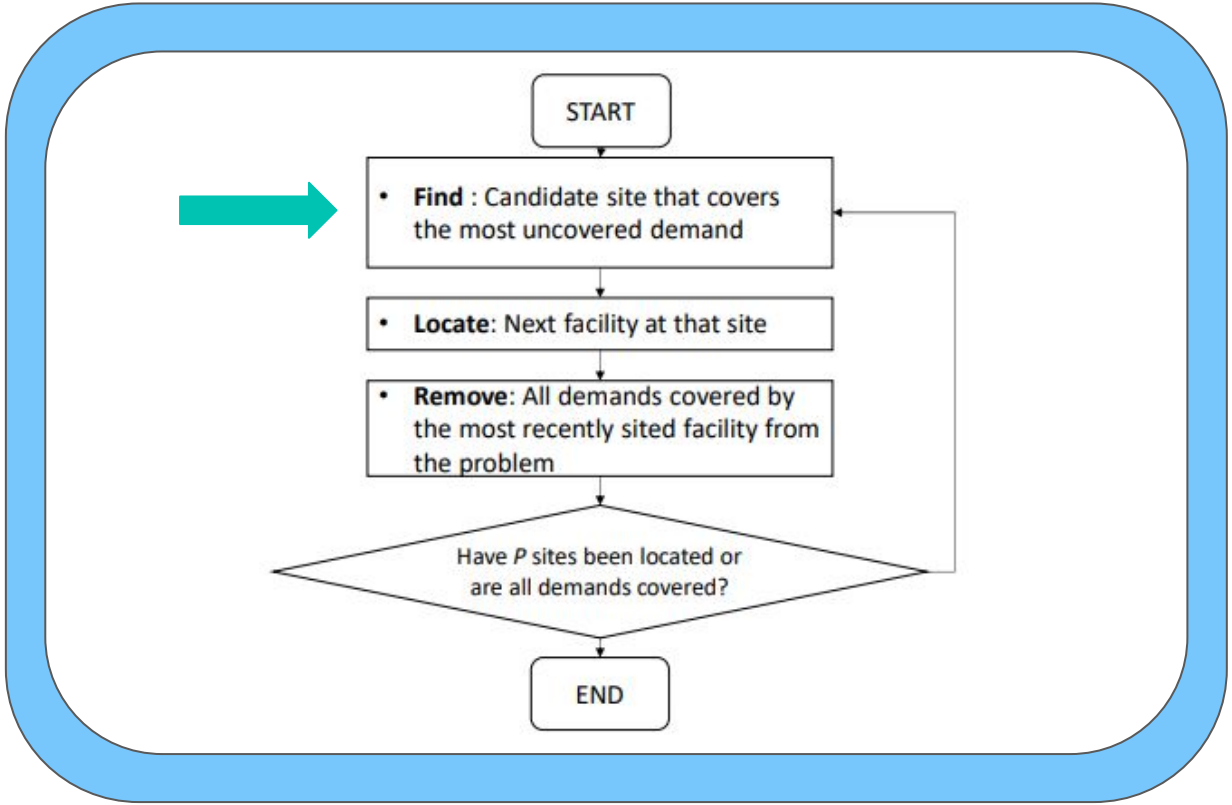
1

Greedy Adding Algorithm for Max Covering



1

Greedy Adding Algorithm for Max Covering



2

Find Candidates Sites Covering Demand

Candidate Site	Nodes Covered	Demand Covered
A	A,B,D	36
B	A,B,D	36
C	C,E,F	84
D	A,B,D,E	43
E	C,D,E	47
F	C,F	77

Input Variables

```
zones = ['A', 'B', 'C', 'D', 'E', 'F']
demand= [10,   8,   22,  18,  7,   55]

routes = [
    [1,1,0,1,0,0],
    [1,1,0,1,0,0],
    [0,0,1,0,1,1],
    [1,1,0,1,1,0],
    [0,0,1,1,1,0],
    [0,0,1,0,0,1]]

total_demand = sum(demand)
# total demand = 120

# number of sites to locate
P = 5
```

4

Find Candidates Sites Covering Demand

```
Demand_covered = []
for i in range(len(demand)):
    value = 0
    for j in range(len(routes[i])):
        value += routes[i][j] *demand[j]
    Demand_covered.append(value)

Demand_covered
```

```
[36, 36, 84, 43, 47, 77]
```

route [i] [j] * demand [j]

```
[[1 1 0 1 0 0]
 [1 1 0 1 0 0]
 [0 0 1 0 1 1]
 [1 1 0 1 1 0]
 [0 0 1 1 1 0]
 [0 0 1 0 0 1]]
```

```
[10, 8, 22, 18, 7, 55]
```

when i = 0 :

value = (1 × 10) + (1 × 8) + (0 × 22) + (1 × 18) + (0 × 7) + (0 × 55)

↳ 36

5

Find : Candidate site that covers the most uncovered demand

Candidate Site	Nodes Covered	Demand Covered
A	A,B,D	36
B	A,B,D	36
C	C,E,F	84
D	A,B,D,E	43
E	C,D,E	47
F	C,F	77

6

Find : Candidate site that covers the most uncovered demand

```
Demand_covered
```

```
[36, 36, 84, 43, 47, 77]
```

```
max_demand_covered = max(Demand_covered)
print ('Maximum Demand Covered', max_demand_covered)
maxindex_demand_covered = Demand_covered.index(max_demand_covered)
print('Index Of The Maximum Demand Covered',maxindex_demand_covered)
```

```
Maximum Demand Covered 84
Index Of The Maximum Demand Covered 2
```

max (Demand_covered)

→ [36, 36, 84, 43, 47, 77]

Demand_covered.index(84)

→ [36, 36, 84, 43, 47, 77]

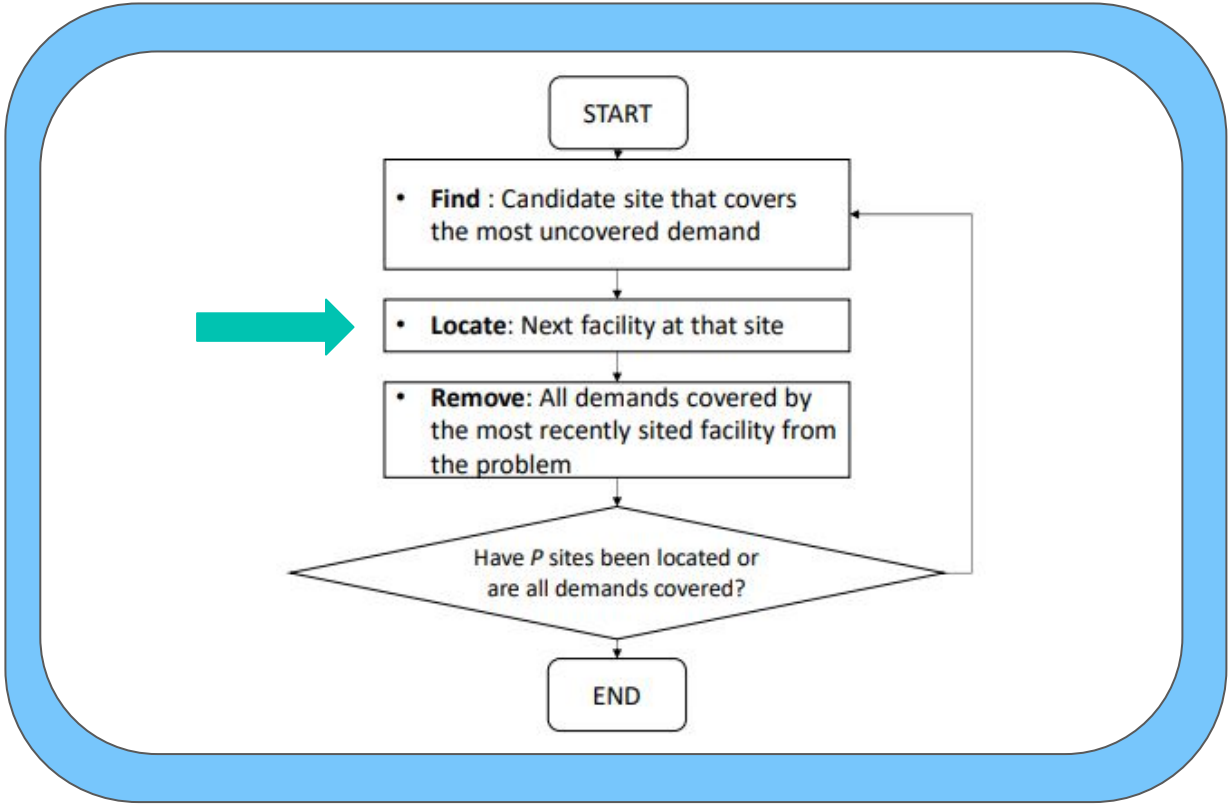
0 , 1. 2. 3 , 4, 5

Save Facility Location & Demand

```
facility_located = []
facility_demand_covered = []

facility_located.append(zones[maxindex_demand_covered])
facility_demand_covered.append(max_demand_covered)

print('Fecility Location', facility_located,
      ', Demand Covered',facility_demand_covered)
```

```
Fecility Location ['C'] , Demand Covered [84]
```

Locate : Next Facility At That Site



Find Demand & Location Covered by Location 'C'

Candidate Site	Nodes Covered	Demand Covered
A	A,B,D	36
B	A,B,D	36
C	C,E,F	84
D	A,B,D,E	43
E	C,D,E	47
F	C,F	77

Find Demand & Location Covered by Location 'C'

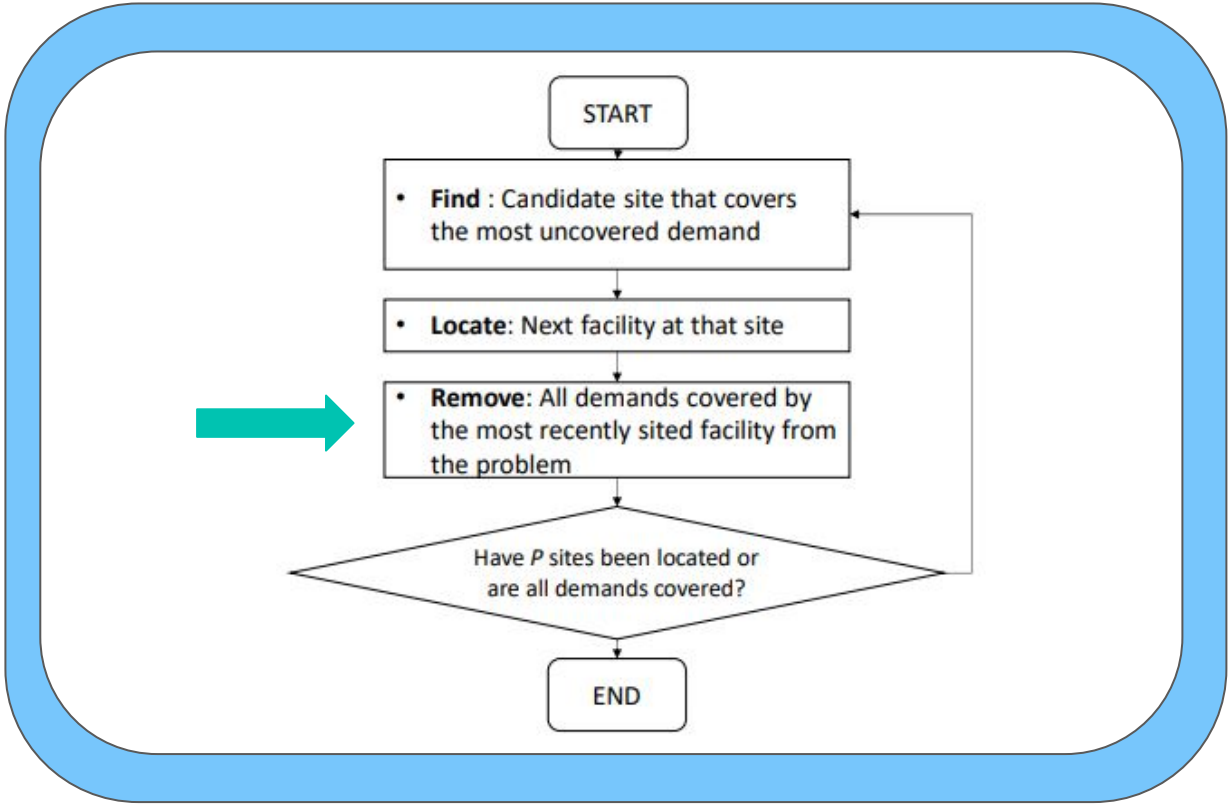
```
# find the nodes covered by the facility located
item = 1
indexes = [i for i, j in enumerate(routes[maxindex_demand_covered]) if j == 1]
indexes
```

```
[2, 4, 5]
```

route [a] [j]

	A	B	C	D	E	F
A	1	1	0	1	0	0
B	1	1	0	1	0	0
C	0	0	1	0	1	1
D	1	1	0	1	1	0
E	0	0	1	1	1	0
F	0	0	1	0	0	1

Remove Demand Covered



Remove Demand Covered

Candidate Site	Nodes Covered	Demand Covered
A	A,B,D	36
B	A,B,D	36
C	-	-
D	A,B,D	36
E	D	18
F	-	-

Remove Demand Covered

for i in range(len(routes))
↳ i → 0, 1, 2, 3, 4, 5

for j in indexes
↳ j → 2, 4, 5

	A	B	C	D	E	F
A	1	1	0	1	0	0
B	1	1	0	1	0	0
C	0	0	1	0	1	1
D	1	1	0	1	1	0
E	0	0	1	1	1	0
F	0	0	1	0	0	1

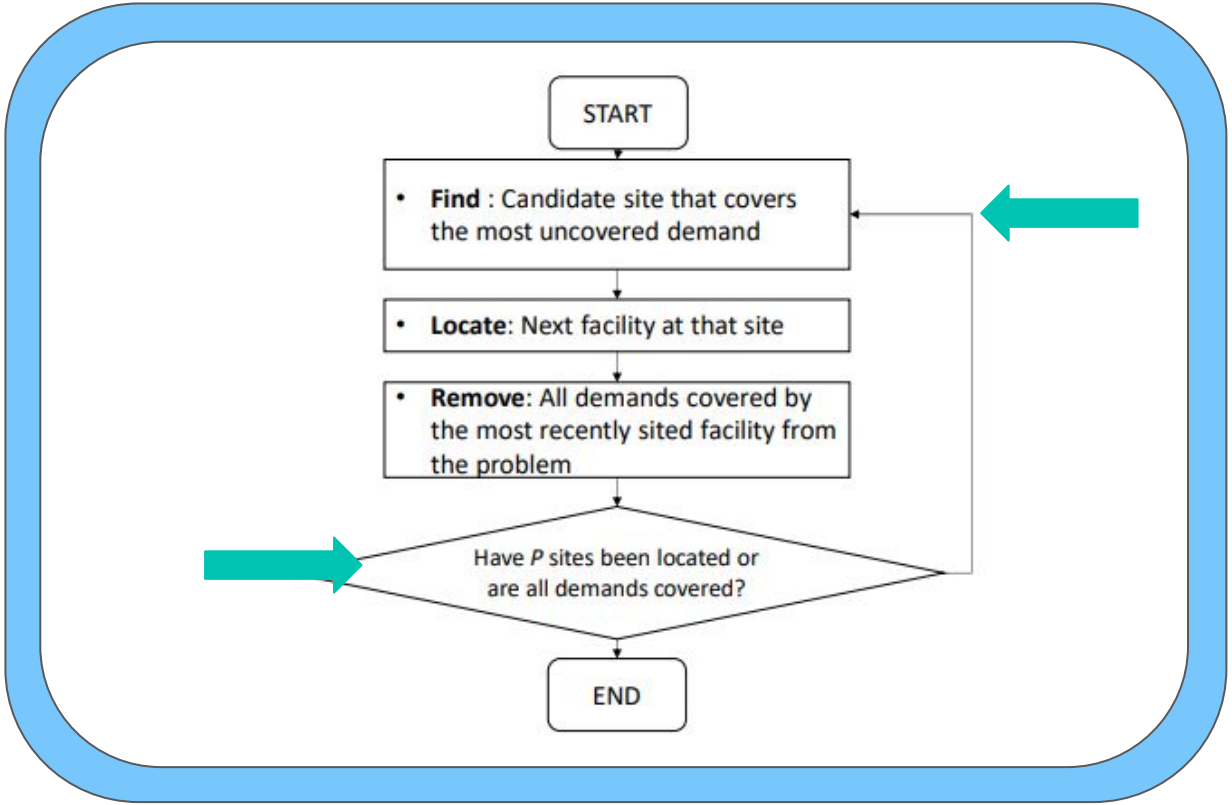
```
# update cover demand
for i in range(len(routes)):
    for j in indexes:
        routes[i][j] = 0

routes
```

```
[[1, 1, 0, 1, 0, 0],
 [1, 1, 0, 1, 0, 0],
 [0, 0, 1, 0, 1, 1],
 [1, 1, 0, 1, 1, 0],
 [0, 0, 1, 1, 1, 0],
 [0, 0, 1, 0, 0, 1]]
```

→

```
[[1, 1, 0, 1, 0, 0],
 [1, 1, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0],
 [1, 1, 0, 1, 0, 0],
 [0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0]]
```

Have P site been located or are all demands covered?



Create a Loop

```
def MaxCovering(zones, routes, demand, P):
    facility_located = []
    facility_demand_covered = []
    for i in range(P):
        if total_demand == sum(facility_demand_covered):
            return facility_located, facility_demand_covered
        else:

            Demand_covered = []
            for i in range(len(demand)):
                value = 0
                for j in range(len(routes[i])):
                    value += routes[i][j] *demand[j]
                Demand_covered.append(value)

            max_demand_covered = max(Demand_covered)
            maxindex_demand_covered = Demand_covered.index(max_demand_covered)

            # Facility located

            facility_located.append(zones[maxindex_demand_covered])
            facility_demand_covered.append(max_demand_covered)

            # find the nodes covered by the facility located
            item = 1
            indexes = [i for i, j in enumerate(routes[maxindex_demand_covered]) if j == item]

            # update cover demand
            for i in range(len(routes)):
                for j in indexes:
                    routes[i][j] = 0
```

Create a Loop

```
def MaxCovering(zones, routes, demand, P):
    facility_located = []
    facility_demand_covered = []
    for i in range(P):
        if total_demand == sum(facility_demand_covered):
            return facility_located, facility_demand_covered
        else:

            Demand_covered = []
            for i in range(len(demand)):
                value = 0
                for j in range(len(routes[i])):
                    value += routes[i][j] *demand[j]
                Demand_covered.append(value)

            max_demand_covered = max(Demand_covered)
            maxindex_demand_covered = Demand_covered.index(max_demand_covered)

            # Facility located

            facility_located.append(zones[maxindex_demand_covered])
            facility_demand_covered.append(max_demand_covered)

            # find the nodes covered by the facility located
            item = 1
            indexes = [i for i, j in enumerate(routes[maxindex_demand_covered]) if j == item]

            # update cover demand
            for i in range(len(routes)):
                for j in indexes:
                    routes[i][j] = 0
```

Result

```
print(facility_located, facility_demand_covered )
```

```
['C', 'A'] [84, 36]
```

04

Our Comparison

Our Comparison

Differences

1. The **LP Model** one requires the objective function, constrains, variables and parameters, while the **Greedy Algorithm** one requires instruction **loops**.
2. LP model need **pulp** to import the model, but Greedy Algorithm don't need to use any other packages.
3. Compared to the long model building, **Greedy Algorithm has a shorter process, and give the priority of the locations.**

Similarity

1. Both of procedures in this scenario **gives the same result,** which showed that implemented facility in location C & A can cover the total demand.
2. Both of procedures in this scenario **stop at the point that all demand were satisfied.**

Thanks for watching