Yu-Heng Chien 132005200

8.2

(a) Optimal locations: 1, 2, 3, 4, 5, 7, 8 and 9

(b) Optimal cost: 1940

8.8

Optimal locations: 1, 2, 5, 6, 7 and 9

8.9

(a) Optimal locations: 1, 3, 5 and 6

(b) Total number of demands covered: 367

8.19

(a) Bethlehem and Springfield

(b)
Akron is from Bethlehem
Albany is from Springfield
Nasuha is from Springfield
Scranton is from Bethlehem
Utica is from Springfield

(c) 16265000

Calculation

In [1]:

```python
import pandas as pd
import gurobipy as gp
from gurobipy import GRB
from itertools import product
```

In [ ]:

```python

```

8.2

In [2]:

```python
# Demand
df_demand = pd.read_excel('10node.xlsx', 'Customer')
df_demand = df_demand['Demand'].tolist()
df_demand
```

Out[2]:

```
[60, 27, 29, 26, 33, 15, 17, 97, 97, 19]
```

In [3]:

```python
# Fixed cost
f = 200
```

In [4]:

```python
# Transportation cost
df_dis = pd.read_excel('10node.xlsx', 'Distance')
df_dis = df_dis.iloc[:,1:]
df_dis
```

Out[4]:

	1	2	3	4	5	6	7	8	9	10
0	0.000000	9.219544	3.000000	7.071068	8.544004	4.472136	6.708204	3.605551	5.099020	5.000000
1	9.219544	0.000000	7.615773	5.000000	4.472136	5.000000	3.162278	5.656854	6.082763	7.071068
2	3.000000	7.615773	0.000000	7.280110	5.830952	4.123106	6.000000	3.162278	2.236068	2.000000
3	7.071068	5.000000	7.280110	0.000000	8.062258	3.162278	2.236068	4.123106	7.211103	8.062258
4	8.544004	4.472136	5.830952	8.062258	0.000000	6.082763	5.830952	6.000000	3.605551	4.242641
5	4.472136	5.000000	4.123106	3.162278	6.082763	0.000000	2.236068	1.000000	4.242641	5.000000
6	6.708204	3.162278	6.000000	2.236068	5.830952	2.236068	0.000000	3.162278	5.385165	6.324555
7	3.605551	5.656854	3.162278	4.123106	6.000000	1.000000	3.162278	0.000000	3.605551	4.242641
8	5.099020	6.082763	2.236068	7.211103	3.605551	4.242641	5.385165	3.605551	0.000000	1.000000
9	5.000000	7.071068	2.000000	8.062258	4.242641	5.000000	6.324555	4.242641	1.000000	0.000000

In [5]:

```python
# transportation cost

shipping_cost = {(customer, facility): df_dis.iloc[customer, facility] * 10
            for customer in range(0, 10)
            for facility in range(0, 10)}

print("Number of viable pairings: {0}".format(len(shipping_cost.keys())))
```

```
Number of viable pairings: 100
```

In [6]:

```python
m = gp.Model("Facility location")
```

```
Restricted license - for non-production use only - expires 2023-10-25
```

In [7]:

```python
# Decision variables: facilities open or close
fact = m.addVars(10, vtype=GRB.BINARY, name='fact')
```

In [8]:

```python
# Decision variables: assign customer clusters to a facility location
cartesian_prod = list(product(range(0, 10), range(0, 10)))
cust = m.addVars(cartesian_prod, lb = 0 ,vtype = GRB.CONTINUOUS, name='cust')
```

In [9]:

```python
# Deploy Objective Function
# Minimize total cost
# obj = gp.quicksum(f * fact[facility] + df_demand[customer] * cust[customer, facility] * shipping_cost[customer, facility] for customer, facility in shipping_cos
m.setObjective(gp.quicksum(df_demand[customer] * shipping_cost[customer, facility] * cust[customer, facility] for customer in range(0, 10) for facility in range(

# 1.
m.addConstrs((gp.quicksum(cust[(customer, facility)] for facility in range(0, 10)) == 1 for customer in range(0, 10)), name='Demand')

# 2.
m.addConstrs((cust[customer, facility] <= fact[facility] for customer, facility in cartesian_prod), name='Setup2ship')

m.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 110 rows, 110 columns and 300 nonzeros
Model fingerprint: 0x5f8380a0
Variable types: 100 continuous, 10 integer (10 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+02, 7e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Presolve time: 0.00s
Presolved: 110 rows, 110 columns, 300 nonzeros
Variable types: 100 continuous, 10 integer (10 binary)
Found heuristic solution: objective 2000.0000000

Root relaxation: objective 1.940000e+03, 10 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0    1940.0000000 1940.00000  0.00%     -    0s

Explored 1 nodes (10 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 2: 1940 2000

Optimal solution found (tolerance 1.00e-04)
Best objective 1.940000000000e+03, best bound 1.940000000000e+03, gap 0.0000%
```

In [10]:

```python
# display optimal values of decision variables

for facility in fact.keys():
    if (abs(fact[facility].x) > 1e-6):
        print(f"\n Build a factory at location {facility + 1}.")
```

```
 Build a factory at location 1.

 Build a factory at location 2.

 Build a factory at location 3.

 Build a factory at location 4.

 Build a factory at location 5.

 Build a factory at location 7.

 Build a factory at location 8.

 Build a factory at location 9.
```

8.8

In [11]:

```python
# Parameter
# Fixed cost
f = 1
# distance
c = 2.5
```

In [12]:

```python
# aij
aij = {(customer, facility): 1 if df_dis.iloc[customer, facility] <= c else 0
            for customer in range(0, 10)
            for facility in range(0, 10)}

print("Number of viable pairings: {0}".format(len(aij.keys())))
```

```
Number of viable pairings: 100
```

In [13]:

```python
m = gp.Model("8.8 SCLP")
```

In [14]:

```python
# Decision variables: facilities open or close
fact = m.addVars(10, vtype = GRB.BINARY, name='fact')
fact
```

Out[14]:

```
{0: <gurobi.Var *Awaiting Model Update*>,
 1: <gurobi.Var *Awaiting Model Update*>,
 2: <gurobi.Var *Awaiting Model Update*>,
 3: <gurobi.Var *Awaiting Model Update*>,
 4: <gurobi.Var *Awaiting Model Update*>,
 5: <gurobi.Var *Awaiting Model Update*>,
 6: <gurobi.Var *Awaiting Model Update*>,
 7: <gurobi.Var *Awaiting Model Update*>,
 8: <gurobi.Var *Awaiting Model Update*>,
 9: <gurobi.Var *Awaiting Model Update*>}
```

In [15]:

```python
# 1.
m.addConstrs((gp.quicksum(aij[(customer, facility)] * fact[facility] for facility in range(0, 10)) >= 1
              for customer in range(0, 10)), name='coverage')
```

Out[15]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>}
```

In [16]:

```python
obj = gp.quicksum(f * fact[facility] for facility in range(0, 10))
m.setObjective(obj, GRB.MINIMIZE)
```

In [17]:

```python
m.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 10 rows, 10 columns and 22 nonzeros
Model fingerprint: 0xa1e43763
Variable types: 0 continuous, 10 integer (10 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 6.0000000
Presolve removed 10 rows and 10 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 8 available processors)

Solution count 1: 6

Optimal solution found (tolerance 1.00e-04)
Best objective 6.000000000000e+00, best bound 6.000000000000e+00, gap 0.0000%
```

In [18]:

```python
# display optimal values of decision variables

for facility in fact.keys():
    if (abs(fact[facility].x) > 1e-6):
        print(f"\n Build a factory at location {facility + 1}.")
```

```
 Build a factory at location 1.

 Build a factory at location 2.

 Build a factory at location 5.

 Build a factory at location 6.

 Build a factory at location 7.

 Build a factory at location 9.
```

8.9

In [19]:

```python
p = 4
c = 2.5
```

In [20]:

```python
# demand
df_demand
```

Out[20]:

```
[60, 27, 29, 26, 33, 15, 17, 97, 97, 19]
```

In [21]:

```python
# aij
aij = {(customer, facility): 1 if df_dis.iloc[customer, facility] <= c else 0
            for customer in range(0, 10)
            for facility in range(0, 10)}

print("Number of viable pairings: {0}".format(len(aij.keys())))
```

```
Number of viable pairings: 100
```

In [22]:

```python
# Decision variables: facilities open or close
fact = m.addVars(10, vtype = GRB.BINARY, name='fact')
fact
```

Out[22]:

```
{0: <gurobi.Var *Awaiting Model Update*>,
 1: <gurobi.Var *Awaiting Model Update*>,
 2: <gurobi.Var *Awaiting Model Update*>,
 3: <gurobi.Var *Awaiting Model Update*>,
 4: <gurobi.Var *Awaiting Model Update*>,
 5: <gurobi.Var *Awaiting Model Update*>,
 6: <gurobi.Var *Awaiting Model Update*>,
 7: <gurobi.Var *Awaiting Model Update*>,
 8: <gurobi.Var *Awaiting Model Update*>,
 9: <gurobi.Var *Awaiting Model Update*>}
```

In [23]:

```python
z = m.addVars(10, vtype = GRB.BINARY, name='demand')
z
```

Out[23]:

```
{0: <gurobi.Var *Awaiting Model Update*>,
 1: <gurobi.Var *Awaiting Model Update*>,
 2: <gurobi.Var *Awaiting Model Update*>,
 3: <gurobi.Var *Awaiting Model Update*>,
 4: <gurobi.Var *Awaiting Model Update*>,
 5: <gurobi.Var *Awaiting Model Update*>,
 6: <gurobi.Var *Awaiting Model Update*>,
 7: <gurobi.Var *Awaiting Model Update*>,
 8: <gurobi.Var *Awaiting Model Update*>,
 9: <gurobi.Var *Awaiting Model Update*>}
```

In [24]:

```python
# constraint
m.addConstrs((gp.quicksum(aij[(customer, facility)] * fact[facility] for facility in range(0, 10)) >= z[customer]
              for customer in range(0, 10)), name='coverage')
```

Out[24]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>}
```

In [25]:

```python
# constraint
m.addConstr((gp.quicksum(fact[facility] for facility in range(0, 10)) == p) , name='coverage')
```

Out[25]:

```
<gurobi.Constr *Awaiting Model Update*>
```

In [26]:

```python
obj = gp.quicksum(z[customer] * df_demand[customer] for customer in range(0, 10))
m.setObjective(obj, GRB.MAXIMIZE)
```

In [27]:

```python
m.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 21 rows, 30 columns and 64 nonzeros
Model fingerprint: 0xd35f46fd
Variable types: 0 continuous, 30 integer (30 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [2e+01, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 4e+00]

MIP start from previous solve produced solution with objective 367 (0.00s)
Loaded MIP start from previous solve with objective 367

Presolve removed 21 rows and 30 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 1 (of 8 available processors)

Solution count 1: 367

Optimal solution found (tolerance 1.00e-04)
Best objective 3.670000000000e+02, best bound 3.670000000000e+02, gap 0.0000%
```

In [28]:

```python
# display optimal values of decision variables

for facility in fact.keys():
    if (abs(fact[facility].x) > 1e-6):
        print(f"\n Build a warehouse at location {facility + 1}.")
```

```
 Build a warehouse at location 1.

 Build a warehouse at location 3.

 Build a warehouse at location 5.

 Build a warehouse at location 6.
```

8.19

In [29]:

```python
place = ['Akron', 'Albany',  'Nasuha', 'Scranton', 'Utica']
h = [1200000, 1150000, 1350000, 1800000, 900000]
fact = ['Bethlehem', 'Pittsburgh', 'Rochester', 'Springfield']
```

In [30]:

```python
f = [4000000, 7500000, 4500000, 5200000]
cap = [3300000, 4800000, 4200000, 3750000]
```

In [31]:

```python
# Decision variables: facilities open or close
x = m.addVars(len(fact), vtype = GRB.BINARY, name='factility')
```

In [32]:

```python
cartesian_prod = list(product(range(0, len(place)), range(0, len(fact))))
y = m.addVars(cartesian_prod, lb = 0 ,vtype = GRB.CONTINUOUS, name='custnomer friction')
```

In [33]:

```python
transp_cost = [[2.2, 1.6, 3.2, 0.8, 1.6], [1.8, 3.2, 4, 2.1, 2.4], [2.7, 1.2, 2.5, 1.4, 0.7],
               [3.8, 0.6, 0.7, 1.3, 1.5]]
```

In [34]:

```python
df_transp_cost = pd.DataFrame(transp_cost, columns = place)
df_transp_cost
```

Out[34]:

	Akron	Albany	Nasuha	Scranton	Utica
0	2.2	1.6	3.2	0.8	1.6
1	1.8	3.2	4.0	2.1	2.4
2	2.7	1.2	2.5	1.4	0.7
3	3.8	0.6	0.7	1.3	1.5

In [35]:

```python
c = {(customer, facility): df_transp_cost.iloc[facility, customer]
            for customer in range(0, len(place))
            for facility in range(0, len(fact))}
```

In [36]:

```python
# constraint
m.addConstrs((gp.quicksum(y[(customer, facility)] for facility in range(0, len(fact))) == 1
              for customer in range(0, len(place))), name='1')
```

Out[36]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>}
```

In [37]:

```python
m.addConstrs((y[customer, facility] <= x[facility] for customer in range(0, len(place))
              for facility in range(0, len(fact))), name='2')
```

Out[37]:

```
{(0, 0): <gurobi.Constr *Awaiting Model Update*>,
 (0, 1): <gurobi.Constr *Awaiting Model Update*>,
 (0, 2): <gurobi.Constr *Awaiting Model Update*>,
 (0, 3): <gurobi.Constr *Awaiting Model Update*>,
 (1, 0): <gurobi.Constr *Awaiting Model Update*>,
 (1, 1): <gurobi.Constr *Awaiting Model Update*>,
 (1, 2): <gurobi.Constr *Awaiting Model Update*>,
 (1, 3): <gurobi.Constr *Awaiting Model Update*>,
 (2, 0): <gurobi.Constr *Awaiting Model Update*>,
 (2, 1): <gurobi.Constr *Awaiting Model Update*>,
 (2, 2): <gurobi.Constr *Awaiting Model Update*>,
 (2, 3): <gurobi.Constr *Awaiting Model Update*>,
 (3, 0): <gurobi.Constr *Awaiting Model Update*>,
 (3, 1): <gurobi.Constr *Awaiting Model Update*>,
 (3, 2): <gurobi.Constr *Awaiting Model Update*>,
 (3, 3): <gurobi.Constr *Awaiting Model Update*>,
 (4, 0): <gurobi.Constr *Awaiting Model Update*>,
 (4, 1): <gurobi.Constr *Awaiting Model Update*>,
 (4, 2): <gurobi.Constr *Awaiting Model Update*>,
 (4, 3): <gurobi.Constr *Awaiting Model Update*>}
```

In [38]:

```python
m.addConstrs((gp.quicksum(y[(customer, facility)] * h[customer] for customer in range(0, len(place))) <= cap[facility]
              for facility in range(0, len(fact))), name='3')
```

Out[38]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>}
```

In [39]:

```python
# Objection
# obj = gp.quicksum(f[facility] * x[facility] + h[customer] * c[customer, facility] * y[customer, facility] for facility in range(0, len(fact)) for customer in ra
m.setObjective(gp.quicksum(h[customer] * c[customer, facility] * y[customer, facility]
                           for customer in range(0, len(place)) for facility in range(0, len(fact)))
               + gp.quicksum(f[facility] * x[facility] for facility in range(0, len(fact))) , GRB.MINIMIZE)
```

In [40]:

```python
m.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (mac64[rosetta2])
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 50 rows, 54 columns and 144 nonzeros
Model fingerprint: 0x92fe1b6b
Variable types: 20 continuous, 34 integer (34 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+06]
  Objective range  [6e+05, 8e+06]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 5e+06]

MIP start from previous solve produced solution with objective 1.76e+07 (0.01s)
MIP start from previous solve produced solution with objective 1.6265e+07 (0.01s)
Loaded MIP start from previous solve with objective 1.6265e+07

Presolve removed 21 rows and 30 columns
Presolve time: 0.00s
Presolved: 29 rows, 24 columns, 84 nonzeros
Variable types: 20 continuous, 4 integer (4 binary)

Root relaxation: objective 1.605667e+07, 17 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1.6057e+07    0    1 1.6265e+07 1.6057e+07  1.28%     -    0s
     0     0     cutoff    0      1.6265e+07 1.6265e+07  0.00%     -    0s

Explored 1 nodes (21 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 8 (of 8 available processors)

Solution count 2: 1.6265e+07 1.76e+07

Optimal solution found (tolerance 1.00e-04)
Best objective 1.626500000000e+07, best bound 1.626500000000e+07, gap 0.0000%
```

In [41]:

```python
for facility in x.keys():
    if (abs(x[facility].x) > 1e-6):
        print(fact[facility])
```

```
Bethlehem
Springfield
```

In [42]:

```python
for customer, facility in cartesian_prod:
    if (abs(y[customer, facility].x) > 1e-6):
        print(place[customer], 'is from', fact[facility])
```

```
Akron is from Bethlehem
Albany is from Springfield
Nasuha is from Springfield
Scranton is from Bethlehem
Utica is from Springfield
```