Assignment3 A0186040M

Q1

In [2]:

```
from gurobipy import *
import numpy as np

supply = np.array([100, 100, 100, 100, 100, 100])
N = len(supply)
demand = np.array([100, 100, 100, 100, 100, 100])
M = len(demand)
```

In [2]:

```
#########Model Set-up Using Function###############

def model_setup():

    m = Model("Process_Flexi")

    # number of weeks to offer price level i
    x = m.addVars(ARCS, name = "x")

    # set objective
    m.setObjective( quicksum(x[i,j] for (i,j) in ARCS), GRB.MAXIMIZE)

    # capcity constraint:
    m.addConstrs( ( quicksum(x[i,j] for (i,j) in ARCS.select(i,'*')) <= supply[i] for i in range(N)

    # demand constraint:
    m.addConstrs( ( quicksum(x[i,j] for (i,j) in ARCS.select('*',j)) <= demand[j] for j in range(M)

    #Supressing the optimization output
    m.setParam( 'OutputFlag', False )

    return m
```

In [3]:

```
#########Evaluate the open chain design for Random Demand############
ARCS = tuplelist([(0,0),(0,1), (1,1),(1,2), (2,2),(2,3), (3,3),(3,4), (4,4),(4,5), (5,5)])


#mean of the demand
mean = np.array([100, 100, 100, 100, 100, 100])

#covariance matrix of the demand (independent with s.d. 30)
cov = np.array([[900, 0, 0, 0, 0, 0],
                [0, 900, 0, 0, 0, 0],
                [0, 0, 900, 0, 0, 0],
                [0, 0, 0, 900, 0, 0],
                [0, 0, 0, 0, 900, 0],
                [0, 0, 0, 0, 0, 900]])

Sample_Size = 1000

sales_open = np.zeros(Sample_Size)

np.random.seed(123)
for i in range(Sample_Size):

    # demand is sampled from multivariate normal distribution with mean and cov (and truncated above
    demand = np.maximum(np.random.multivariate_normal(mean, cov), 0)

    # setup the model again
    m = model_setup()

    # solving the model
    m.optimize()

    # store the maximum sales for the i-th sample
    sales_open[i] = m.objVal

# compute the average of maximum sales
avg_sales_open = np.average(sales_open)

print('Average maximum sales for open chain design:', avg_sales_open)
```

```
Academic license - for non-commercial use only
Average maximum sales for open chain design: 558.2499366880537
```

In [4]:

```
#########Evaluate the long chain design for Random Demand############
ARCS = tuplelist([(0,0),(0,1), (1,1),(1,2), (2,2),(2,3), (3,3),(3,4), (4,4),(4,5), (5,5),(5,0)])


#mean of the demand
mean = np.array([100, 100, 100, 100, 100, 100])

#covariance matrix of the demand (independent with s.d. 30)
cov = np.array([[900, 0, 0, 0, 0, 0],
                [0, 900, 0, 0, 0, 0],
                [0, 0, 900, 0, 0, 0],
                [0, 0, 0, 900, 0, 0],
                [0, 0, 0, 0, 900, 0],
                [0, 0, 0, 0, 0, 900]])

Sample_Size = 1000

sales_long = np.zeros(Sample_Size)

np.random.seed(123)
for i in range(Sample_Size):

    # demand is sampled from multivariate normal distribution with mean and cov (and truncated above
    demand = np.maximum(np.random.multivariate_normal(mean, cov), 0)

    # setup the model again
    m = model_setup()

    # solving the model
    m.optimize()

    # store the maximum sales for the i-th sample
    sales_long[i] = m.objVal

# compute the average of maximum sales
avg_sales_long = np.average(sales_long)

print('Average maximum sales for long chain design:', avg_sales_long)
```

```
Average maximum sales for long chain design: 571.872486731333
```

Q2

(a)

In [5]:

```
supply = np.array([300, 500, 500])
N = len(supply)
demand = np.array([300, 500, 500])
M = len(demand)
```

In [6]:

```
#########Evaluate the open chain design for Random Demand############
ARCS = tuplelist([(0,0),(1,1),(2,2)])


#mean of the demand
mean = np.array([300, 500, 500])

#covariance matrix of the demand (independent with s.d. 30)
cov = np.array([[400, 0, 0],
                [0, 400, 0],
                [0, 0, 1600]])

Sample_Size = 1000

sales_open = np.zeros(Sample_Size)

np.random.seed(123)
for i in range(Sample_Size):

    # demand is sampled from multivariate normal distribution with mean and cov (and truncated above
    demand = np.maximum(np.random.multivariate_normal(mean, cov), 0)

    # setup the model again
    m = model_setup()

    # solving the model
    m.optimize()

    # store the maximum sales for the i-th sample
    sales_open[i] = m.objVal

# compute the average of maximum sales
avg_sales_open = np.average(sales_open)

print('Average maximum sales :', avg_sales_open)
```

```
Average maximum sales : 1268.8236304807747
```

(c)

In [7]:

```
supply = np.array([300, 500, 500])
N = len(supply)
M = N

cost = np.array([[0, 22, 19],
                 [22, 0, 7],
                 [19, 7, 0]])
```

In [8]:

```
def model_setup():

    m = Model("Process_Flexi")

    # number of weeks to offer price level i
    x = m.addVars(ARCS, name = "x")

    # set objective
    m.setObjective( quicksum((100-cost[i,j])*x[i,j] for (i,j) in ARCS)-200-1300*50, GRB.MAXIMIZE)

    # capcity constraint:
    m.addConstrs( ( quicksum(x[i,j] for (i,j) in ARCS.select(i,'*')) <= supply[i] for i in range(N)

    # demand constraint:
    m.addConstrs( ( quicksum(x[i,j] for (i,j) in ARCS.select('*',j)) <= demand[j] for j in range(M)

    #Supressing the optimization output
    m.setParam( 'OutputFlag', False )

    return m
```

In [9]:

```
#########Evaluate the open chain design for Random Demand############
ARCS = tuplelist([(0,0), (0,1), (0,2),
                  (1,0), (1,1), (1,2),
                  (2,0), (2,1), (2,2)])


#mean of the demand
mean = np.array([300, 500, 500])

#covariance matrix of the demand (independent with s.d. 30)
cov = np.array([[400, 0, 0],
                [0, 400, 0],
                [0, 0, 1600]])

Sample_Size = 10000

profit = np.zeros(Sample_Size)

np.random.seed(12)
for i in range(Sample_Size):

    # demand is sampled from multivariate normal distribution with mean and cov (and truncated above
    demand = np.maximum(np.random.multivariate_normal(mean, cov), 0)

    # setup the model again
    m = model_setup()

    # solving the model
    m.optimize()

    # store the maximum sales for the i-th sample
    profit[i] = m.objVal

# compute the average of maximum sales
avg_profit = np.average(profit)

print('Average maximum profit :', avg_profit)
```

```
Average maximum profit : 62680.0774000502
```

Q4

In [3]:

```
cost = np.array([[1000,3,3,10,9,10],
                 [3,1000,3,7,6,7],
                 [3,3,1000,7,6,7],
                 [10,7,7,1000,1,2],
                 [9,6,6,1,1000,1],
                 [10,7,7,2,1,1000]])
N = cost.shape[0]

#the big M
M = 10000
```

In [4]:

```
#########Model Set-up############


tsp = Model("traveling_salesman")

# Creat variables
x = tsp.addVars(N, N, vtype=GRB.BINARY, name = "x")

u = tsp.addVars(N, name = "u")

# Set objective
tsp.setObjective( quicksum(cost[i,j]*x[i,j] for i in range(N) for j in range(N)), GRB.MINIMIZE)

# Assignment constraints:
tsp.addConstrs(( quicksum(x[i,j] for j in range(N)) == 1 for i in range(N) ))

tsp.addConstrs(( quicksum(x[i,j] for i in range(N)) == 1 for j in range(N) ))

# Subtour-breaking constraints:
tsp.addConstrs(( u[i] + 1 - u[j] <= M*(1 - x[i,j])  for i in range(N) for j in range(1,N) ))


# Solving the model
tsp.optimize()
```

```
Academic license - for non-commercial use only
Optimize a model with 42 rows, 42 columns and 152 nonzeros
Variable types: 6 continuous, 36 integer (36 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [1e+00, 1e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Found heuristic solution: objective 37.0000000
Presolve removed 5 rows and 7 columns
Presolve time: 0.00s
Presolved: 37 rows, 35 columns, 150 nonzeros
Variable types: 5 continuous, 30 integer (30 binary)

Root relaxation: objective 1.300000e+01, 17 iterations, 0.00 seconds

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

     0     0   13.00000    0    6   37.00000   13.00000  64.9%     -    0s
H    0     0                      25.0000000   13.00000  48.0%     -    0s
     0     0   13.00000    0    6   25.00000   13.00000  48.0%     -    0s
H    0     0                      22.0000000   13.00000  40.9%     -    0s
     0     0 infeasible    0         22.00000   22.00000  0.00%     -    0s

Explored 1 nodes (26 simplex iterations) in 0.33 seconds
Thread count was 4 (of 4 available processors)

Solution count 3: 22 25 37

Optimal solution found (tolerance 1.00e-04)
Best objective 2.200000000000e+01, best bound 2.200000000000e+01, gap 0.0000%
```

In [5]:

```
#  Print optimal x for x nonzero and optimal value
s_edge = []
for v in x:
    if x[v].x > 0.001:
        print(x[v].VarName, x[v].x)
        #add both of the indicies by 1
        edge = np.add(v, (1,1))
        #append the edge to the resulting list of edges
        s_edge.append(edge)


print('Obj:', tsp.objVal)
print(s_edge)
for v in u:
    print(u[v].VarName, u[v].x)
```

```
x[0,2] 1.0
x[1,0] 1.0
x[2,3] 1.0
x[3,4] 1.0
x[4,5] 1.0
x[5,1] 1.0
Obj: 22.0
[array([1, 3]), array([2, 1]), array([3, 4]), array([4, 5]), array([5, 6]), array
([6, 2])]
u[0] 0.0
u[1] 5.0
u[2] 1.0
u[3] 2.0
u[4] 3.0
u[5] 4.0
```

(b)

In [19]:

```
#########Model Set-up############
cost = np.array([[1000,3,3,10,9,10],
                 [3,1000,3,7,6,7],
                 [3,3,1000,7,6,7],
                 [10,7,7,1000,1,2],
                 [9,6,6,1,1000,1],
                 [10,7,7,2,1,1000]])
waiting_time = np.array([0,5,10,15,13,14])
tsp = Model("traveling_salesman")

# Creat variables
x = tsp.addVars(N, N, vtype=GRB.BINARY, name = "x")

u = tsp.addVars(N, name = "u")

# Set objective
tsp.setObjective( quicksum(cost[i,j]*x[i,j] for i in range(N) for j in range(N)), GRB.MINIMIZE)

# Assignment constraints:
tsp.addConstrs(( quicksum(x[i,j] for j in range(N)) == 1 for i in range(N) ))

tsp.addConstrs(( quicksum(x[i,j] for i in range(N)) == 1 for j in range(N) ))


# Subtour-breaking constraints:
tsp.addConstrs(( u[i] + cost[i,j] - u[j] <= M*(1 - x[i,j])  for i in range(N) for j in range(1,N)))
tsp.addConstrs((u[i] <= waiting_time[i] for i in range(N)))

# Solving the model
tsp.optimize()
```

```
Optimize a model with 48 rows, 42 columns and 158 nonzeros
Variable types: 6 continuous, 36 integer (36 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [1e+00, 1e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+04]
Presolve removed 22 rows and 17 columns
Presolve time: 0.00s
Presolved: 26 rows, 25 columns, 134 nonzeros
Variable types: 4 continuous, 21 integer (21 binary)

Root relaxation: objective 1.300000e+01, 15 iterations, 0.00 seconds

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

     0     0   13.00000    0    6          -   13.00000      -     -    0s
H    0     0                      25.0000000   13.00000  48.0%     -    0s
     0     0   13.00000    0    6   25.00000   13.00000  48.0%     -    0s
     0     0   13.00000    0    6   25.00000   13.00000  48.0%     -    0s
     0     0   13.57143    0   11   25.00000   13.57143  45.7%     -    0s
     0     0   13.70588    0   11   25.00000   13.70588  45.2%     -    0s
     0     0   15.25000    0   11   25.00000   15.25000  39.0%     -    0s
     0     0   15.25000    0   11   25.00000   15.25000  39.0%     -    0s
     0     0   16.05357    0   11   25.00000   16.05357  35.8%     -    0s
     0     0   16.05357    0   13   25.00000   16.05357  35.8%     -    0s
```

```
     0     0   16.05357    0    9   25.00000   16.05357  35.8%     -    0s
     0     2   16.05357    0    9   25.00000   16.05357  35.8%     -    0s

Cutting planes:
  Clique: 1
  MIR: 2

Explored 16 nodes (94 simplex iterations) in 0.29 seconds
Thread count was 4 (of 4 available processors)

Solution count 1: 25

Optimal solution found (tolerance 1.00e-04)
Best objective 2.500000000000e+01, best bound 2.500000000000e+01, gap 0.0000%
```

In [20]:

```
# Print optimal x for x nonzero and optimal value
s_edge = []
for v in x:
    if x[v].x > 0.001:
        print(x[v].VarName, x[v].x)
        #add both of the indicies by 1
        edge = np.add(v, (1,1))
        #append the edge to the resulting list of edges
        s_edge.append(edge)


print('Obj:', tsp.objVal)
print(s_edge)
for v in u:
    print(u[v].VarName, u[v].x)
```

```
x[0,1] 1.0
x[1,2] 1.0
x[2,4] 1.0
x[3,0] 1.0
x[4,5] 1.0
x[5,3] 1.0
Obj: 25.0
[array([1, 2]), array([2, 3]), array([3, 5]), array([4, 1]), array([5, 6]), array
([6, 4])]
u[0] 0.0
u[1] 3.0
u[2] 6.0
u[3] 15.0
u[4] 11.99999999999994
u[5] 13.0
```

Q5

(b)

In [20]:

```
#########Model Set-up############
setup_cost = 2
holding_cost = 0.2
M = 1000
d = np.array([0,3,2,3,2])
model = Model("production_problem")

# Creat variables
x = model.addVars(5,name = "x")
y = model.addVars(5,name = "y")

u = model.addVars(5,vtype=GRB.BINARY, name = "u")

# Set objective
model.setObjective(0.2*quicksum(x[i] for i in range(5))+2*quicksum(u[i] for i in range(5)), GRB.MIN

# Assignment constraints:
model.addConstrs(x[i-1] + y[i] - x[i] == d[i] for i in range(1,5))

model.addConstrs((quicksum(y[i] for i in range(5)) == 10 for i in range(5)))

model.addConstrs(y[i]<= M*u[i] for i in range(5))

#model.addConstrs(x[0]==0)


# Solving the model
model.optimize()
```

```
Optimize a model with 14 rows, 15 columns and 47 nonzeros
Variable types: 10 continuous, 5 integer (5 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+03]
  Objective range  [2e-01, 2e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 1e+01]
Presolve removed 4 rows and 2 columns
Presolve time: 0.00s
Presolved: 10 rows, 13 columns, 26 nonzeros
Variable types: 8 continuous, 5 integer (5 binary)

Root relaxation: objective 2.600000e+00, 8 iterations, 0.00 seconds

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

     0     0    2.60000    0    4          -    2.60000      -     -    0s
H    0     0                         8.6000000    2.60000  69.8%     -    0s
H    0     0                         4.8000000    2.60000  45.8%     -    0s
     0     0    4.00000    0    4    4.80000    4.00000  16.7%     -    0s
     0     0     cutoff    0         4.80000    4.80000  0.00%     -    0s

Cutting planes:
  Gomory: 3
  MIR: 1
  Flow cover: 2

Explored 1 nodes (17 simplex iterations) in 0.29 seconds
```

```
Thread count was 4 (of 4 available processors)

Solution count 2: 4.8 8.6

Optimal solution found (tolerance 1.00e-04)
Best objective 4.800000000000e+00, best bound 4.800000000000e+00, gap 0.0000%
```

In [23]:

```
#  Print optimal x for x nonzero and optimal value
s_edge = []
for v in y:
    if y[v].x > 0.001:
        print(y[v].VarName, y[v].x)
        #add both of the indicies by 1
        edge = np.add(v, (1,1))
        #append the edge to the resulting list of edges
        s_edge.append(edge)

print('Obj:', model.objVal)
```

```
y[1] 5.0
y[3] 5.0
Obj: 4.8
```

In [24]:

```
s_edge = []
for v in x:
    if x[v].x > 0.001:
        print(x[v].VarName, x[v].x)
        #add both of the indicies by 1
        edge = np.add(v, (1,1))
        #append the edge to the resulting list of edges
        s_edge.append(edge)
```

```
x[1] 2.0
x[3] 2.0
```

(c)

In [31]:

```
#########Model Set-up############
setup_cost = 2
holding_cost = 0.2
M = 10000
d = np.array([0,3,2,3,2])
model = Model("production_problem")

# Creat variables
x = model.addVars(5,name = "x")
y = model.addVars(5,name = "y")

u = model.addVars(5, name = "u")

# Set objective
model.setObjective(0.2*quicksum(x[i] for i in range(5))+2*quicksum(u[i] for i in range(5)), GRB.MIN

# Assignment constraints:
model.addConstrs(x[i-1] + y[i] - x[i] == d[i] for i in range(1,5))

model.addConstrs((quicksum(y[i] for i in range(5)) == 10 for i in range(5)))

model.addConstrs(y[i]<= M*u[i] for i in range(5))

model.addConstrs(u[i] <= 1 for i in range(5))
#model.addConstrs(x[0]==0)


# Solving the model
model.optimize()
```

```
Optimize a model with 19 rows, 15 columns and 52 nonzeros
Coefficient statistics:
  Matrix range     [1e+00, 1e+04]
  Objective range  [2e-01, 2e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+01]
Presolve removed 14 rows and 5 columns
Presolve time: 0.01s
Presolved: 5 rows, 10 columns, 17 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.0000000e-03   0.000000e+00   0.000000e+00      0s
       0    2.0000000e-03   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.02 seconds
Optimal objective  2.000000000e-03
```

In [32]:

```
#  Print optimal x for x nonzero and optimal value
print('Obj:', model.objVal)
```

```
Obj: 0.002
```