OR Project

Supply Chain Network Optimization under Demand Uncertainty

Name: Bura Samshritha
Roll No.: 22IM10012

Problem Statement

We design a global supply chain network involving **five countries**: USA, Germany, Japan, Brazil, and India. The aim is to decide:

1. Which plants to operate.

2. How much each plant should produce.

3. How products should be shipped to meet customer demand at minimum cost.

Decision Variables

- **Xij**: quantity shipped from plant i to market j (continuous, ≥ 0).

- **Yi**: binary variable, 1 if plant i is opened, 0 otherwise.

Parameters

- **Dj**: demand in market j.
- **Ci**: maximum capacity of plant i (units/month).
- **Fi**: fixed cost of opening plant iii.
- **vi**: per-unit variable production cost at plant iii.
- **fij**: per-unit freight cost from iii to j.
- **bi**: quadratic coefficient capturing congestion at plant iii.

<u>Mathematical Formulation (MIQP)</u>

<u>Objective: minimize total cost</u> -

Min Z = (i∈I∑) (j∈J∑) (vi * Xij + bi * Xij2 + fij * Xij) + (i∈I∑) Fi * yi

<u>Constraints</u>:

1. **Demand satisfaction**

   $$(i \in I \sum) X_{ij} = D_j \qquad \forall\ j \in J$$

2. **Capacity limits (linked to activation)**

   $$(j \in J \sum) X_{ij} \leq C_i * Y_i \qquad \forall\ i \in I$$

3. **Non-negativity & binary**

   $$X_{ij} \geq 0\ \forall\ i \in I\ ,\ j \in J \qquad ; \qquad Y_i \in \{0,1\}\ \forall\ i \in I$$

Why Quadratic Term?

- Linear models push all demand into the single cheapest plant.
- In reality, costs **escalate with volume** due to overtime, bottlenecks, inefficiencies and producing more units in a plant is not perfectly linear — overtime labor, machine wear, extra shifts, or subcontracting raise the per-unit cost.
- The quadratic penalty bi X^(2) reflects this: production is cheap at low levels, expensive at very high levels.
- **Business implication:** The solver naturally spreads production across plants instead of overloading one.
- bi captures the **increasing marginal cost of production at plant i**.
- In the real world, that “convex” effect is modeled by the quadratic term.

Results (to be added by you)

- Total optimal cost.
- Flows (plant → market).
- Plant activation status.

Monte Carlo Simulation for Demand Uncertainty

- Demand is rarely deterministic in real life.
- We simulate **50 scenarios** of demand using a normal distribution centered at forecast demand.
- For each scenario, we re-run the optimization model.
- We analyze:
    - How often each plant is selected.
    - Distribution of flows.
- **Final decision:** choose the network configuration that appears most frequently (robust choice).

Business Value

- Balances cost minimization with risk under demand uncertainty.
- Provides decision support for **plant location, production allocation, and logistics**.
- Demonstrates use of **operations research, optimization, and simulation** techniques.

Code:

```
!pip install docplex
```

Output: Defaulting to user installation because normal site-packages is not writeable

```
Requirement already satisfied: docplex in c:\users\bura samshritha\appdata\roaming\python\python312\site-packages (2.30.251)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from docplex) (1.16.0)
```

```
from docplex.mp.model import Model
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
random.seed(1447)

pd.set_option('display.max_colwidth', 0)
pd.set_option('display.max_columns', None)
pd.options.display.max_seq_items = 2000

%%html
<style>
.dataframe td {
   white-space: nowrap;
}
</style>


# Notebook and data folder in same directory
countries = ["USA", "GERMANY", "JAPAN", "BRAZIL", "INDIA"]

# Read all Excel files
demand_df = pd.read_excel("data/demand (1).xlsx", index_col=0)
capacity_df = pd.read_excel("data/capacity (1).xlsx", index_col=0)
fixed_cost_df = pd.read_excel("data/fixed cost.xlsx", index_col=0)
variable_cost_df = pd.read_excel("data/variable costs.xlsx", index_col=0)
freight_cost_df = pd.read_excel("data/freight costs.xlsx", index_col=0)

# Convert to dictionaries / matrices
demand = demand_df["Demand"].to_dict()
```

```
capacity_low = capacity_df["LOW"].to_dict()
capacity_high = capacity_df["HIGH"].to_dict()
fixed_low = fixed_cost_df["LOW"].to_dict()
fixed_high = fixed_cost_df["HIGH"].to_dict()
var_costs = variable_cost_df.loc[countries, countries].to_numpy()  # production cost matrix
freight_costs = freight_cost_df.loc[countries, countries].to_numpy() / 1000.0  # per-unit
# Create CPLEX model
mdl = Model("SupplyChain_MI_QP")
```

Demand_df (give code for visualizing this graphically with proper labels)
capacity_df (give code for visualizing this graphically with proper labels)

fixed_cost_df (give code for visualizing this graphically with proper labels)
variable_cost_df (give code for visualizing this graphically with proper labels)

freight_cost_df (give code for visualizing this graphically with proper labels)

```
# Create CPLEX model
mdl = Model("SupplyChain_MI_QP")

n = len(countries)

# Decision variables
# x[i,j] = units shipped from plant i to market j (continuous)
x = {(i,j): mdl.continuous_var(lb=0, name=f"x_{i}_{j}")
     for i in range(n) for j in range(n)}

# y[i] = 1 if plant i is active, 0 otherwise (binary)
y = {i: mdl.binary_var(name=f"y_{i}") for i in range(n)}
# Quadratic production costs + fixed cost + transportation
b = [1e-6, 1.2e-6, 0.8e-6, 1.5e-6, 0.5e-6]  # quadratic cost coefficients

total_cost = mdl.sum(
    var_costs[i,i] * x[i,j] + b[i] * x[i,j]*x[i,j] + freight_costs[i,j] * x[i,j]
    for i in range(n) for j in range(n)
) + mdl.sum(fixed_high[countries[i]] * y[i] for i in range(n))

mdl.minimize(total_cost)

# 1. Meet demand in each country
for j in range(n):
    mdl.add_constraint(mdl.sum(x[i,j] for i in range(n)) == demand[countries[j]],
                       ctname=f"demand_{j}")
```

```
# 2. Plant capacity (link to activation)
for i in range(n):
    mdl.add_constraint(mdl.sum(x[i,j] for j in range(n)) <= capacity_high[countries[i]]*y[i],
                       ctname=f"capacity_{i}")
solution = mdl.solve(log_output=True)  # set log_output=False to hide logs
if solution:
    print("Optimal total cost:", solution.objective_value)
else:
    print("No feasible solution found")
```

Output:

```
Version identifier: 22.1.2.0 | 2024-12-09 | 8bd2200c8
CPXPARAM_Read_DataCheck                          1
3 of 3 MIP starts provided solutions.
MIP start 'm2' defined initial solution with objective 7.8322e+07.
Tried aggregator 1 time.
MIQP Presolve eliminated 20 rows and 0 columns.
Reduced MIQP has 10 rows, 30 columns, and 55 nonzeros.
Reduced MIQP has 5 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 25 nonzeros.
Presolve time = 0.09 sec. (0.05 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIQP has 10 rows, 30 columns, and 55 nonzeros.
Reduced MIQP has 5 binaries, 0 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 25 nonzeros.
Presolve time = 0.02 sec. (0.02 ticks)
Classifier predicts products in MIQP should be linearized.
Probing time = 0.00 sec. (0.00 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.06 sec. (0.20 ticks)

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                         7.83218e+07        0.0000           100.00%
      0     0   7.25313e+07     1    7.83218e+07   7.25313e+07       14    7.39%
      0     0   7.78914e+07     2    7.83218e+07       Cuts: 4       20    0.55%
      0     0        cutoff          7.83218e+07                     25    0.00%
Elapsed time = 0.39 sec. (0.62 ticks, tree = 0.01 MB, solutions = 3)

Flow cuts applied:  1
Gomory fractional cuts applied:  1

Root node processing (before b&c):
  Real time             =    0.39 sec. (0.62 ticks)
Parallel b&c, 8 threads:
  Real time             =    0.00 sec. (0.00 ticks)
```

```
  Sync time (average)   =    0.00 sec.
  Wait time (average)   =    0.00 sec.
                            ------------
Total (root+branch&cut) =    0.39 sec. (0.62 ticks)
Optimal total cost: 78321799.15965211
```

```
# Flows
flows = pd.DataFrame(np.zeros((n,n)), index=countries, columns=countries)
for i in range(n):
    for j in range(n):
        flows.iloc[i,j] = x[i,j].solution_value

# Plant activation
plant_status = {countries[i]: y[i].solution_value for i in range(n)}

print("\nOptimal flows (rows = plants, cols = markets):")
print(flows.round(2))
print("\nPlant activation (1=open, 0=closed):")
print(plant_status)
```

Output: Optimal flows (rows = plants, cols = markets):

```
                USA   GERMANY       JAPAN    BRAZIL     INDIA
USA     -0.00        0.00      0.00         0.0       0.0
GERMANY  164226.99   48086.42  182686.59   0.0       0.0
JAPAN    259280.30   0.00      1240719.70  0.0       0.0
BRAZIL   1036492.70  41913.58  276593.71   145000.0  0.0
INDIA    1340000.00  0.00      0.00        0.0       160000.0

Plant activation (1=open, 0=closed):
{'USA': 0, 'GERMANY': 1.0, 'JAPAN': 1.0, 'BRAZIL': 1.0, 'INDIA': 1.0}
```

Initial linear programming code where monte carlo is also applied :

```
# comparision
import pandas as pd
import numpy as np
```

```
from pulp import *
import matplotlib.pyplot as plt
import random
random.seed(1447)

pd.set_option('display.max_colwidth', 0)
pd.set_option('display.max_columns', None)
pd.options.display.max_seq_items = 2000

%%html
<style>
.dataframe td {
    white-space: nowrap;
}
</style>

# 1. production(includes service-intangible pdts too)/manufacturing  variable costs
# all m are p , but not all p are m
manvar_costs = pd.read_excel('data/variable costs.xlsx', index_col = 0)
manvar_costs

# freight variable costs
freight_costs = pd.read_excel('data/freight costs.xlsx', index_col = 0)
freight_costs

# total variable costs (manufacturing variable +freight variable)
var_cost = freight_costs/1000 + manvar_costs
var_cost

# factory fixed cost
fixed_costs = pd.read_excel('data/fixed cost.xlsx', index_col = 0)
fixed_costs

# plants capacity - low and high
cap = pd.read_excel('data/capacity (1).xlsx', index_col = 0)
cap

# Demand (by Market)
demand = pd.read_excel('data/demand (1).xlsx', index_col = 0)
demand
# Define Decision Variables
loc = ['USA', 'GERMANY', 'JAPAN', 'BRAZIL', 'INDIA']
size = ['LOW', 'HIGH']
plant_name = [(i,s) for s in size for i in loc]
```

```
prod_name = [(i,j) for i in loc for j in loc]

# Initialize Class
model = LpProblem("Capacitated Plant Location Model", LpMinimize)

# Create Decision Variables
x = LpVariable.dicts("production_", prod_name,
                lowBound=0, upBound=None, cat='continuous')
y = LpVariable.dicts("plant_",
                plant_name, cat='Binary')

# Define Objective Function
model += (lpSum([fixed_costs.loc[i,s] * y[(i,s)] * 1000 for s in size for i in loc])
        + lpSum([var_cost.loc[i,j] * x[(i,j)]   for i in loc for j in loc]))

# Add Constraints
for j in loc:
   model += lpSum([x[(i, j)] for i in loc]) == demand.loc[j,'Demand']
for i in loc:
   model += lpSum([x[(i, j)] for j in loc]) <= lpSum([cap.loc[i,s]*y[(i,s)] * 1000
                                         for s in size])

# Solve Model
model.solve()
print("Status: {}".format(LpStatus[model.status]))
print("Total Costs: {:,.2f} ($/Month)".format(pulp.value(model.objective)))


# Results Plant (Boolean)
df_bool = pd.DataFrame(data = [y[plant_name[i]].varValue for i in range(len(plant_name))],
index = [i + '-' + s for s in size for i in loc],
                  columns = ['Plant Opening'])
df_bool
# Plant Opening graph
cap_plot = cap.copy()

ax = df_bool.astype(int).plot.bar(figsize=(8, 5), edgecolor='black', color = 'tab:green', y='Plant
Opening', legend= False)
plt.xlabel('Plant')
plt.ylabel('Open/Close (Boolean)')
plt.title('Initial Solution')
plt.show()

##simulating several scenarios
```

```
def optimization_model(fixed_costs, var_cost, demand, demand_col, cap):
    '''Build the optimization based on input parameters'''
    # Define Decision Variables
    loc = ['USA', 'GERMANY', 'JAPAN', 'BRAZIL', 'INDIA']
    size = ['LOW', 'HIGH']
    plant_name = [(i,s) for s in size for i in loc]
    prod_name = [(i,j) for i in loc for j in loc]

    # Initialize Class
    model = LpProblem("Capacitated Plant Location Model", LpMinimize)

    # Create Decision Variables
    x = LpVariable.dicts("production_", prod_name,
                     lowBound=0, upBound=None, cat='continuous')
    y = LpVariable.dicts("plant_",
                     plant_name, cat='Binary')

    # Define Objective Function
    model += (lpSum([fixed_costs.loc[i,s] * y[(i,s)] * 1000 for s in size for i in loc])
            + lpSum([var_cost.loc[i,j] * x[(i,j)]   for i in loc for j in loc]))

    # Add Constraints
    for j in loc:
        model += lpSum([x[(i, j)] for i in loc]) == demand.loc[j,demand_col]
    for i in loc:
        model += lpSum([x[(i, j)] for j in loc]) <= lpSum([cap.loc[i,s]*y[(i,s)] * 1000
                                                   for s in size])
    # Solve Model
    model.solve()

# Results
    status_out = LpStatus[model.status]
    objective_out  = pulp.value(model.objective)
    plant_bool = [y[plant_name[i]].varValue for i in range(len(plant_name))]
    fix = sum([fixed_costs.loc[i,s] * y[(i,s)].varValue * 1000 for s in size for i in loc])
    var = sum([var_cost.loc[i,j] * x[(i,j)].varValue for i in loc for j in loc])
    plant_prod = [x[prod_name[i]].varValue for i in range(len(prod_name))]
    return status_out, objective_out, y, x, fix, var

#normal distribution of demand
N = 50
df_demand = pd.DataFrame({'scenario': np.array(range(1, N + 1))})
data = demand.reset_index()
# Demand
```

CV = 0.5 (#coefficient of variation CV=σ/μ) (Using CV ensures that volatility is proportional to market size.#Big markets (mean = 1000) → bigger fluctuations (σ = 500).#Small markets (mean = 100) → smaller fluctuations (σ = 50).)

```
markets = data['(Units/month)'].values
for col, value in zip(markets, data['Demand'].values):
    sigma = CV * value
    df_demand[col] = np.random.normal(value, sigma, N)
    df_demand[col] = df_demand[col].apply(lambda t: t if t>=0 else 0)

# Add Initial Scenario
COLS = ['scenario'] + list(demand.index)
VALS = [0] + list(demand['Demand'].values)
df_init = pd.DataFrame(dict(zip(COLS, VALS)), index = [0])

# Concat
df_demand = pd.concat([df_init, df_demand])
df_demand.to_excel('data/df_demand-{}PC.xlsx'.format(int(CV * 100)))

df_demand.astype(int).head()

# Plot
figure, axes = plt.subplots(len(markets), 1)
colors = ['tab:green', 'tab:red', 'black', 'tab:blue', 'tab:orange']
for i in range(len(markets)):
    df_demand.plot(figsize=(20, 12), xlim=[0,N], x='scenario', y=markets[i], ax=axes[i], grid = True, color = colors[i])
    axes[i].axhline(df_demand[markets[i]].values[0], color=colors[i], linestyle="--")
plt.xlabel('Scenario')
plt.ylabel('(Units)')
plt.xticks(rotation=90)
plt.show()

# calculation : initial scenario
# Record results per scenario
list_scenario, list_status, list_results, list_totald, list_fixcost, list_varcost = [], [], [], [], [], []
# Initial Scenario
status_out, objective_out, y, x, fix, var = optimization_model(fixed_costs, var_cost, demand, 'Demand', cap)

# Add results
list_scenario.append('INITIAL')
total_demand = demand['Demand'].sum()
list_totald.append(total_demand)
list_status.append(status_out)
```

```
list_results.append(objective_out)
list_fixcost.append(fix)
list_varcost.append(var)
# Dataframe to record the solutions
df_bool = pd.DataFrame(data = [y[plant_name[i]].varValue for i in range(len(plant_name))],
index = [i + '-' + s for s in size for i in loc],
                        columns = ['INITIAL'])
df_bool.head()


# Simulate all scenarios
demand_var = df_demand.drop(['scenario'], axis = 1).T

# Loop
for i in range(1, 50): # 0 is the initial scenario
    # Calculations
    status_out, objective_out, y, x, fix, var = optimization_model(fixed_costs, var_cost,
demand_var, i, cap)

    # Append results
    list_status.append(status_out)
    list_results.append(objective_out)
    df_bool[i] = [y[plant_name[i]].varValue for i in range(len(plant_name))]
    list_fixcost.append(fix)
    list_varcost.append(var)
    total_demand = demand_var[i].sum()
    list_totald.append(total_demand)
    list_scenario.append(i)
# Final Results
# Boolean
df_bool = df_bool.astype(int)
df_bool.to_excel('data/boolean-{}PC.xlsx'.format(int(CV * 100)))
# Other Results
df_bool.head()


##FINAL PLOT
#BOOLEAN ALONE
# Plot the Grid
plt.figure(figsize = (20,4))
plt.pcolor( df_bool, cmap = 'Blues', edgecolors='k', linewidths=0.5)   #
plt.xticks([i + 0.5 for i in range(df_bool.shape[1])], df_bool.columns, rotation = 90, fontsize=12)
plt.yticks([i + 0.5 for i in range(df_bool.shape[0])], df_bool.index, fontsize=12)
plt.show()
```

```
# ADDING DEMAND
# Plot
figure, axes = plt.subplots(len(markets), 1)
colors = ['tab:green', 'tab:red', 'black', 'tab:blue', 'tab:orange']
for i in range(len(markets)):
    df_demand.plot(figsize=(15, 15), xlim=[1,N], x='scenario', y=markets[i], ax=axes[i], grid =
True, color = colors[i])
    axes[i].axhline(df_demand[markets[i]].mean(), color=colors[i], linestyle="--")
plt.xlabel('Scenario')
plt.ylabel('(Units)')

# add the scenario plot
plt.figure(figsize=(15, 5))
plt.pcolor(df_bool, cmap = 'Blues', edgecolors='k', linewidths=0.5)   #
plt.xticks([i + 0.5 for i in range(df_bool.shape[1])], df_bool.columns, rotation = 90, fontsize=12)
plt.yticks([i + 0.5 for i in range(df_bool.shape[0])], [d[0:5]+ '-H' * ('HIGH' in d) + '-L' * ('LOW' in d)
for d in df_bool.index], fontsize=12)
plt.xticks(rotation=90)
plt.show()

# FINDING OPTIMAL SOLUTION BY UNIQUE COMBINATIONS COUNT
# Unique combinations
df_unique = df_bool.T.drop_duplicates().T
df_unique.columns = ['INITIAL'] + ['C' + str(i) for i in range(1, len(df_unique.columns))]
# Plot the Grid
plt.figure(figsize = (12,4))
plt.pcolor( df_unique, cmap = 'Blues', edgecolors='k', linewidths=0.5)   #
plt.xticks([i + 0.5 for i in range(df_unique.shape[1])], df_unique.columns, rotation = 90,
fontsize=12)
plt.yticks([i + 0.5 for i in range(df_unique.shape[0])], df_unique.index, fontsize=12)
plt.show()


# Number of columns
COL_NAME, COL_NUMBER = [], []
for col1 in df_unique.columns:
    count = 0
    COL_NAME.append(col1)
    for col2 in df_bool.columns:
        if (df_bool[col2]!=df_unique[col1]).sum()==0:
            count += 1
    COL_NUMBER.append(count)
df_comb = pd.DataFrame({'column':COL_NAME, 'count':COL_NUMBER}).set_index('column')
```

```
my_circle = plt.Circle( (0,0), 0.8, color='white')
df_comb.plot.pie(figsize=(8, 8), x='column', y='count', legend= False, pctdistance=0.7,
                                    autopct='%1.0f%%', labeldistance=1.05,
                                    wedgeprops = { 'linewidth' : 7, 'edgecolor' : 'white' })
plt.xlabel('Business Vertical')
# plt.title('{:.2f} M€ Budget Applications in 9 Vertical Markets'.format(df_p['TOTAL'].sum()/1e6))
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.axis('off')
plt.show()
```