Optimization of Multistage Logistics Chain Network

May 8, 2018

1 Optimization of Multistage Logistics Chain Network:

Overview: The logistics problem in the process of satifying consumer demand with the limited supply has many elements to it, e.g. facility location, optimal distribution etc. Facility location and supply network optimization do not, in many cases, capture the true essence of real life logistics problem. Many companies deal with multi-stage logistics problem where optimizing the production quantity, distribution network and supply network is required. Here, in this project we are considering a multi-stage logistics problem which involves optimization of production quatity, distribution network and supply network. To make this problem closer to the real life situation we also have capacity constraints on supply quatity, production quantity and number of facilities (production plants and distribution centers) to be chosen from the given set of possible facility locations. Hence, this problem can viewed as a combination of multiple choice Knapsack and Capacited location-allocation problem.

The formulated problem has following decision variables,

1. $x_{i,j}$ = quantity to be produced at plant 'j' with supply from 'ith' supplier
2. $y_{j,k}$ = quantity to be sent to distribution center 'k' from the plant 'j'
3. $z_{k,l}$ = quantity to be supplied to 'lth' demand point from the distribution center 'k'
4. w_j = binary variable for plant 'j'
5. z_k = binary variable for distribution center 'k'

The objective function is comprised of following costs,

1. Cost of production
2. Cost of distribution
3. Cost of supply
4. Fixed cost for production plant and distribution center

And the constraints capture the following bounds

1. Supply capacity of each supplier

2. Production capacity for each plant
3. Storage capacity of distribution center
4. Demand satisfaction
5. Total number of plants and distribution center selected in satisfying demand

The 0-1 Mixed Integer Linear formulation for the problem is defined in the next cell and after that we create and solve a mathematical model with help Gurobi.

1.1 Mathematical Model

$$min \quad \sum_{i,j} s_{i,j}x_{i,j} + \sum_{j,k} t_{j,k}y_{j,k} + \sum_{k,l} u_{k,l}z_{k,l} + \sum_{j} f_j w_j + \sum_{k} g_k z_k \tag{1}$$

such that,

$$\sum_{j} x_{i,j} <= a_i, \forall i \tag{2}$$

$$\sum_{k} y_{j,k} <= b_j w_j, \forall j \tag{3}$$

$$\sum_{j} w_j <= P \tag{4}$$

$$\sum_{l} z_{k,l} <= c_k z_k, \forall k \tag{5}$$

$$\sum_{k} z_k <= W \tag{6}$$

$$\sum_{k} z_{k,l} >= d_l, \forall l \tag{7}$$

w_j, z_k = [0, 1], $\forall j, k$, $x_{i,j}, y_{j,k}, z_{k,l} >= 0, \forall i, j, k, l$

In [1]:
```python
# import libraries
from gurobipy import *
import numpy as np
import pandas as pd
import sys
```

In [2]:
```python
# defining dictionary for RHS
RHS_m = {
    "cap_supply": [500, 650, 390],
    "cap_plant" : [400, 550, 490, 300, 500],
    "cap_dc"    : [530, 590, 400, 370, 580],
    "demand"    : [460, 330, 450, 300]
}
```

```python
# some other parameters
parameters = {
    "suppliers"        : 3,
    "locations_plant"  : 5,
    "locations_dc"     : 5,
    "locations_demand" : 4,
    "ub_plant"         : 4,
    "ub_dc"            : 4
}

# defining dictionary for unit prod cost, distribution cost and supply cost
# we define the cost parameters as dictionary having the same keys as the respective
# variable, because it becomes very easy to use such dictionaries to generate linear
# expressions for defining objective fucntion
cost_array_p = [[5,6,4,7,5],[6,5,6,6,8],[7,6,3,9,6]]
cost_array_d = [[5,8,5,8,5],[8,7,8,6,8],[4,7,4,5,4],[3,5,3,5,3],[5,6,6,8,3]]
cost_array_s = [[7,4,5,6],[5,4,6,7],[7,5,3,6],[3,5,6,4],[4,6,5,7]]
prod_c, dist_c, supp_c = {}, {}, {}

for i in range(parameters["suppliers"]):
    for j in range(parameters["locations_plant"]):
        prod_c[(i,j)] = cost_array_p[i][j]

for i in range(parameters["locations_plant"]):
    for j in range(parameters["locations_dc"]):
        dist_c[(i,j)] = cost_array_d[i][j]

for i in range(parameters["locations_dc"]):
    for j in range(parameters["locations_demand"]):
        supp_c[(i,j)] = cost_array_s[i][j]

# defining dictionary for fixed costs
fixed_p, fixed_dc = {}, {}
fixed_p_array     = [1800, 900, 2100, 1100, 900]
fixed_dc_array    = [1000, 900, 1600, 1500, 1400]

for i in range(parameters["locations_plant"]):
    fixed_p[(i)] = fixed_p_array[i]

for i in range(parameters["locations_dc"]):
    fixed_dc[(i)] = fixed_dc_array[i]

# defining dictionary for costs
costs = {
    "unit_prod_cost"   : prod_c,
    "unit_dist_cost"   : dist_c,
    "unit_supply_cost" : supp_c,
    "fixed_plant"      : fixed_p,
```

```
        "fixed_dc"          : fixed_dc
    }
```

In [4]:
```
# Test Cell

#print(prod_c.keys())
```

In [72]:
```
# In Gurobi, to create a model out of a formulation requires following steps,
# 1. Create an empty model
# 2. Define variables, add to model, update model
# 3. Define constraints, add to model, update model
# 4. Define objective function, add to model, update model

# In this cell we'll define first three

# function for model
def get_empty_model():

    # define empty model
    m = Model()
    return m

# function for variables
def get_decision_var(m, paramaeters, RHS_m):

    # production var
    # x = m.addVars(parameters["suppliers"], parameters["locations_plant"],
    #               lb = 0, vtype = GRB.INTEGER)
    x = {}
    for i in range(parameters["suppliers"]):
        for j in range(parameters["locations_plant"]):
            x[(i,j)] = m.addVar(lb = 0, ub = RHS_m["cap_supply"][i],
                                vtype = GRB.INTEGER,
                                name = "X(%d, %d)"%(i,j))


    # distribution var
    #y = m.addVars(parameters["locations_plant"], parameters["locations_dc"], lb = 0,
    #              vtype = GRB.INTEGER)
    y = {}
    for i in range(parameters["locations_plant"]):
        for j in range(parameters["locations_dc"]):
            y[(i,j)] = m.addVar(lb = 0, ub = RHS_m["cap_plant"][i],
                                vtype = GRB.INTEGER,
                                name = "Y(%d, %d)"%(i,j))

    # supply var
    #z = m.addVars(parameters["locations_dc"], parameters["locations_demand"], lb = 0,
```

```python
    #.              vtype = GRB.INTEGER)
    z = {}
    for i in range(parameters["locations_dc"]):
        for j in range(parameters["locations_demand"]):
            z[(i,j)] = m.addVar(lb = 0, ub = RHS_m["cap_dc"][i],
                                vtype = GRB.INTEGER,
                                name = "Z(%d, %d)"%(i,j))

    # binary for plant selection
    #binary_plant = m.addVars(parameters["locations_plant"], 1, lb = 0,
    #                          vtype = GRB.BINARY)
    binary_plant = {}
    for i in range(parameters["locations_plant"]):
        binary_plant[(i)] = m.addVar(lb = 0,
                                     vtype = GRB.BINARY,
                                     name = "Plant(%d)"%(i))

    # binary for DC selection
    #binary_dc = m.addVars(parameters["locations_dc"], 1, lb = 0,
    #                       vtype = GRB.BINARY)
    binary_dc = {}
    for i in range(parameters["locations_dc"]):
        binary_dc[(i)] = m.addVar(lb = 0,
                                  vtype = GRB.BINARY,
                                  name = "DC(%d)"%(i))

    m.update()
    var = {
        "prod_quant"   : tupledict(x),
        "dist_quant"   : tupledict(y),
        "supply_quant" : tupledict(z),
        "decision_p"   : tupledict(binary_plant),
        "decision_dc"  : tupledict(binary_dc)
    }
    return m, var

# function for constraints
def get_constraints(m, var, RHS_m, paramaeters):

    con = {}

    # unpack var
    x            = var["prod_quant"]
    y            = var["dist_quant"]
    z            = var["supply_quant"]
    binary_plant = var["decision_p"]
    binary_dc    = var["decision_dc"]
```

```
    # capacity limit on production
    con["1"] = m.addConstrs(x.sum(i, '*') <= RHS_m["cap_supply"][i]
                            for i in range(parameters["suppliers"]))
    #print(con["1"].keys())

    # supply is possible from the plants for which binary_plant = 1
    con["2"] = m.addConstrs(y.sum(j, '*') - RHS_m["cap_plant"][j] * binary_plant[j]
                            <= 0
                            for j in range(parameters["locations_plant"]))
    #print(con["2"].keys())

    # upper bound on total number of plants to be constructed
    con["3"] = m.addConstr(binary_plant.sum() <= parameters["ub_plant"])

    # supply to customer points is only possible from the distribution
    # centers in existance
    con["4"] = m.addConstrs(z.sum(k, '*') - RHS_m["cap_dc"][k] * binary_dc[k] <= 0
                            for k in range(parameters["locations_dc"]))
    #print(con["4"].keys())

    # upper bound on total number of dcs to be constructed
    con["5"] = m.addConstr(binary_dc.sum() <= parameters["ub_dc"])

    # demand
    con["6"] = m.addConstrs(z.sum('*', l) >= RHS_m["demand"][l]
                            for l in range(parameters["locations_demand"]))
    #print(con["6"].keys())

    # mass balance
    con["7"] = m.addConstrs(x.sum('*', j) == y.sum(j, '*')
                            for j in range(parameters["locations_plant"]))
    con["8"] = m.addConstrs(y.sum('*', k) == z.sum(k, '*')
                            for k in range(parameters["locations_dc"]))


    m.update()
    return m, con
```

In [73]:
```
# Test Cell
m = get_empty_model()
m, var = get_decision_var(m, parameters, RHS_m)
m, con = get_constraints(m, var, RHS_m, parameters)
```

In [74]:
```
# In this cell we'll define fourth step of building model for a formulation,
# i.e we'll define objective function in this cell

# The whole objective function can be broken down into five different costs,
# we'll define expressions for each cost and will add everything together
```

```
# for defining objective function.

def get_objective(m, var, con):

    objectives = {}

    # unpack var
    x            = var["prod_quant"]
    y            = var["dist_quant"]
    z            = var["supply_quant"]
    binary_plant = var["decision_p"]
    binary_dc    = var["decision_dc"]


    # 1. Production cost
    objectives["Production"] = x.prod(costs["unit_prod_cost"])

    # 2. Distribution cost
    objectives["Distribution"] = y.prod(costs["unit_dist_cost"])

    # 3. Supply Cost
    objectives["Supply"] = z.prod(costs["unit_supply_cost"])

    # 4. Fixed cost for plant
    objectives["Fixed_Plant"] = binary_plant.prod(costs["fixed_plant"])

    # 5. Fixed cost for distribution center
    objectives["Fixed_DC"] = binary_dc.prod(costs["fixed_dc"])

    # Add objective to the model
    print(objectives.keys())
    m.setObjective(sum(objectives.itervalues()), GRB.MINIMIZE)

    # update model
    m.update()

    # solve
    m.optimize()

    # get solution
    m.printAttr('x')

    return m
```

In [75]:
```
# In this cell we'll call all the functions and then we'll solve the
# optimization problem to get the solution

# 1. define model
```

```
    m = get_empty_model()

    # 2. define vars
    m, var = get_decision_var(m, parameters, RHS_m)

    # 3. define constraints
    m, con = get_constraints(m, var, RHS_m, parameters)

    # 4. solve the optimization problem
    m = get_objective(m, var, con)
```

```
['Distribution', 'Production', 'Fixed_Plant', 'Fixed_DC', 'Supply']
Optimize a model with 29 rows, 70 columns and 185 nonzeros
Variable types: 0 continuous, 70 integer (10 binary)
Coefficient statistics:
  Matrix range     [1e+00, 6e+02]
  Objective range  [3e+00, 2e+03]
  Bounds range     [1e+00, 6e+02]
  RHS range        [4e+00, 6e+02]
Found heuristic solution: objective 37910.000000
Presolve time: 0.00s
Presolved: 29 rows, 70 columns, 185 nonzeros
Variable types: 0 continuous, 70 integer (10 binary)

Root relaxation: objective 2.800354e+04, 47 iterations, 0.00 seconds

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

     0     0 28003.5398    0    6 37910.0000 28003.5398  26.1%     -    0s
H    0     0                    30400.000000 28003.5398  7.88%     -    0s
H    0     0                    30020.000000 28003.5398  6.72%     -    0s
     0     0 28420.9128    0   12 30020.0000 28420.9128  5.33%     -    0s
     0     0 28519.7810    0   19 30020.0000 28519.7810  5.00%     -    0s
     0     0 28522.0235    0   25 30020.0000 28522.0235  4.99%     -    0s
     0     0 28675.4157    0   12 30020.0000 28675.4157  4.48%     -    0s
H    0     0                    29997.000000 28675.4157  4.41%     -    0s
H    0     0                    29640.000000 28675.4157  3.25%     -    0s
     0     0 28706.4448    0   25 29640.0000 28706.4448  3.15%     -    0s
     0     0 28715.2481    0   27 29640.0000 28715.2481  3.12%     -    0s
     0     0 28770.0000    0   15 29640.0000 28770.0000  2.94%     -    0s
H    0     0                    29478.000000 28770.0000  2.40%     -    0s
     0     0 28775.0000    0   21 29478.0000 28775.0000  2.38%     -    0s
     0     0 28781.4856    0   30 29478.0000 28781.4856  2.36%     -    0s
H    0     0                    29440.000000 28781.4856  2.24%     -    0s
     0     0 28783.5080    0   30 29440.0000 28783.5080  2.23%     -    0s
     0     0 28786.8421    0   21 29440.0000 28786.8421  2.22%     -    0s
     0     0 28786.8421    0   21 29440.0000 28786.8421  2.22%     -    0s
```

```
H    0     0                    28960.000000 28786.8421  0.60%     -    0s
     0     0 28786.8421    0    3 28960.0000 28786.8421  0.60%     -    0s
     0     0 28793.3409    0   10 28960.0000 28793.3409  0.58%     -    0s
     0     0 28821.8182    0   13 28960.0000 28821.8182  0.48%     -    0s
     0     0 28834.0741    0   17 28960.0000 28834.0741  0.43%     -    0s
     0     0 28843.1481    0   22 28960.0000 28843.1481  0.40%     -    0s
     0     0 28862.5155    0   18 28960.0000 28862.5155  0.34%     -    0s
*    0     0               0    28870.000000 28870.0000  0.00%     -    0s

Cutting planes:
  Gomory: 1
  Implied bound: 3
  MIR: 8
  Flow cover: 1

Explored 1 nodes (176 simplex iterations) in 0.15 seconds
Thread count was 4 (of 4 available processors)

Solution count 9: 28870 28960 29440 ... 37910

Optimal solution found (tolerance 1.00e-04)
Best objective 2.887000000000e+04, best bound 2.887000000000e+04, gap 0.0000%

    Variable            x
-------------------------
     X(0, 4)          500
     X(1, 1)          550
     X(1, 2)          100
     X(2, 2)          390
     Y(1, 1)          550
     Y(2, 1)           10
     Y(2, 2)          400
     Y(2, 4)           80
     Y(4, 4)          500
     Z(1, 1)          330
     Z(1, 3)          230
     Z(2, 2)          400
     Z(4, 0)          460
     Z(4, 2)           50
     Z(4, 3)           70
    Plant(1)            1
    Plant(2)            1
    Plant(4)            1
       DC(1)            1
       DC(2)            1
       DC(4)            1
```