Gerald Jones

Individual project assignment 5: Shortest Path SILSP

ISE522 Spg 22

Notebook Links:

Problem Description:

> Do the following:

1. Solve the classic single-item lot sizing problem (with fixed setup costs for orders; a slightly simpler version of the problem you solved in individual assignment 2) by modeling it as a **shortest path** problem and solving it using a solver of your choice. The dataset for this problem is attached.

2. Based on what you found in your literature review in assignment IA-5, recommend a method (apart from modeling it as a mixed-integer program) for solving the problem you solved in IA-4. (You do not need to implement the method you recommend; just describe it.)

Notes and Observations

Assumptions:

Module imports and data loading

In [1]:
```
from _GUROBI_TOOLS_.GUROBI_MODEL_BUILDING_TOOLS import *
from _NOTE_BOOK_UTILS import *
import pandas as pd
import numpy as np
import networkx as nx
from pyomo.environ import *
import pyomo.opt as po

infinity = float('inf')                # define infinity for use in expressions


notebook_title = "_IP5.ipynb"          # used to generate pdf
```

Data Load and Display

> The below cell uses the provided excel file to load the data and parameters into variables used in the pyomo implementation

In [2]:
```
# display data for problem

# grab top of file to get some parameters
param_df = pd.read_excel("classic_single_item_lot_sizing_problem_data.xlsx", nrows=2)

# get the fixed cost of ordering
FixedCharge = int(param_df.columns.tolist()[1])

# minimum amount on hand
MH = 0

# initial amount on hand
IH = 0

U = 2       # fixed holding cost

# parse demand and use it to set up the demand parameter below in the pyomo implementation
usecols = ["Demand",]
df = pd.read_excel("classic_single_item_lot_sizing_problem_data.xlsx", skiprows=[0, 1, 2, 4], nrows=10, usecol

demand = list(df["Demand"])

# set up a variable to set the node indices
nodes = list(range(11))



def generate_onebranch_tree_edges(sink):
        return [(i, i+1)    for i in range(sink-1)]

edges = generate_onebranch_tree_edges(11)

holding_costs = {}
ordering_costs = {}
demand_dict = {}
for c, edge in enumerate(edges):
    print(c, edge)
    holding_costs[edge] = U
    ordering_costs[edge] = FixedCharge
    demand_dict[edge] = demand[c-1]
print(nodes)

holding_costs
ordering_costs

BigM = sum(demand)
print(demand_dict)
BigM
```

```
0 (0, 1)
1 (1, 2)
2 (2, 3)
3 (3, 4)
4 (4, 5)
5 (5, 6)
6 (6, 7)
7 (7, 8)
8 (8, 9)
9 (9, 10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
{(0, 1): 10, (1, 2): 10, (2, 3): 10, (3, 4): 10, (4, 5): 0, (5, 6): 0, (6, 7): 15, (7, 8): 20, (8, 9): 20, (9,
10): 0}
```

Out[2]:
```
95
```

Shortest Path

- Model-Formulation-LP | Variables | Shortest Path | Objective

Sets:

> N = sets of amounts held for the week including the initial week $\in$ {0, ...,10 } (Nodes)
>
> W = set of weeks defined by tuple {i,j} $\forall$ i,j $\in$ W (Arcs or Edges)

Parameters:

> s = start week
>
> t = end week
>
> U = per unit cost of held stock in week (i,j), $\forall(i,j) \in$ W
>
> F = Fixed cost of ordering in week {i,j}
>
> D$_{i,j}$ $=$ demand for week {i,j}
>
> ## Variables
>
> C$_{i,j}$ $=$ total costs of ordering and held stock for week i, $\forall(i,j) \in$ W
>
> H$_{i,j}$ $=$ amount held at end of week i and beginning of week j (i,j), $\forall(i,j) \in$ W
>
> X$_{i,j}$ $=$ amount ordered at beginning of week i, $\forall(i,j) \in$ W
>
> O$_{i,j}$ $=$ binary decision variable for week i,j, $\forall(i,j) \in$ W

Expression, Equations, and Constraints

> Minimum order quantity (handled by nonnegative reals since minimum = 0)
>
> $$X_w \geq \mu, \forall \text{ w}$$
>
> Amount to Order Constraint:
>
> $$X_{i,j} + H_{i,j} \geq D_{i,j}$$
>
> Weekly cost Equation:
>
> $$c_{i,j} = \sum(o_{i,j} \cdot F + h_{i,j} \cdot U)$$
>
> Binary Decision Binding
>
> $$X_w \leq M \cdot O_w, \forall \text{ w}$$
>
> Amount on hand in the current week
>
> $$(X_{\text{w-1}}) - D_{\text{w-1}} + H_{\text{w-1}} = H_{\text{w}}$$

Objective:

> min($\sum C_{i,j}$)

Pyomo Implementation

In [3]:
```
# create an abstract model
U = 2
# model = AbstractModel()
model = ConcreteModel()

#                                        set up the edges and the nodes
model.W = Set(initialize=nodes)                   # represents the amount on hand at week w $\in$ {0, ...., 10
model.T = Set(within=model.W*model.W, initialize=edges)             # Edges with a cost for each edge, each ed



# set up source and sink
model.s = 0
model.t = 10


    ##########################################################################################
    ################################# Parameters set up ######################################
    ##########################################################################################
#### cost holding that week
model.u = Param(model.T, within=NonNegativeReals, initialize=holding_costs)

### demand
model.d = Param(model.T, within=NonNegativeReals, initialize=demand_dict)


    ##########################################################################################
    ################################# Variables set up #######################################
    ##########################################################################################
#### cost of a given week
model.c = Var(model.T, within=NonNegativeReals)

### amount ordered
model.x = Var(model.T, within=NonNegativeReals)



### amount on hand
model.h = Var(model.T, within=NonNegativeReals)

### if an order occured betwwen weeks i and j
model.o = Var(model.T, domain=Binary, bounds=(0,1))

    ##########################################################################################
    ################################# Objective set up #######################################
    ##########################################################################################
def minimize_costs(model):
#     return sum(model.x[i, j] * FixedCharge + model.h[i,j]*U for (i, j) in model.A if j==value(model.t))
    # return sum(model.c[i, j] for (i, j) in model.T if j==value(model.t))
    return sum(model.c[i, j] for (i, j) in model.T)

model.total = Objective(rule=minimize_costs, sense=minimize)


    ##########################################################################################
    ################################# Constraint set up ######################################
    ##########################################################################################
# rule for meeting customer demands
def amount_on_hand(model, i, j):
    if i == model.s:
        # return model.h[i, j] == model.x[i, j] - model.d[i, j]
        return model.h[i, j] == model.x[i, j] - model.d[i, j]
    elif j == model.t:
        return model.h[i, j] == 0
    else:
        return model.h[i, j] == model.x[i, j] + model.h[i-1, i] - model.d[i, j]
model.happy = Constraint(model.T, rule=amount_on_hand)

def weekly_cost(model, i, j):
    return model.c[i,j] == model.o[i, j]*FixedCharge + model.h[i, j]*U
model.weekly_costs = Constraint(model.T, rule=weekly_cost)


def binary_ordering(model, i, j):
    return model.x[i, j] <= BigM * model.o[i, j]
model.ordering_c = Constraint(model.T, rule=binary_ordering)

def ordering_constraint(model, i, j):
    if i == (model.s):
        return model.x[i, j] >= model.d[i, j] - model.x[i, j]
    else:
        return model.x[i, j] + model.h[i-1, i] >= model.d[i, j]
model.ordering_g = Constraint(model.T, rule=ordering_constraint)


# def flow_control(model, i, j):
#     if i == value(model.s):
#         rhs =  -1
#         return model.c[i, j] - model.c[j, j+1] <= 0
#     elif j == value(model.t):
#         rhs =  1
#     else:
#         rhs =  0
#     inFlow = sum(model.c[i, j] for (i,j) in model.T if j==k)
#     outFlow = sum(model.c[i, j] for (i,j) in model.T if i==k)

#     inFlow = model.c[i, j]
#     outFlow = sum(model.c[i, j] for (i,j) in model.T if i==k)

#     return inFlow - outFlow == rhs
# model.flow = Constraint(model.T, rule=flow_control)

solver = po.SolverFactory('gurobi')

result = solver.solve(model, tee=True)

print(result)
```

```
Restricted license - for non-production use only - expires 2023-10-25
Read LP format model from file C:\Users\gjone\AppData\Local\Temp\tmp43267zka.pyomo.lp
Reading time = 0.00 seconds
x41: 41 rows, 41 columns, 97 nonzeros
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (win64)
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 41 rows, 41 columns and 97 nonzeros
Model fingerprint: 0x597c9ee9
Variable types: 31 continuous, 10 integer (10 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+01]
Presolve removed 27 rows and 22 columns
Presolve time: 0.00s
Presolved: 14 rows, 19 columns, 35 nonzeros
Variable types: 11 continuous, 8 integer (8 binary)
Found heuristic solution: objective 700.0000000

Root relaxation: objective 2.715789e+02, 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  271.57895    0    5  700.00000  271.57895  61.2%     -    0s
H    0     0                     640.0000000  271.57895  57.6%     -    0s
H    0     0                     490.0000000  271.57895  44.6%     -    0s
     0     0  363.50574    0    3  490.00000  363.50574  25.8%     -    0s
H    0     0                     440.0000000  363.50574  17.4%     -    0s
     0     0  381.05263    0    3  440.00000  381.05263  13.4%     -    0s
     0     0     cutoff    0       440.00000  440.00000  0.00%     -    0s

Cutting planes:
  Gomory: 2
  Implied bound: 4
  MIR: 3
  Flow cover: 3
  Flow path: 1

Explored 1 nodes (22 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 12 (of 12 available processors)

Solution count 4: 440 490 640 700

Optimal solution found (tolerance 1.00e-04)
Best objective 4.400000000000e+02, best bound 4.400000000000e+02, gap 0.0000%

Problem:
- Name: x41
  Lower bound: 439.9999999999998
  Upper bound: 439.99999999999983
  Number of objectives: 1
  Number of constraints: 41
  Number of variables: 41
  Number of binary variables: 10
  Number of integer variables: 10
  Number of continuous variables: 31
  Number of nonzeros: 97
  Sense: minimize
Solver:
- Status: ok
  Return code: 0
  Message: Model was solved to optimality (subject to tolerances), and an optimal solution is available.
  Termination condition: optimal
  Termination message: Model was solved to optimality (subject to tolerances), and an optimal solution is avai
lable.
  Wall time: 0.0
  Error rc: 0
  Time: 0.12505722045898438
Solution:
- number of solutions: 0
  number of solutions displayed: 0
```

Results Display

In [4]:
```
print("\n\t\t{}".format("Weekly strategy and outcomes") )
for (i,j) in model.T:
    print("week: {}, left-over-stock: {:.0f}, demand: {:.0f}, order:{:.0f}, cost: {:.0f}".format(i+1, value(mo

optimal_cost = value(model.total)
print("The optimal cost: ${:.2f}".format(optimal_cost))
```

```
                Weekly strategy and outcomes
week: 1, left-over-stock: 30, demand: 10, order:40, cost: 160
week: 2, left-over-stock: 20, demand: 10, order:0, cost: 40
week: 3, left-over-stock: 10, demand: 10, order:0, cost: 20
week: 4, left-over-stock: 0, demand: 10, order:0, cost: 0
week: 5, left-over-stock: 0, demand: 0, order:0, cost: 0
week: 6, left-over-stock: 0, demand: 0, order:0, cost: 0
week: 7, left-over-stock: 40, demand: 15, order:55, cost: 180
week: 8, left-over-stock: 20, demand: 20, order:0, cost: 40
week: 9, left-over-stock: 0, demand: 20, order:0, cost: 0
week: 10, left-over-stock: 0, demand: 0, order:0, cost: 0
The optimal cost: $440.00
```

In [5]:
```
held = np.array([30, 20, 10, 40, 20])

sum(held*U)
```

Out[5]:
```
240
```

Solution Discussion

> The solution suggests that for **weeks 1, and 7 amounts of 40 and 55 should be ordered respectively**. This ordering strategy leads to **ordering costs at weeks 1 and 7 of 100 each for total ordering costs of 200**. From the demands and amounts ordered there are left over stock held of 30, 20, 10 for weeks 1, 2, and 3 respectively and amounts of 40, and 20 for weeks 7 and 8. The leads to **total held over stock cost of 240** thus, a **total cost of 200 + 240 or 440** is incured during the ten week period according to the optimal solution.

Problem 2:

1. Based on what you found in your literature review in assignment IA-5, recommend a method (apart from modeling it as a mixed-integer program) for solving the problem you solved in IA-4. (You do not need to implement the method you recommend; just describe it.)

Answer:

> From our literature review we found the algorithm purposed by Fredrgruen and Lee (1990) that looked at both incremental and all-unit discounts. Their solution can uses a dynamic programing algortihm with complextiy of O(T^2) for the all-unit discounts version, where T is the number of periods. For the problem for IP4 the value of T is 10. Their work looked at a problem with a fixed start up cost A_t and a discounted cost if the amount ordered was above some threshold $\mathbf{N}$.

Model Formulation:

Variables

> X_t amount to order at time t
>
> I_t amount of inventory on hand at time t
>
> D_t demand at time t
>
> DH_t holding cost at time t
>
> A_t start up cost at time t
>
> c_t cost without discount at time t
>
> r discount rate
>
> C_t cost of buying X_t units at time t
>
> U per unit holding cost for stock held at time t
>
> N threshold amount below which the discount does not apply
>
> M minimum order amount

cost-constraint-original

There method uses a piecwise function to control when the cost is at the discounted rate as seen in the "cost-constraint-original" expression seen below. $C(X_t) = \{$

0, iff $X_t = 0$,

$A_t + c_t \cdot X_t$, iff $0 < X_t \leq N$,

$A_t + c_t(1-r) \cdot X_t$, iff $X_t \geq N$,

}

> Their method would need a slight adjustment to the cost calculation by adding the holding cost for the stock held at the end of period T and replacing the start up cost A_t with this value which can be seen below. I would also need to add the minimum order constraint so that the MOQ rule is followed. Due to this the expression above would replace the zero amounts with the MOQ.

Modification for MOQ and holding costs

$C(X_t) =$

$X_t \cdot c_t + I_t \cdot U$, iff $M < X_t \leq N$

$X_t \cdot c_t(1-r) + I_t \cdot U$, iff $X_t \geq N$

> The goal is to minimize the cost function which is seen in the expression labeled "objective". The adjusted cost function can be seen below:

Objective:

$$Min \sum_{t=1}^{T} C(X_t)$$

Their algorithm uses two expressions to represent the demand from time i to j and the amount on hand at time i to j as shown below. These are also piecewise functions where the demand is 0 if i> j as seen below and holding cost DH(i,j).

$D(i,j) = \{$

$D_1 + \cdots + D_j$, if $i \leq j$,

$0, otherwise$ }

$DH(i,j) = \{$

$U_i[D_{1+1} + D_{1+1} + \cdots + (j-1)D_j$, if $i < j$,

$0, otherwise$ }

> Next they set constraints. First to ensure that if inventory on hand is more than enough to meet demand no order will be placed. Second, if the inventory on hand is less than demand then an order of either the limit N or the difference between demand and inventory will be places. This is slight change to there model allowing more than the threshold N to be ordered

if $I_{t-1} \geq D_t$, Xt = *0 if $I{t-1} < D_t$, Xt = D(i,j) - I*{t-1}, for some j \geq t.

Using the below algorithm you iterate through to find the most efficient option for different regeneration point paths.

Algorithm:

> step 0. Set t = i+1, P(i,t)={t}, k:=t; I:=N-D$_{t+1}$; C(i,t)=I$_t$U + c_t(1-r)N
>
> step 1: if t=j, then stop, otherwise go to step 2
>
> step 2: if I $\leq$ 0, then stop. Otherwise set C(i,t) := C(i,t) + h_k(1-r)I, t:=t+1, P(i,t) := P(i, t-1); := C(i,t-1) and go to step 3
>
> step 3: If I < Dn, set P(i,t) = P(i,t) $\cup$ {t}, k:= r, C(i,t) := C(i,t) + I_kU + ck(1-r)N, I := I + N+D_t, and go to step 1, otherwise, see I_t=I_{t-1}-D_t, and go to step 1
>
> Using the objective function below, we can find which of these paths results in the lowest cost.
>
> $M(i,j) = min M_k(i,j), k \in P(i,j)$

The equation will use the previously calculated P(i,t) and C(i,t) values to combine these as M_k(i,j) represents the cost of the replenishing plan for the time periods i through j. The final equation for this value is shown below.

> $M_k(i,j) = C(i, k-1) + I_t \cdot U$ + either:

{ $c_k X_k + DH(k,j)$, if $X_t < N$, or

$c_k X_k(1-r) + DH(k,j)(1-r)$, if $X_t \geq N$ }

Once this value is minimized using the constraints necessary to the specific problem, an optimal solution may be found.

In [6]:
```
# save the notebook as a pdf
# to_PDF(notebook_title)
```