Bottleneck problem

A company must complete three jobs. The amounts of processing time (in minutes) required to complete the jobs are given below.

A job cannot be processed on machine j unless for all $i < j$ the job has completed its processing on machine i. Once a job begins its processing on machine j, the job cannot be preempted on machine j. The flow time for a job is the difference between the job's completion time and the time at which the job begins its first stage of processing. Formulate a GAMSPy model whose solution can be used to minimize the average flow time of the three jobs.

Hint: two types of contraints will be needed: one to ensure a job cannot begin to be procesed until all earlier portions of the job are completed. The other ensures only one job will occupy a machine at a given time.

In [133…
```
import sys
import pandas as pd
import numpy as np

import gamspy as gp
from gamspy import Sum, Card

options = gp.Options(relative_optimality_gap=0.,absolute_optimality_gap=0.5)

m = gp.Container(options=options)
```

In [134…
```
i = gp.Set(m,'i',description='job',records=[f'i{ind+1}' for ind in range(3)]
j = gp.Alias(m,'j',i)
c = gp.Set(m,'c',description='machine',records=[f'c{ind+1}' for ind in range

proctime = gp.Parameter(m,'proctime',domain=[i, c],
    records=np.array([
        [20,   0,  25, 30],
        [15, 20, 0, 18],
        [0, 35, 28, 0]]))

lastc = m.addSet('lastc',domain=c,is_singleton=True)
lastc[c] = c.last

M = m.addParameter('M',description="big M")
M[:] = Sum([i, c], proctime[i, c])

before = m.addVariable('before','binary',domain=[i, j, c],description='i bef
start = m.addVariable('start','positive',domain=[c, i])
finish = m.addVariable('finish','positive',domain=[c, i])
avgDur = m.addVariable('avgDur','free')
```

```
# # Either or Constraint
eitherC = m.addEquation('eitherC', domain=[i, j, c])
eitherC[i, j, c].where[i.ord < j.ord]= (
  start[c, i] + proctime[i, c] <= start[c, j] + M*(1-before[i, j, c]))


orC = m.addEquation('orC', domain=[i, j, c])
orC[i, j, c].where[i.ord < j.ord]= (
  start[c, j] + proctime[j, c] <= start[c, i] + M*before[i, j, c])

deffinish = m.addEquation('deffinish', domain=[c, i])
deffinish[c, i] = finish == start[c, i] + proctime[i, c]

# # Can't process in next machine until finished in machine c
defnext = m.addEquation('defnext', domain=[c, i])
defnext[c, i].where[~lastc[c]]= (
    start[c.lead(1), i] >= start[c, i] + proctime[i, c])

# # project finishes after all f done in last bath
defDur = m.addEquation('defDur')
defDur[:]=  (avgDur == Sum([i], (start['c4', i] + proctime[i, 'c4']) - start
```

In [135…

```
# put model code here
jobshop = m.addModel('jobshop',
    equations=m.getEquations(),
    problem=gp.Problem.MIP,
    sense=gp.Sense.MIN,
    objective=avgDur,
)

jobshop.solve()
```

Out[135…

	Solver Status	Model Status	Objective	Num of Equations	Num of Variables	Model Type	Solve
0	Normal	OptimalGlobal	63.6666666666667	46	37	MIP	CPLE

In [132…

```
# assumes data in start and finish variables
import plotly.express as px

results = start.records.copy()
results.columns = ['Machine','Job','Start', "_", "_", "_", "_"]
results['Finish'] = finish.records['level']
results['delta'] = results['Finish'] - results['Start']

fig = px.timeline(results, x_start="Start", x_end="Finish", y="Machine", col
fig.layout.xaxis.type = 'linear'
for d in fig.data:
  filt = results['Job'] == d.name
  d.x = results[filt]['delta'].tolist()
fig.update_yaxes(autorange="reversed")
```