Taller 2

Código	Nombre	% Participación en el trabajo /100%	Nota
20191015122	Daniel Alejandro León Castañeda	33.333%	
20191015148	Cristian David Monsalve Alfonso	33.333%	
20191015019	Nicolas Francisco Rozo Rojas	33.333%	

Universidad Distrital Francisco José de Caldas

Facultad de Ingeniería

Proyecto Curricular de Ingeniería Industrial

Bogotá D.C

2023

Taller 2

En el presente documento se encuentra la solución al taller 2 de programación y control de la producción en donde se abarcan temas de programación de secuencias para single machine con precedencias Flow shop y Jop shop por medio de heurísticas y modelos matemáticos para investigación de operaciones.

El resultado de este taller con operaciones y scripts puede encontrarse en el repositorio: https://github.com/CDMonsalveA/Taller_2_Rozo_Leon_Monsalve - además de en la carpeta de trabajo colaborativo: https://1drv.ms/f/s!AuJol_psYJ7Nh7VF6p_qOrTpogeVvA?e=Da2f5K .

Problema 1

Enunciado

Suponga que se tienen los siguientes trabajos que deben ser secuenciados y tienen restricciones de precedencia como se muestra en la Ilustración 1. Se tiene una sola máquina.

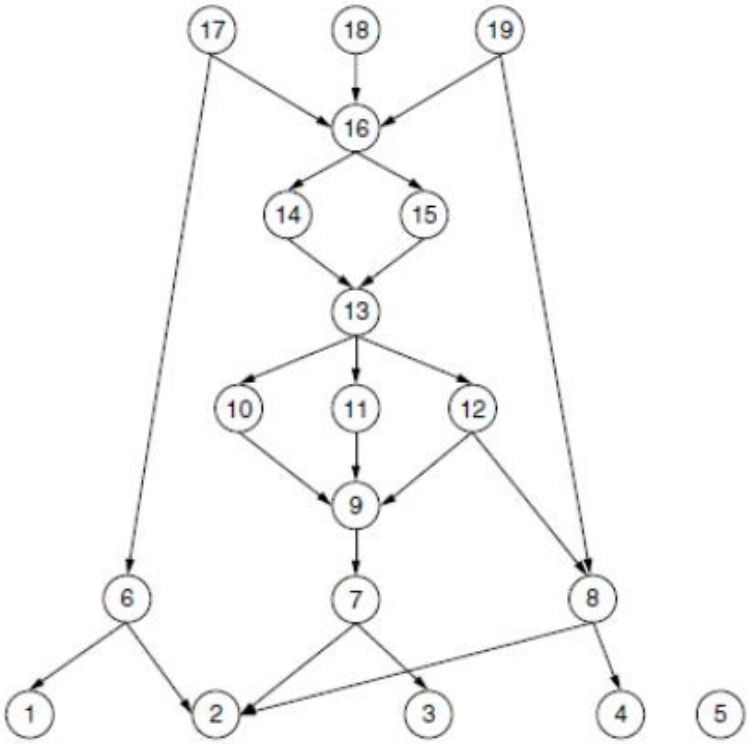


Ilustración 1: Precedencia de trabajos.

Tabla 1: Tiempos de los trabajos

Job	dj	pj	Precedencias	Suc.
1	8	15	6	
2	4	14	6,7,8	
3	7	15	7	
4	5	4	8	
5	15	13		
6	12	8	17	1,2
7	3	16	9	2,3
8	7	8	12,19	2,4
9	15	12	10,11,12	7
10	6	5	13	9
11	9	14	13	9
12	17	12	13	9
13	14	16	14,15	10,11 ,12
14	5	6	16	13
15	20	4	16	13
16	14	16	17,18,19	14,15
17	8	8		16
18	16	11		16
19	16	3		16

En la Tabla 1 se muestra las fechas de entrega d_j, los tiempos de procesamiento p_j, las actividades que son precedencias de otras y las actividades que son sucesoras de otras.

a. Con los datos presentados anteriormente usted debe codificar un algoritmo que solucione el problema de minimizar el retardo máximo. Recuerde que el algoritmo debe solucionar cualquier tipo de instancia independiente del tamaño de esta. Usted puede utilizar cualquier programa para codificar el algoritmo, pero el código debe estar debidamente comentado (java, phyton, c++, vba, gams, Xpress-MP,…).

Por la definición del ejercicio, se tiene que se trabaja con una sola máquina, se deben programar los trabajos con restricciones de precedencia y se debe minimizar el retraso. Por lo que, la notación de Graham para el presente ejercicio está definido como:

$$1|prec, d_j|h_{\max}$$

La función objetivo de min h_{max} se define asi con el fin de aplicar el algoritmo de Lawler a este ejercicio. Ya que, los datos, al ser solo tiempos de procesamiento y fechas de entrega, el ejercicio queda resumido en aplicar la regla EDD para los distintos conjuntos J', que se actualizan en medida que van quedando vacíos. Mas adelante se detallará el paso a paso.

El código en python pensado para el presente caso, se basó en el código trabajado en clase sobre job shop, es decir, haciendo uso de la librería Pulp con el fin de usar la minimización por medio de programación lineal. Por lo que, lo que se propuso fue el siguiente código:

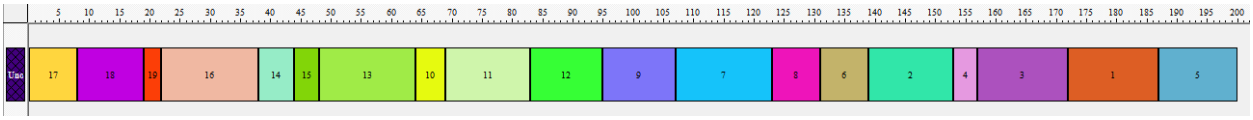
```
%pip install pulp
#Script that solves a linear programming problem using PuLP
# for single machine problem with constrains


# Import PuLP modeler functions
from pulp import *


# Create the 'prob' variable to contain the problem data
prob = LpProblem("SM",LpMinimize)
# sets
# 19 jobs
J =
["J1","J2","J3","J4","J5","J6","J7","J8","J9","J10","J11","J12","J13","J14","J15","J16","J17","J18","J19"]
p=[15.0, 14.0, 15.0, 4.0, 13.0, 8.0, 16.0, 8.0, 12.0, 5.0, 14.0, 12.0, 16.0, 6.0, 4.0, 16.0, 8.0, 11.0, 3.0]
d=[8.0, 4.0, 7.0, 5.0, 15.0, 12.0, 3.0, 7.0, 15.0, 6.0, 9.0, 17.0, 14.0, 5.0, 20.0, 14.0, 8.0, 16.0, 16.0]
p = makeDict([J],p)
d = makeDict([J],d)
# Create problem variables
# S_j = start time of job j
# C_j = completion time of job j
# Seq = Sequence variable that
# T_j = Tardiness of job j
# E_j = Earliness of job j
S = LpVariable.dicts("S",(j for j in J),lowBound=0,cat='Integer')
C = LpVariable.dicts("C",(j for j in J),lowBound=0,cat='Integer')
T = LpVariable.dicts("T",(j for j in J),lowBound=0,cat='Integer')
```

```
E = LpVariable.dicts("E",(j for j in J),lowBound=0,cat='Integer')
Tmax = LpVariable("Tmax",lowBound=0,cat='Integer')
Emax = LpVariable("Emax",lowBound=0,cat='Integer')


# Completion time of each job
for j in J:
    # get the completion time following the sequence
    prob += C[j] == S[j] + p[j], "C[%s]" % j
    # get the tardiness
    prob += T[j] >= C[j] - d[j], "T[%s]" % j
    # get the earliness
    prob += E[j] >= d[j] - C[j], "E[%s]" % j
    # get the maximum tardiness
    prob += Tmax >= T[j] , "Tmax[%s]" % j
    # get the maximum earliness
    prob += Emax >= E[j] , "Emax[%s]" % j

# precedences
# j17 must end before 16 starts
prob += C["J17"] <= S["J16"]
# J17 must end before 6 starts
prob += C["J17"] <= S["J6"]
# J18 must end before 16 starts
prob += C["J18"] <= S["J16"]
# J19 must end before 16 starts
prob += C["J19"] <= S["J16"]
# J19 must end before 8 starts
prob += C["J19"] <= S["J8"]
# J16 must end before 14 starts
prob += C["J16"] <= S["J14"]
# J16 must end before 15 starts
prob += C["J16"] <= S["J15"]
# J14 must end before 13 starts
prob += C["J14"] <= S["J13"]
# J15 must end before 13 starts
prob += C["J15"] <= S["J13"]
# J13 must end before 12 starts
prob += C["J13"] <= S["J12"]
# J13 must end before 11 starts
prob += C["J13"] <= S["J11"]
# J13 must end before 10 starts
prob += C["J13"] <= S["J10"]
# J12 must end before 9 starts
prob += C["J12"] <= S["J9"]
```

```
# J11 must end before 9 starts
prob += C["J11"] <= S["J9"]
# J10 must end before 9 starts
prob += C["J10"] <= S["J9"]
# J9 must end before 7 starts
prob += C["J9"] <= S["J7"]
# J7 must end before 2 starts
prob += C["J7"] <= S["J2"]
# J7 must end before 3 starts
prob += C["J7"] <= S["J3"]
# J6 must end before 1 starts
prob += C["J6"] <= S["J1"]
# J6 must end before 2 starts
prob += C["J6"] <= S["J2"]
# J8 must end before 2 starts
prob += C["J8"] <= S["J2"]
# J8 must end before 4 starts
prob += C["J8"] <= S["J4"]


#EDD Earliness Due Date
#Job 17 must start first
prob += S["J17"] == 0
#Job 18 must start after 17
prob += S["J18"] >= C["J17"]
#Job 19 must start after 18
prob += S["J19"] >= C["J18"]


#Job 15 must start after 14
prob += S["J15"] >= C["J14"]


#Job 10 must start after 13
prob += S["J10"] >= C["J13"]
#Job 11 must start after 10
prob += S["J11"] >= C["J10"]
#Job 12 must start after 11
prob += S["J12"] >= C["J11"]


#Job 8 must start after 7
prob += S["J8"] >= C["J7"]
#Job 6 must start after 8
prob += S["J6"] >= C["J8"]
```

```
#Job 2 must start after 6
prob += S["J2"] >= C["J6"]
#Job 4 must start after 2
prob += S["J4"] >= C["J2"]
#Job 3 must start after 4
prob += S["J3"] >= C["J4"]
#Job 1 must start after 3
prob += S["J1"] >= C["J3"]
#Job 5 must start after 1
prob += S["J5"] >= C["J1"]


# Objective function minimizes the max tardiness
prob += Tmax, "Objective function"
# The problem data is written to an .lp file
prob.writeLP("SM.lp")
# The problem is solved using PuLP's choice of Solver
prob.solve()
# The status of the solution is printed to the screen
print("Status:", LpStatus[prob.status])
# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    print(v.name, "=", v.varValue)
# The optimised objective function value is printed to the screen
print("Tardiness = ", value(prob.objective))
#Create a dictionary to hold the sequence of jobs ordered by start time
Seq = {}
#Create a dictionary to hold the start time of each job
Start = {}
#Create a dictionary to hold the completion time of each job
Comp = {}
#Put the sequence of jobs in the dictionary
for v in prob.variables():
    if v.name[0] == "S":
        Start[v.name[2:]] = v.varValue
    if v.name[0] == "C":
        Comp[v.name[2:]] = v.varValue
#Sort the jobs by start time
for key, value in sorted(Start.items(), key=lambda item: item[1]):
    Seq[key] = value
#Print the sequence of jobs
print("Sequence of jobs")
for key, value in Seq.items():
    print(key, value)
```

Con este código, se obtiene la siguiente secuencia con sus tiempos de inicio
Sequence of jobs
J17 0.0
J18 8.0
J19 19.0
J16 22.0
J14 38.0
J15 44.0
J13 48.0
J10 64.0
J11 69.0
J12 83.0
J9 95.0
J7 107.0
J8 123.0
J6 131.0
J2 139.0
J4 153.0
J3 157.0
J1 172.0
J5 187.0
Con esta secuencia se obtiene que la tardanza máxima es de 185 u.t.
Para visualizar el diagrama de gantt, se introdujo dicha secuencia en Lekin, y se halló tanto el gantt, como la tabla resumen de distintos indicadores de la secuencia.



Mientras que la tabla resumen de esta secuencia se muestra acontinuación:

Summary	
Time	1
C_{max}	200
T_{max}	185
ΣU_j	18
ΣC_j	1859
ΣT_j	1658
$\Sigma w_j C_j$	1859
$\Sigma w_j T_j$	1658

b. Formule y Resuelva un modelo de programación lineal entera mixta de manera general que represente la situación del problema anterior, formule claramente conjuntos, parámetros, variables de decisión, función objetivo y restricciones

Para formular el modelo, se definen los siguientes conjuntos y variables:

$J = \{1,2,3,\dots,19\}$

$M = \{1\}$

$p_{j1} = tiempo\ de\ procesamiento\ del\ trabajo\ j \in J\ en\ la\ máquina\ 1$

$X_{j1} = Tiempo\ de\ inicio\ del\ trabajo\ j \in J\ en\ la\ máquina\ 1$

$O_j = conjunto\ de\ operaciones\ para\ un\ trabajo\ j \in J,$

$y\ \left\{o_1^j, o_2^j, \dots, o_{|o_j|}^j\right\}\ es\ el\ orden\ de\ procesamiento\ para\ j \in J$

$$Y_{vq1} = \begin{cases} 1\ si\ el\ trabajo\ v \in J\ precede\ al\ trabajo\ q \in J | v \neq q, en\ la\ máquina\ 1 \\ 0\ de\ lo\ contrario \end{cases}$$

$C_j = Tiempo\ de\ finalización\ del\ trabajo\ j \in J$

$C_j = X_j + p_j \rightarrow Actualización\ del\ C_j\ cuando\ se\ programa\ el\ trabajo\ j \in J$

$d_j = Fecha\ de\ entrega\ del\ trabajo\ j \in J$

$$h_{max} = Retraso\ de\ la\ solución = \sum C_j - d_j = \sum X_j + p_j - d_j$$

Con las variables y conjuntos definidos, el modelo para el presente problema queda representado de la siguiente manera:

$$1|prec, d_j|h_{max}$$

$$F.O. = Min\ Z = \sum_{i=1}^{19} X_j + p_j - d_j$$

$s.a.$

$$X_{j,o_r^j} + p_{j,o_r^j} \leq X_{j,o_{r+1}^j} \quad ; \ \forall j \in J, r \in \{1, \dots, |O_j| - 1\}$$

$$X_v + p_v \leq X_q + 10000(1 - Y_{vq}) \, ; \forall v, q \in J | q \neq v$$

$$X_q + p_q \leq X_v + 10000(Y_{vq}) \, ; \forall v, q \in J | q \neq v$$

$$X_j + p_j - d_j \leq \sum_{i=1}^{19} X_j + p_j - d_j \ ; \forall i \in j$$

Con esto, para realizar el codigo en python correspondiente, se usó la libreria pulp, y se usó como base el código de “JobShop”. El codigo es el siguiente (Tambien anexo en el drive)

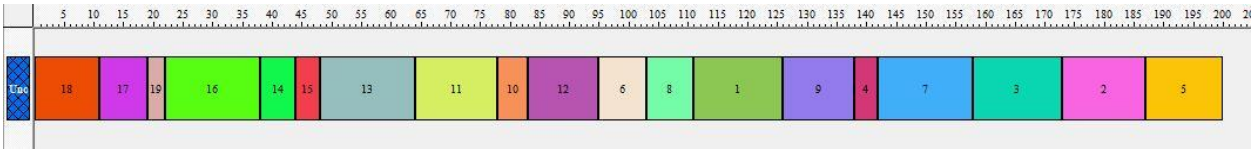
```
!pip install pulp

!pip install panda
import numpy as np
import pandas as pd
from pulp import *
J=['J1','J2','J3','J4','J5','J6','J7','J8','J9','J10','J11','J12','J13','J14','J15','J16','J17','J18','J19']
pj=[15, 14, 15, 4, 13, 8, 16, 8, 12, 5, 14, 12, 16, 6, 4, 16, 8, 11, 3]
dj=[8, 4, 7, 5, 15, 12, 3, 7, 15, 6, 9, 17, 14, 5, 20, 14, 8, 16, 16]
p=makeDict([J],pj)
d=makeDict([J],dj)
problema=LpProblem('Lawler',LpMinimize)
X=LpVariable.dicts('X',((j) for j in J),lowBound=0,cat='Trabajos')
Y=LpVariable.dicts('Y',((i,j) for i in J for j in J),lowBound=0,upBound=1,cat='Binary')
Tmax=LpVariable('Tmax',cat='Trabajos')


DG=1000
for i in J:
  for j in J:
    if(j!=i):
      problema+=X[i]+p[i]<=X[j]+DG*(1-Y[i,j])
      problema+=X[j]+p[j]<=X[i]+DG*(Y[i,j])
problema+=X['J19']+p['J19']<=X['J16']
problema+=X['J18']+p['J18']<=X['J16']
problema+=X['J17']+p['J17']<=X['J16']
problema+=X['J16']+p['J16']<=X['J15']
problema+=X['J16']+p['J16']<=X['J14']
problema+=X['J14']+p['J14']<=X['J13']
problema+=X['J15']+p['J15']<=X['J13']
problema+=X['J13']+p['J13']<=X['J12']
```

```
problema+=X['J13']+p['J13']<=X['J11']
problema+=X['J13']+p['J13']<=X['J10']
problema+=X['J12']+p['J12']<=X['J9']
problema+=X['J11']+p['J11']<=X['J9']
problema+=X['J10']+p['J10']<=X['J9']
problema+=X['J19']+p['J19']<=X['J8']
problema+=X['J12']+p['J12']<=X['J8']
problema+=X['J9']+p['J9']<=X['J7']
problema+=X['J17']+p['J17']<=X['J6']
problema+=X['J8']+p['J8']<=X['J4']
problema+=X['J7']+p['J7']<=X['J3']
problema+=X['J7']+p['J7']<=X['J2']
problema+=X['J8']+p['J8']<=X['J2']
problema+=X['J6']+p['J6']<=X['J2']
problema+=X['J6']+p['J6']<=X['J1']
problema+=X['J1']+p['J1']<=X['J5']
problema+=X['J2']+p['J2']<=X['J5']
problema+=X['J3']+p['J3']<=X['J5']
problema+=X['J4']+p['J4']<=X['J5']


for j in J:
  problema+=X[j]+p[j]-d[j]<=Tmax
problema+=Tmax
problema.writeLP(' prueba')
problema.solve()
print("Tmax=",value(problema.objective))
import operator
Sol=makeDict([J],(value(X[j])for j in J))
Sol=sorted(Sol.items(),key=operator.itemgetter(1), reverse=False)
Sol=list(map(list,Sol))
print("Secuencia")
for j in range(len(Sol)):
  for h in range(len(Sol[j])-1):
    print(Sol[j][h])
```

La solución que brinda este codigo es:
Tmax= 185.0
Secuencia
J18
J17
J19
J16
J14
J15

J13
J11
J10
J12
J6
J8
J1
J9
J4
J7
J3
J2
J5

Se observa que la secuencia cambia con respecto a la secuencia hallada con el codigo del inciso a). Sin embargo, los indicadores no cambian, lo que nos indica que las dos secuencias son factibles y óptimas, ya que minimizan el retraso, el cual es el objetivo del presente ejercicio. Para visualizar el gantt, se digito la secuencia en Lekin, de mismo modo que este software ofrece la tabla resumen de los distintos indicadores



Summary

Time	1
C_{max}	200
T_{max}	185
ΣU_j	18
ΣC_j	1840
ΣT_j	1644
$\Sigma w_j C_j$	1840
$\Sigma w_j T_j$	1644

Problema 2

Enunciado.

Suponga que cuenta con 20 trabajos que deben ser secuenciados en tres diferentes máquinas siguiendo la misma ruta M1-M2-M3, los tiempos de procesamiento se dan en la tabla.

Trabajos	M1	M2	M3
1	14	12	17
2	14	15	12
3	18	17	10
4	19	16	8
5	12	6	5
6	13	15	19
7	11	10	19
8	14	13	21
9	6	5	5
10	7	12	11
11	10	13	6
12	4	9	13
13	4	13	26
14	13	15	9
15	18	16	15
16	21	12	25
17	21	5	13
18	15	5	16
19	7	16	21
20	8	13	14

a. **Codifique un algoritmo en algún lenguaje de programación (VBA, java, c++, gams, MATLAB, etc.), que le permita la adaptación del algoritmo de Johnson para el caso de tres máquinas para minimizar el Make span.**

Algoritmo para Python pensado para ser ejecutado en Jupiter Notebooks, o Google Colab:

"""script that take the following data

[14,12,17],

[14,15,12],

[18,17,10],

[19,16,8],

[12,6,5],

[13,15,19],

[11,10,19],

[14,13,21],

[6,5,5],

[7,12,11],

[10,13,6],

[4,9,13],

[4,13,26],

[13,15,9],

[18,16,15],

[21,12,25],

[21,5,13],

[15,5,16],

[7,16,21],

[8,13,14] and apply the following algorith

for sequence of 3 machines for 20 jobs

is the Johnson algorithm"

"""

import numpy as np

#Processing time

```
P = np.array([[14,12,17], [14,15,12], [18,17,10], [19,16,8], [12,6,5], [13,15,19], [11,10,19],
[14,13,21], [6,5,5], [7,12,11], [10,13,6], [4,9,13], [4,13,26], [13,15,9], [18,16,15], [21,12,25],
[21,5,13], [15,5,16], [7,16,21], [8,13,14]])
#Number of jobs
J =
["J1","J2","J3","J4","J5","J6","J7","J8","J9","J10","J11","J12","J13","J14","J15","J16","J17","J1
8","J19","J20"]
#Number of machines
M = ["M1","M2","M3"]
# Create a dictionary with the jobs, machines and the processing time
data = {}
for i in range(len(J)):
    data[J[i]] = {}
    for j in range(len(M)):
        data[J[i]][M[j]] = P[i][j]
# Create a dictionary with the columns G1 = M1+M2 and G2 = M2+M3
G = {}
for i in range(len(J)):
    G[J[i]] = {}
    G[J[i]]["M1"] = P[i][0]+P[i][1]
    G[J[i]]["M2"] = P[i][1]+P[i][2]


def Johnson(data: dict):
```

"""Function that takes a dictionary with the jobs, machines and the processing time

and return the sequence of jobs using the johnson algorithm

where:

data: dictionary with the jobs, machines and the processing time

return: sequence of jobs

Steps

❑ Step 1. Schedule the group of jobs U that are shorter on the

first machine than the second.

U = { j | p1j < p2j }

❑ Step 2. Schedule the group of jobs V that are shorter on the

second machine than the first.

V = { j | p1j □ p2j }

❑ Step 3. Arrange jobs in U in non-decreasing order by their

processing times on the first machine.

❑ Step 4. Arrange jobs in V in non-increasing order by their

processing times on the second machine.

❑ Step 5. Concatenate U and V and that is the processing order

for both machines.

**The ties are broken at random.

"""

Step 1

U = []

for i in data:

```
        if data[i]["M1"] < data[i]["M2"]:
            U.append(i)
    # Step 2
    V = []
    for i in data:
        if data[i]["M1"] >= data[i]["M2"]:
            V.append(i)
    # Step 3
    U = sorted(U, key=lambda x: data[x]["M1"])
    # Step 4
    V = sorted(V, key=lambda x: data[x]["M2"], reverse=True)
    # Step 5
    return U+V
# Print the sequence of jobs
print("The sequence of jobs is: ", Johnson(G))
```

Teniendo en cuenta que:

$$\min(p_{1j}) \geq \max(p_{2j}) \;||\; \min(p_{3j}) \geq \max(p_{2j})$$

No se puede asegurar que la secuencia sea la óptima:

La obtenida es

The sequence of jobs is: ['J12', 'J13', 'J10', 'J18', 'J7', 'J20', 'J19', 'J1', 'J8', 'J6', 'J16', 'J15', 'J2', 'J3', 'J4', 'J14', 'J11', 'J17', 'J5', 'J9']. Para la cual se obtiene un Make Span de 298

b. Resuelva este problema aplicando el método de ramificación y acotamiento para minimizar el Make span

El método del articulo:

Ignall, E., & Schrage, L. (1965). Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. *Operations Research*, *13*(3), 400–412. https://doi.org/10.1287/opre.13.3.400

Genera problemas para la estipulación de C2 y C3 para más de 4 trabajos, ya que no define métodos para la solución de estos dos parámetros, sin embargo, se opto por una solución iterativa en Python que busca de forma semi automatizada una vez conociendo una ruta óptima las mejores opciones para llegar a la solución usando la siguiente función en Python

```
import numpy as np

J = ["J1", "J2", "J3", "J4", "J5", "J6", "J7", "J8", "J9", "J10", "J11", "J12",
"J13", "J14", "J15", "J16", "J17", "J18", "J19", "J20"] # Jobs
M = ["M1", "M2", "M3"] # Machines
P = np.array([[14,12,17],
[14,15,12],
[18,17,10],
[19,16,8],
[12,6,5],
[13,15,19],
[11,10,19],
[14,13,21],
[6,5,5],
[7,12,11],
[10,13,6],
[4,9,13],
[4,13,26],
[13,15,9],
[18,16,15],
[21,12,25],
[21,5,13],
[15,5,16],
[7,16,21],
[8,13,14]]) # Processing times
# Function that calculates the lower bound of the partial sequence
def lower_bound(Jr_parcial_secuence, J_jobs, P_processing_times):
```

```
    """Function that calculates the lower bound of the partial sequence
    Parameters
    ----------
    partial_sequence : list
        Partial sequence of jobs.
    P : numpy.ndarray
        Processing times matrix.
    Returns : float
        Lower bound of the partial sequence.
    ----------
    Process
    ----------
    sets:
    i in M
    j in J
    J = Set of jobs.
    Jr = Set of jobs that are part of the partial sequence.
    Jc = Set of jobs that are not part of the partial sequence.

    1. Calculate the latest completion time of each machine.
    if len(Jr) == 1:
        - Latest completion time of machine 1: C1(Jr) = sum_{j in Jr} P_{j,1}
        - Latest completion time of machine 2: C2(Jr) = sum_{j in Jr} P_{j,1} +
sum_{j in Jr} P_{j,2}
        - Latest completion time of machine 3: C3(Jr) = sum_{j in Jr} P_{j,1} +
sum_{j in Jr} P_{j,2} + sum_{j in Jr} P_{j,3}
    else
        - Latest completion time of machine 1: C1(Jr) = sum_{j in Jr[:-1]}
P_{j,1} + P_{Jr[-1],1}
        - Latest completion time of machine 2: C2(Jr) = max(C1(Jr)+P_{Jr[-1],2},
sum_{j in Jr[:-1]} P_{j,1} + P_{Jr[-1],2})
        - Latest completion time of machine 3: C3(Jr) = max(C2(Jr)+P_{Jr[-1],3},
sum_{j in Jr[:-1]} P_{j,1} + P_{Jr[-1],3})

    2. Calculate the bound of the partial sequence on each machine.
    - Bound of the partial sequence on machine 1: B1(Jr, Jc) = C1(Jr) + sum_{j in
Jc} P_{j,1} + min_{j in Jc}(P_{j,2}+P_{j,3})
    - Bound of the partial sequence on machine 2: B2(Jr, Jc) = C2(Jr) + sum_{j in
Jc} P_{j,2} + min_{j in Jc}(P_{j,3})
    - Bound of the partial sequence on machine 3: B3(Jr, Jc) = C3(Jr) + sum_{j in
Jc} P_{j,3}
    3. Calculate the lower bound of the partial sequence.
    - Lower bound of the partial sequence: B(Jr, Jc) = max(B1(Jr, Jc), B2(Jr,
Jc), B3(Jr, Jc))
    ----------
```

```
    """
    Jr = Jr_parcial_secuence
    P = P_processing_times
    J = J_jobs
    Jc = [j for j in J if j not in Jr]
    Pr = P[[J.index(j) for j in Jr]]
    Pc = P[[J.index(j) for j in Jc]]
    if len(Jr) == 1:
        C1 = np.sum(Pr[:,0])
        C2 = C1 + np.sum(Pr[:,1])
        C3 = C2 + np.sum(Pr[:,2])
    else:
        C1 = np.sum(Pr[:-1,0]) + Pr[-1,0]
        C2 = max(C1 + Pr[-1,1], np.sum(Pr[0,:-1]) + np.sum(Pr[1:,2]))
        C3 = max(C2 + Pr[-1,2], np.sum(Pr[0,:]) + np.sum(Pr[1:,2]))

    B1 = C1 + np.sum(Pc[:,0]) + np.min(Pc[:,1]+Pc[:,2])
    B2 = C2 + np.sum(Pc[:,1]) + np.min(Pc[:,2])
    B3 = C3 + np.sum(Pc[:,2])
    return max(B1, B2, B3)
```

a lo que, al realizar elecciones por capas, siendo la primera capa:

[['J1', 'J2', 'J3', 'J4', 'J5', 'J6', 'J7', 'J8', 'J9', 'J10', 'J11', 'J12', 'J13', 'J14', 'J15', 'J16', 'J17', 'J18', 'J19', 'J20'], [311, 314, 320, 320, 303, 313, 306, 312, 296, 304, 308, 298, 302, 313, 319, 318, 311, 305, 308, 306]]

Desde la cual se iteró finalizando en que la secuencia de 298 consistentemente mantiene el valor más bajo iniciando en J12 y así por las siguientes 18 iteraciones

para lo cual escogío la ruta.

Optimal Schedule: ['J12', 'J9', 'J13', 'J5', 'J11', 'J18', 'J10', 'J20', 'J14', 'J1', 'J7', 'J19', 'J2', 'J6', 'J17', 'J3', 'J4', 'J15', 'J8', 'J16']

c. Resuelva el problema de secuenciación de operaciones mediante un modelo de programación matemática, formule claramente conjuntos, parámetros, variables de decisión, función objetivo y restricciones.

Usando como referencia el modelo matemático aplicado en:

Wilson, J. M. (1989). Alternative Formulations of a Flow-shop Scheduling Problem. *Journal of the Operational Research Society*, *40*(4), 395–399. https://doi.org/10.1057/palgrave.jors.0400410

Que se resumirá a continuación;

Set

i: índice que indica el trabajo en donde i=1,2,3,…,N

j: índice que indica posición de procesamiento, j=1,2,3,…N

k = índice que indica la máquina j=1,2,..,M

Variables de decisión

Zij = 1 si un trabajo i es asignado a la posición j, 0 de lo contrario: implica la asignación de un trabajo i a la posición j

Sjk = Tiempo en el cual inicia un trabajo en la posición j inicia para la maquina k

Parámetros

Pik = Tiempos de procesamiento para trabajo i en la máquina k

Función objetivo

$$min\ z = S_{NM} + \sum_{i=1}^{N} P_{iM} Z_{iN}$$

Restricciones

Única posición

$$\sum_{i=1}^{N} Z_{ij} = 1 | j = 1,2,3, \dots N$$

$$\sum_{j=1}^{N} Z_{ij} = 1 = 1 | i = 1,2,3, \dots N$$

No tiempos muertos en la maquina 1 o trabajo 1

$$S_{11} = 0$$

$$S_{j+1,1} = S_{j,1} + \sum_{i=1}^{N} Z_{ij}\, p_{ik} | j = 1,2, \dots, N-1$$

$$S_{1,k+1} = S_{1,k} + \sum_{i=1}^{N} Z_{ij}\, p_{ik} | k = 1, .., M-1$$

Funcionales

$$S_{j,k+1} \geq S_{j,k} + \sum_{i=1}^{N} Z_{ij}\, p_{ik} | for j = 1,2, \dots, N; k = 1,2, \dots, M-1$$

$$S\_\{j+1,k\} \geq S\_\{j,k\} + \sum_{i=1}^{N} Z_{ij}\, p\_\{ik\}\, | for\ j\ =\ 1,2,\dots,N-1;\ k\ =\ 1,2,\dots,M$$

Con la cual se obtiene la secuencia:

Optimal Schedule: ['J12', 'J9', 'J13', 'J5', 'J11', 'J18', 'J10', 'J20', 'J14', 'J1', 'J7', 'J19', 'J2', 'J6', 'J17', 'J3', 'J4', 'J15', 'J8', 'J16']

d. Realice un diagrama de Gantt de las secuencias obtenidas, y calcular varios indicadores de desempeño y analice los resultados

	Cmax	mean(S1)	mean(S2)	mean(S3)	mean(C1)	mean(C2)	mean(C3)	Sum(Idle)
Johnson	298	104.85	121.75	170.55	117.3	133.65	184.8	12
B&B	298	94.45	110.6	136.5	106.9	122.5	150.75	19
PL	298	94.45	110.6	136.5	106.9	122.5	150.75	19

Ya que Branch and bound dio varias soluciones, se tomó la que por defecto dio el código solucionando, seleccionando aleatorio, que resultó ser la misma que por programación lineal. Se puede ver entonces que johnson programa la secuencia que deja menor tiempo inactivas las secuencias, esto al programar primero las maquinas más lentas, que se ve en el promedio de inicio y finalización de los trabajos por máquina, además de proporcionar la solución más fácil de operar, a pesar de no cumplir la prueba óptima, se encuentra una secuencia aceptable para minimizar el makespan

Los diagramas de Gantt se realizaron de manera dinámica para facilidad del lector,

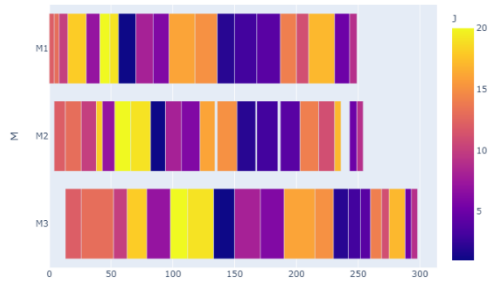


Ilustración 2: Diagrama de gantt para la secuencia del punto a

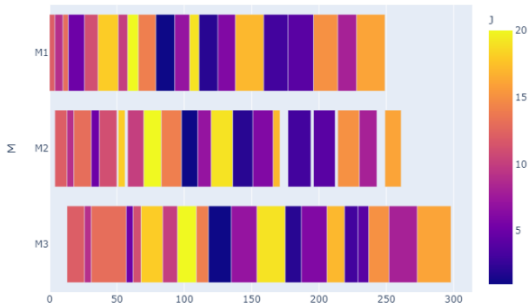


Ilustración 3: Diagrama de gantt para la secuencia de puntos b y c

La versión dinámica en html de los diagramas se encuentra en las carpetas y enlazadas:

Problema 3

Enunciado.

Una empresa manufacturera recibe 5 órdenes de trabajo que deben ser procesados en sus instalaciones. La secuencia de proceso se da en la tabla 3; los tiempos de elaboración estándar por producto p_{ij} se muestran en la tabla 4 en horas por trabajo. Las fechas de entrega se dan en la tabla 5, suponga que se trabajan 3 turnos por día, de lunes a domingo. Por condiciones logísticas del cliente el recibe en ciertos días a partir de hoy, y puede recibir a cierta hora del día (tercera columna de la tabla 5), por ejemplo para el trabajo 5 se debe entregar en 4 días a las 18 horas (6 pm).

Determine la secuencia óptima de operaciones en la empresa bajos las siguientes situaciones:

- Minimizar el Makespan.
- Minimizar el Mean Flow Time.
- Minimizar las desviaciones respecto a la fecha de entrega, suponga el costo de anticipación es de 2$/hora, y de entrega tardía de 5$/hora
- Minimizar la máxima demora
- Minimizar la máxima anticipación

La ruta de procesamiento, tiempos de procesamiento y fechas de entrega se encuentran en la Tabla 2, Tabla 3, Tabla 4 respectivamente.

Tabla 2: Ruta de Procesamiento

SECUENCIA DE EJECUCION DE LOS TRABAJOS

JOB 1	M1 – M2 – M4 – M5 – M6 – M7
JOB 2	M6 – M3 – M5 – M7 – M2 – M1
JOB 3	M3 – M7 – M6 – M1 – M5 – M4
JOB 4	M5 – M2 – M7 – M4 – M1 – M6
JOB 5	M7 – M1 – M5 – M3 – M4 – M2

Tabla 3: Tiempos de procesamiento

	MAQ 1	MAQ 2	MAQ 3	MAQ 4	MAQ 5	MAQ 6	MAQ 7
JOB 1	12	8		18	7	12	12
JOB 2	16	11	6		14	9	6
JOB 3	3		12	14	8	14	14
JOB 4	15	13		8	9	15	16
JOB 5	10	15	19	6	13		11

Tabla 4: Fechas de entrega

	Fecha De Entrega en días	Hora de recibo
JOB 1	3	6
JOB 2	4	6
JOB 3	3	6
JOB 4	4	13
JOB 5	4	18

Minimizar el Makespan

Para minimizar el Makespan, se tomará la siguiente formulación matemática,

Sets

$J: Set\ of\ jobs$

$M: Set\ of\ Machines$

Information

$O_j: set\ of\ operations\ for\ a\ job\ j \in J, and\ \left\{O_1^j, O_2^j, \dots, O_{|o_j|}^j,\right\} is\ the\ processign\ order\ for\ j \in J$

$p_{ij}: processing\ time\ of\ job\ j \in J\ on\ machine\ i \in M$

Decision variables

$X_{ji}: starting\ time\ job\ j \in J\ on\ machine\ i \in M$

$$Y_{vqi} = \begin{Bmatrix} 1\ if\ job\ v \in J\ precedes\ job\ q \in J | v \neq q, on\ machine\ i \in M \\ 0\ otherwise \end{Bmatrix}$$

$C_{max}: Makespan\ of\ the\ schedule\ edition$

Formulation

$$F5|\quad |C_{max}$$

$$\min C_{max}$$

s.t,

$$\begin{aligned}
X_{j,O_r^j} + p_{j,O_r^j} &\leq X_{j,O_{r+1}^j} && \forall j \in J, r \in \{1, \dots, |O_j| - 1\} \\
X_{vi} + p_{vi} &\leq X_{qi} + M(1 - Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
X_{qi} + p_{qi} &\leq X_{vi} + M(Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
X_{j,O_{|o_j|}^j} + p_{j,O_{|o_j|}^j} &\leq C_{max} && \forall j \in J \\
X_{ji} &\geq 0 && \forall\ j \in J, i \in M \\
Y_{vqi} &\in 0,1 && \forall v, q \in J |\ q \neq v, \forall i \in M
\end{aligned}$$

Para la resolución, se hace uso de la librería Pulp de Python, en donde el modelo anterior se encuentra en la carpeta anexa en la siguiente ruta **[*. scripts/Problema 3/a.ipynb*]**.

Minimizar el Mean Flow time

Para minimizar el Mean Flow time, se tomará la siguiente formulación matemática,

Sets

$J: Set\ of\ jobs$

$M: Set\ of\ Machines$

Information

O_j: $set\ of\ operations\ for\ a\ job\ j \in J, and\ \left\{O_1^j, O_2^j, \dots, O_{|O_j|}^j,\right\} is\ the\ processign\ order\ for\ j \in J$

p_{ij}: $processing\ time\ of\ job\ j \in J\ on\ machine\ i \in M$

Decision variables

X_{ji}: $starting\ time\ job\ j \in J\ on\ machine\ i \in M$

$$Y_{vqi} = \left\{ \begin{matrix} 1\ if\ job\ v \in J\ precedes\ job\ q \in J | v \neq q, on\ machine\ i \in M \\ 0\ otherwise \end{matrix} \right\}$$

Formulation

$$F5| \quad | \min \sum_j^J C_j$$

$$\min \sum_j^J C_j$$

s.t,

$$\begin{matrix} X_{j,O_r^j} + p_{j,O_r^j} \leq X_{j,O_{r+1}^j} & \forall j \in J, r \in \{1, \dots, |O_j| - 1\} \\ X_{vi} + p_{vi} \leq X_{qi} + M(1 - Y_{v,q,i}) & \forall i \in M, \forall v, q \in J | q \neq v \\ X_{qi} + p_{qi} \leq X_{vi} + M(Y_{v,q,i}) & \forall i \in M, \forall v, q \in J | q \neq v \\ X_{ji} \geq 0 & \forall\ j \in J, i \in M \\ Y_{vqi} \in 0,1 & \forall v, q \in J |\ q \neq v, \forall i \in M \end{matrix}$$

El modelo anterior se encuentra en la carpeta anexa en la siguiente ruta **[*. scripts/Problema 3/b.ipynb*]**.

Minimizar las desviaciones respecto a la fecha de entrega, suponga el costo de anticipación es de 2$/hora, y de entrega tardía de 5$/hora

Para minimizar las desviaciones, se tomará la siguiente formulación matemática,

Sets

$J: Set\ of\ jobs$

$M: Set\ of\ Machines$

Information

$O_j: set\ of\ operations\ for\ a\ job\ j \in J, and\ \left\{O_1^j, O_2^j, \dots, O_{|O_j|}^j,\right\} is\ the\ processign\ order\ for\ j \in J$

$p_{ij}: processing\ time\ of\ job\ j \in J\ on\ machine\ i \in M$

Decision variables

$X_{ji}: starting\ time\ job\ j \in J\ on\ machine\ i \in M$

$Y_{vqi} = \left\{\begin{matrix} 1\ if\ job\ v \in J\ precedes\ job\ q \in J | v \neq q, on\ machine\ i \in M \\ 0\ otherwise \end{matrix}\right\}$

$E_j: Earliness\ for\ job\ j \in J$

$T_j: Tardiness\ for\ job\ j \in J$

Formulation

$$F5| \quad | \min 2 \times \sum_j^J E_j + 5 \times \sum_j^J T_j$$

$$\min 2 \times \sum_j^J E_j + 5 \times \sum_j^J T_j$$

s.t,

$$\begin{aligned}
X_{j,O_r^j} + p_{j,O_r^j} &\leq X_{j,O_{r+1}^j} && \forall j \in J, r \in \{1, \dots, |O_j| - 1\} \\
X_{vi} + p_{vi} &\leq X_{qi} + M(1 - Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
X_{qi} + p_{qi} &\leq X_{vi} + M(Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
X_{j,O_{|O_j|}^j} + p_{j,O_{|O_j|}^j} - T_j + E_j &= d_j && \forall j \in J
\end{aligned}$$

$$X_{ji} \geq 0 \qquad \forall\, j \in J, i \in M$$
$$Y_{vqi} \in 0{,}1 \qquad \forall v, q \in J |\ q \neq v, \forall i \in M$$

El modelo anterior se encuentra en la carpeta anexa en la siguiente ruta **[*. scripts/Problema 3/c.ipynb*]**.

Minimizar la máxima demora

Para minimizar la máxima demora, se tomará la siguiente formulación matemática,

Sets

$J: Set\ of\ jobs$

$M: Set\ of\ Machines$

Information

$O_j: set\ of\ operations\ for\ a\ job\ j \in J, and\ \left\{O_1^j, O_2^j, \dots, O_{|O_j|}^j,\right\} is\ the\ processign\ order\ for\ j \in J$

$p_{ij}: processing\ time\ of\ job\ j \in J\ on\ machine\ i \in M$

Decision variables

$X_{ji}: starting\ time\ job\ j \in J\ on\ machine\ i \in M$

$$Y_{vqi} = \left\{ \begin{matrix} 1\ if\ job\ v \in J\ precedes\ job\ q \in J | v \neq q, on\ machine\ i \in M \\ 0\ otherwise \end{matrix} \right\}$$

$T_{max}: Maximum\ Tardiness\ of\ the\ schedule\ edition$

Formulation

$$F5|\quad |T_{max}$$

$$\min T_{max}$$

s.t,

$$X_{j,O_r^j} + p_{j,O_r^j} \leq X_{j,O_{r+1}^j} \qquad \forall j \in J, r \in \{1, \dots, |O_j| - 1\}$$
$$X_{vi} + p_{vi} \leq X_{qi} + M(1 - Y_{v,q,i}) \qquad \forall i \in M, \forall v, q \in J | q \neq v$$
$$X_{qi} + p_{qi} \leq X_{vi} + M(Y_{v,q,i}) \qquad \forall i \in M, \forall v, q \in J | q \neq v$$
$$X_{j,O_{|O_j|}^j} + p_{j,O_{|O_j|}^j} - d_j \leq T_{max} \qquad \forall j \in J$$
$$X_{ji} \geq 0 \qquad \forall\, j \in J, i \in M$$

$$Y_{vqi} \in 0,1 \qquad \forall v, q \in J |\ q \neq v, \forall i \in M$$

El modelo anterior se encuentra en la carpeta anexa en la siguiente ruta **[. *scripts/Problema 3/d.ipynb*]**.

Minimizar la máxima anticipación

Para minimizar la máxima demora, se tomará la siguiente formulación matemática,

Sets

$J: Set\ of\ jobs$

$M: Set\ of\ Machines$

Information

$O_j: set\ of\ operations\ for\ a\ job\ j \in J, and\ \left\{O_1^j, O_2^j, \dots, O_{|O_j|}^j,\right\} is\ the\ processign\ order\ for\ j \in J$

$p_{ij}: processing\ time\ of\ job\ j \in J\ on\ machine\ i \in M$

Decision variables

$X_{ji}: starting\ time\ job\ j \in J\ on\ machine\ i \in M$

$$Y_{vqi} = \left\{ \begin{matrix} 1\ if\ job\ v \in J\ precedes\ job\ q \in J | v \neq q, on\ machine\ i \in M \\ 0\ otherwise \end{matrix} \right\}$$

$E_{max}: Maximum\ Earliness\ of\ the\ schedule\ edition$

Formulation

$$F5| \quad |E_{max}$$

$$\min E_{max}$$

s.t,

$$\begin{aligned}
X_{j,O_r^j} + p_{j,O_r^j} &\leq X_{j,O_{r+1}^j} && \forall j \in J, r \in \{1, \dots, |O_j| - 1\} \\
X_{vi} + p_{vi} &\leq X_{qi} + M(1 - Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
X_{qi} + p_{qi} &\leq X_{vi} + M(Y_{v,q,i}) && \forall i \in M, \forall v, q \in J | q \neq v \\
d_j - X_{j,O_{|o_j|}^j} - p_{j,O_{|o_j|}^j} &\leq E_{max} && \forall j \in J
\end{aligned}$$

$$X_{ji} \geq 0 \qquad \forall\, j \in J, i \in M$$
$$Y_{vqi} \in 0{,}1 \qquad \forall v, q \in J \mid q \neq v, \forall i \in M$$

El modelo anterior se encuentra en la carpeta anexa en la siguiente ruta **[*. scripts/Problema 3/e.ipynb*]**.

Secuencias de los modelos

Para la secuencia de los modelos se hace uso de la librería plotly.express de Python, en donde para comenzar se tomó como fecha de partida el inicio del 20 de marzo. En efecto, se hace la suma teniendo como base que los tiempos de procesamiento se encuentra en horas. Para poder visualizarlos en diagrama de Gantt, se pueden ver en la parte de Anexos como Ilustración 4, Ilustración 5, Ilustración 6, Ilustración 7, Ilustración 8 respectivo a (Minimizar el Makespan, Minimizar el Mean Flow Time, Minimizar las desviaciones respecto a la fecha de entrega, Minimizar la máxima demora, Minimizar la máxima anticipación).

De forma numérica, según los resultados en Python se encuentra en la

Tabla 5: Resultados en Python.

Function = Cmax result = 87.0	Function = C_J1 + C_J2 + C_J3 + C_J4 + C_J5 result = 405.0
J1 . M1 . S= 0.0 C= 12.0	J1 . M1 . S= 0.0 C= 12.0
J1 . M2 . S= 22.0 C= 30.0	J1 . M2 . S= 12.0 C= 20.0
J1 . M4 . S= 30.0 C= 48.0	J1 . M4 . S= 20.0 C= 38.0
J1 . M5 . S= 53.0 C= 60.0	J1 . M5 . S= 45.0 C= 52.0
J1 . M6 . S= 60.0 C= 72.0	J1 . M6 . S= 52.0 C= 64.0
J1 . M7 . S= 72.0 C= 84.0	J1 . M7 . S= 64.0 C= 76.0
J2 . M6 . S= 0.0 C= 9.0	J2 . M6 . S= 0.0 C= 9.0
J2 . M3 . S= 12.0 C= 18.0	J2 . M3 . S= 12.0 C= 18.0
J2 . M5 . S= 18.0 C= 32.0	J2 . M5 . S= 18.0 C= 32.0
J2 . M7 . S= 42.0 C= 48.0	J2 . M7 . S= 32.0 C= 38.0
J2 . M2 . S= 48.0 C= 59.0	J2 . M2 . S= 38.0 C= 49.0
J2 . M1 . S= 71.0 C= 87.0	J2 . M1 . S= 49.0 C= 65.0
J3 . M3 . S= 0.0 C= 12.0	J3 . M3 . S= 0.0 C= 12.0
J3 . M7 . S= 12.0 C= 26.0	J3 . M7 . S= 12.0 C= 26.0
J3 . M6 . S= 26.0 C= 40.0	J3 . M6 . S= 26.0 C= 40.0
J3 . M1 . S= 40.0 C= 43.0	J3 . M1 . S= 40.0 C= 43.0
J3 . M5 . S= 45.0 C= 53.0	J3 . M5 . S= 52.0 C= 60.0
J3 . M4 . S= 70.0 C= 84.0	J3 . M4 . S= 70.0 C= 84.0
J4 . M5 . S= 0.0 C= 9.0	J4 . M5 . S= 0.0 C= 9.0

J4 . M2 . S= 9.0 C= 22.0	J4 . M2 . S= 20.0 C= 33.0
J4 . M7 . S= 26.0 C= 42.0	J4 . M7 . S= 38.0 C= 54.0
J4 . M4 . S= 48.0 C= 56.0	J4 . M4 . S= 56.0 C= 64.0
J4 . M1 . S= 56.0 C= 71.0	J4 . M1 . S= 65.0 C= 80.0
J4 . M6 . S= 72.0 C= 87.0	J4 . M6 . S= 80.0 C= 95.0
J5 . M7 . S= 0.0 C= 11.0	J5 . M7 . S= 0.0 C= 11.0
J5 . M1 . S= 22.0 C= 32.0	J5 . M1 . S= 22.0 C= 32.0
J5 . M5 . S= 32.0 C= 45.0	J5 . M5 . S= 32.0 C= 45.0
J5 . M3 . S= 45.0 C= 64.0	J5 . M3 . S= 45.0 C= 64.0
J5 . M4 . S= 64.0 C= 70.0	J5 . M4 . S= 64.0 C= 70.0
J5 . M2 . S= 70.0 C= 85.0	J5 . M2 . S= 70.0 C= 85.0

. Function = 2*E_J1 + 2*E_J2 + 2*E_J3 + 2*E_J4 + 2*E_J5 + 5*T_J1 + 5*T_J2 + 5*T_J3 + 5*T_J4 + 5*T_J5 result = 290.0

J1 . M1 . S= 0.0 C= 12.0
J1 . M2 . S= 22.0 C= 30.0
J1 . M4 . S= 30.0 C= 48.0
J1 . M5 . S= 53.0 C= 60.0
J1 . M6 . S= 60.0 C= 72.0
J1 . M7 . S= 72.0 C= 84.0
J2 . M6 . S= 0.0 C= 9.0
J2 . M3 . S= 12.0 C= 18.0
J2 . M5 . S= 18.0 C= 32.0
J2 . M7 . S= 42.0 C= 48.0
J2 . M2 . S= 48.0 C= 59.0
J2 . M1 . S= 71.0 C= 87.0
J3 . M3 . S= 0.0 C= 12.0
J3 . M7 . S= 12.0 C= 26.0
J3 . M6 . S= 26.0 C= 40.0
J3 . M1 . S= 40.0 C= 43.0
J3 . M5 . S= 45.0 C= 53.0
J3 . M4 . S= 56.0 C= 70.0
J4 . M5 . S= 0.0 C= 9.0
J4 . M2 . S= 9.0 C= 22.0
J4 . M7 . S= 26.0 C= 42.0
J4 . M4 . S= 48.0 C= 56.0
J4 . M1 . S= 56.0 C= 71.0
J4 . M6 . S= 72.0 C= 87.0
J5 . M7 . S= 0.0 C= 11.0
J5 . M1 . S= 22.0 C= 32.0
J5 . M5 . S= 32.0 C= 45.0
J5 . M3 . S= 51.0 C= 70.0
J5 . M4 . S= 70.0 C= 76.0
J5 . M2 . S= 76.0 C= 91.0

Function = Emax result = 0.0	Function = Tmax result = 17.0
J1 . M1 . S= 21.0 C= 33.0	J1 . M1 . S= 0.0 C= 12.0
J1 . M2 . S= 56.0 C= 64.0	J1 . M2 . S= 12.0 C= 20.0
J1 . M4 . S= 64.0 C= 82.0	J1 . M4 . S= 20.0 C= 38.0
J1 . M5 . S= 82.0 C= 89.0	J1 . M5 . S= 38.0 C= 45.0
J1 . M6 . S= 99.0 C= 111.0	J1 . M6 . S= 45.0 C= 57.0
J1 . M7 . S= 111.0 C= 123.0	J1 . M7 . S= 57.0 C= 69.0
J2 . M6 . S= 0.0 C= 9.0	J2 . M6 . S= 0.0 C= 9.0
J2 . M3 . S= 12.0 C= 18.0	J2 . M3 . S= 12.0 C= 18.0

J2 . M5 . S= 43.0 C= 57.0	J2 . M5 . S= 24.0 C= 38.0
J2 . M7 . S= 58.0 C= 64.0	J2 . M7 . S= 49.0 C= 55.0
J2 . M2 . S= 64.0 C= 75.0	J2 . M2 . S= 68.0 C= 79.0
J2 . M1 . S= 80.0 C= 96.0	J2 . M1 . S= 79.0 C= 95.0
J3 . M3 . S= 0.0 C= 12.0	J3 . M3 . S= 0.0 C= 12.0
J3 . M7 . S= 12.0 C= 26.0	J3 . M7 . S= 12.0 C= 26.0
J3 . M6 . S= 26.0 C= 40.0	J3 . M6 . S= 26.0 C= 40.0
J3 . M1 . S= 40.0 C= 43.0	J3 . M1 . S= 40.0 C= 43.0
J3 . M5 . S= 57.0 C= 65.0	J3 . M5 . S= 45.0 C= 53.0
J3 . M4 . S= 96.0 C= 110.0	J3 . M4 . S= 57.0 C= 71.0
J4 . M5 . S= 34.0 C= 43.0	J4 . M5 . S= 0.0 C= 9.0
J4 . M2 . S= 43.0 C= 56.0	J4 . M2 . S= 20.0 C= 33.0
J4 . M7 . S= 64.0 C= 80.0	J4 . M7 . S= 33.0 C= 49.0
J4 . M4 . S= 88.0 C= 96.0	J4 . M4 . S= 49.0 C= 57.0
J4 . M1 . S= 96.0 C= 111.0	J4 . M1 . S= 57.0 C= 72.0
J4 . M6 . S= 111.0 C= 126.0	J4 . M6 . S= 72.0 C= 87.0
J5 . M7 . S= 0.0 C= 11.0	J5 . M7 . S= 0.0 C= 11.0
J5 . M1 . S= 11.0 C= 21.0	J5 . M1 . S= 30.0 C= 40.0
J5 . M5 . S= 21.0 C= 34.0	J5 . M5 . S= 53.0 C= 66.0
J5 . M3 . S= 34.0 C= 53.0	J5 . M3 . S= 66.0 C= 85.0
J5 . M4 . S= 82.0 C= 88.0	J5 . M4 . S= 85.0 C= 91.0
J5 . M2 . S= 88.0 C= 103.0	J5 . M2 . S= 91.0 C= 106.0

Tener en cuenta que para el caso del Mean Flow time se utiliza la suma de los tiempos de finalización de cada trabajo, siendo como resultado 405 y el $MFT = \frac{405}{5} = 81$

Conclusiones

Para poder establecer comparaciones, se podría iniciar desde el Makespan como parámetro homogéneo. Del cual, es posible observar que el mayor es el arrojado por la anticipación máxima, esto se debe a que se está forzando la secuencia para que en primera instancia los trabajo es entreguen después de la fecha de entrega d. Luego, al maximizar se está planeando la secuencia que mas tarde en terminar todos los trabajos.

Para las desviaciones con respecto a la fecha de entrega se encuentra que el Makespan es menor a la minimización de la máxima demora, pero mayor al optimo que da hallando el C_{max}. Por lo cual, se podría interpretar como una disminución de la tardanza por su mayor peso con respecto a la anticipación. Por otra parte, cuando se hace la minimización de los tiempos de finalización de

los trabajos se encuentra que hay posibilidad de reducir el tiempo promedio de flujo aumentando en 8 unidades el Makespan.

Código en Python

Información general

```
%pip install pulp
from pulp import *
# Definimos el problema
# Jobshop para 7 máquina y 5 trabajos
J=['J1','J2','J3','J4','J5']
M=['M1','M2','M3','M4','M5','M6','M7']
O=[['M1','M2','M4','M5','M6','M7'],
   ['M6','M3','M5','M7','M2','M1'],
   ['M3','M7','M6','M1','M5','M4'],
   ['M5','M2','M7','M4','M1','M6'],
   ['M7','M1','M5','M3','M4','M2']]
# Tiempos de procesamiento [horas]
p=[[12,8,0,18,7,12,12],
   [16,11,6,0,14,9,6],
   [3,0,12,14,8,14,14],
   [15,13,0,8,9,15,16],
   [10,15,19,6,13,0,11]]
# due dates [horas]
d=[[54],[78],[54],[85],[90]]
# Creación de diccionarios
d=makeDict([J],d)
p=makeDict([J,M],p)
O=makeDict([J,range(6)],O)
```

Makespan

```
problema=LpProblem('Jobshop',LpMinimize)
X=LpVariable.dicts('X',((j,i) for j in J for i in M ),lowBound=0,cat='Cont
inuos')
Y=LpVariable.dicts('Y',((v,q,i) for v in J for q in J for i in M),lowBound
=0, upBound=1,cat='Binary')
Cmax=LpVariable('Cmax', lowBound=0, cat='Continuos')

for j in J:
    h=O[j][len(O[j])-1]
    problema+=X[j,h]+p[j][h]<=Cmax,("makespan",j)
```

```
for j in J:
  for r in range(len(O[j])-1):
      h=O[j][r]
      h1=O[j][r+1]
      problema+=X[j,h]+p[j][h]<=X[j,h1]

MM=1000
for i in M:
  for v in J:
    for q in J:
      if(q!=v):
          problema+=X[v,i]+p[v][i]<=X[q,i]+MM*(1-Y[v,q,i])
          problema+=X[q,i]+p[q][i]<=X[v,i]+MM*(Y[v,q,i])

problema+=Cmax

problema.writeLP('Makespan')

problema.solve()
print(LpStatus[problema.status])
print("Function = ", problema.objective, "result = ", value(problema.objec
tive))
for j in J:
    for r in O[j]:
        i=O[j][r]
        print(j,".",i,".","S=", value(X[j,i])," C=",value(X[j,i]+p[j][i])
 )
```

Suma de C[j]

```
# Minimizar la sumatoria de Cj
problema=LpProblem('Jobshop',LpMinimize)
X=LpVariable.dicts('X',((j,i) for j in J for i in M ),lowBound=0,cat='Cont
inuos')
Y=LpVariable.dicts('Y',((v,q,i) for v in J for q in J for i in M),lowBound
=0, upBound=1,cat='Binary')
C=LpVariable.dicts('C',(j for j in J),lowBound=0,cat='Continuos')

for j in J:
    h=O[j][len(O[j])-1]
    problema+=X[j,h]+p[j][h]<=C[j], ("Completation time of job %s" % j)

for j in J:
  for r in range(len(O[j])-1):
      h=O[j][r]
```

```
      h1=O[j][r+1]
      problema+=X[j,h]+p[j][h]<=X[j,h1]

MM=1000
for i in M:
  for v in J:
    for q in J:
      if(q!=v):
          problema+=X[v,i]+p[v][i]<=X[q,i]+MM*(1-Y[v,q,i])
          problema+=X[q,i]+p[q][i]<=X[v,i]+MM*(Y[v,q,i])
# Minimizar la sumatoria de Cj
problema+= lpSum(C[j] for j in J), "sum of completion times"

problema.writeLP('b.lp')

problema.solve()
# Imprimir el estado de la solución
print("Status:", LpStatus[problema.status])
# Imprimir el valor de la función objetivo
print("Makespan = ", value(problema.objective))
# Imprimir la fución objetivo
print("Function = ", problema.objective, "result = ", value(problema.objec
tive))
for j in J:
    for r in O[j]:
        i=O[j][r]
        print(j,".",i,".","S=", value(X[j,i])," C=",value(X[j,i]+p[j][i])
 )
```

Desviaciones con respecto a la fecha de entrega

```
# Minimizar las desviaciones respecto a la fecha de entrega,
# suponga el costo de anticipación es de 2$/hora, y de entrega tardía de 5
$/hora
problema=LpProblem('Jobshop',LpMinimize)
X=LpVariable.dicts('X',((j,i) for j in J for i in M ),lowBound=0,cat='Cont
inuos')
Y=LpVariable.dicts('Y',((v,q,i) for v in J for q in J for i in M),lowBound
=0, upBound=1,cat='Binary')
E=LpVariable.dicts('E',(j for j in J),lowBound=0,cat='Continuos')
T=LpVariable.dicts('T',(j for j in J),lowBound=0,cat='Continuos')

for j in J:
    h=O[j][len(O[j])-1]
    problema+=X[j,h]+p[j][h]-T[j]+E[j]==d[j]
```

```
for j in J:
  for r in range(len(O[j])-1):
      h=O[j][r]
      h1=O[j][r+1]
      problema+=X[j,h]+p[j][h]<=X[j,h1]

MM=1000
for i in M:
  for v in J:
    for q in J:
      if(q!=v):
          problema+=X[v,i]+p[v][i]<=X[q,i]+MM*(1-Y[v,q,i])
          problema+=X[q,i]+p[q][i]<=X[v,i]+MM*(Y[v,q,i])

problema+= 2*lpSum(E[j] for j in J)+5*lpSum(T[j] for j in J)

problema.writeLP('c.lp')

problema.solve()
print(LpStatus[problema.status])
print("Function = ", problema.objective, "result = ", value(problema.objec
tive))
for j in J:
    for r in O[j]:
        i=O[j][r]
        print(j,".",i,".","S=", value(X[j,i])," C=",value(X[j,i]+p[j][i])
 )
```

Máxima demora

```
problema=LpProblem('Jobshop',LpMinimize)
X=LpVariable.dicts('X',((j,i) for j in J for i in M ),lowBound=0,cat='Cont
inuos')
Y=LpVariable.dicts('Y',((v,q,i) for v in J for q in J for i in M),lowBound
=0, upBound=1,cat='Binary')
Tmax=LpVariable('Tmax', lowBound=0, cat='Continuos')

for j in J:
    h=O[j][len(O[j])-1]
    problema+=X[j,h]+p[j][h]-d[j]<=Tmax,("Meta",j)

for j in J:
  for r in range(len(O[j])-1):
      h=O[j][r]
      h1=O[j][r+1]
      problema+=X[j,h]+p[j][h]<=X[j,h1]
```

```
MM=1000
for i in M:
  for v in J:
    for q in J:
      if(q!=v):
          problema+=X[v,i]+p[v][i]<=X[q,i]+MM*(1-Y[v,q,i])
          problema+=X[q,i]+p[q][i]<=X[v,i]+MM*(Y[v,q,i])

problema+=Tmax

problema.writeLP('d.lp')

problema.solve()
print(LpStatus[problema.status])
print("Function = ", problema.objective, "result = ", value(problema.objec
tive))
for j in J:
    for r in O[j]:
        i=O[j][r]
        print(j,".",i,".","S=", value(X[j,i])," C=",value(X[j,i]+p[j][i])
 )
```

Máxima anticipación

```
problema1=LpProblem('Jobshop',LpMinimize)
X=LpVariable.dicts('X',((j,i) for j in J for i in M ),lowBound=0,cat='Cont
inuos')
Y=LpVariable.dicts('Y',((v,q,i) for v in J for q in J for i in M),lowBound
=0, upBound=1,cat='Binary')
Emax=LpVariable('Emax', lowBound=0, cat='Continuos')

for j in J:
    h=O[j][len(O[j])-1]
    problema1+=d[j]-X[j,h]-p[j][h]<=Emax

for j in J:
  for r in range(len(O[j])-1):
      h=O[j][r]
      h1=O[j][r+1]
      problema1+=X[j,h]+p[j][h]<=X[j,h1]

MM=1000
for i in M:
  for v in J:
    for q in J:
```

```
        if(q!=v):
            problema1+=X[v,i]+p[v][i]<=X[q,i]+MM*(1-Y[v,q,i])
            problema1+=X[q,i]+p[q][i]<=X[v,i]+MM*(Y[v,q,i])
```

Diagrama de Gantt para todas

```
from pandas.core.arrays.datetimelike import timedelta
import datetime
from datetime import datetime
import plotly.express as px
import pandas as pd
df=[]
# Crear un DataFrame con los datos de las tareas
for j in J:
    for r in O[j]:
        i=O[j][r]
        date=datetime(2023, 3, 20, 0, 0, 0)
        Start_date= date+timedelta(days=value(X[j,i])/24)
        Finish_date= date+timedelta(days=value(X[j,i]+p[j][i])/24)
        Data=dict(Job=j, Start=Start_date, Finish=Finish_date, Machines=i)
        df.append(Data)

df=pd.DataFrame(df)
# Crear el diagrama de Gantt
fig = px.timeline(df, x_start="Start", x_end="Finish", y="Machines", color
="Job")

# Mostrar el diagrama de Gantt
fig.show()
```

Anexos

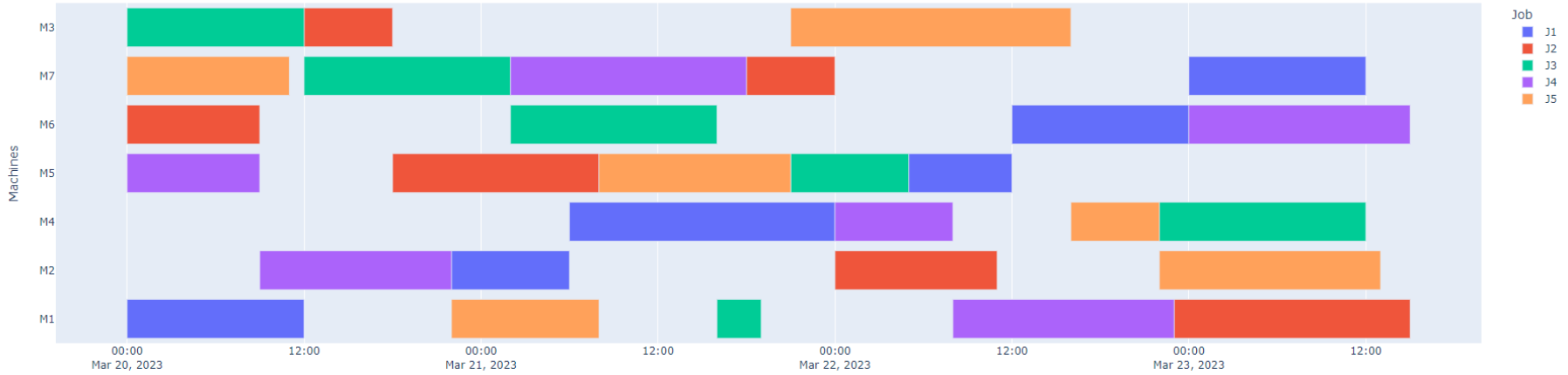


Ilustración 4:Secuencia para Cmax

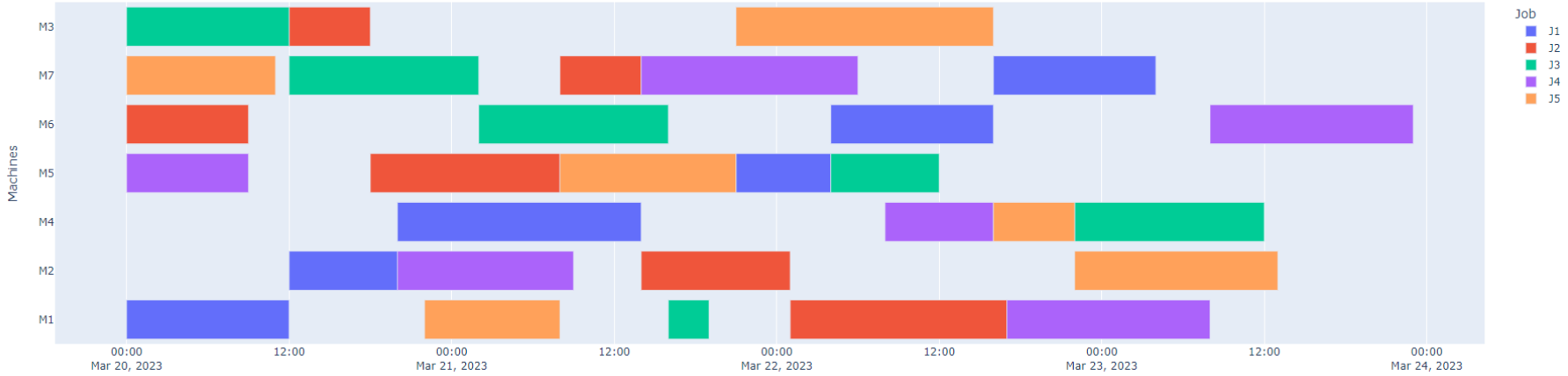


Ilustración 5: Secuencia para la suma de los tiempos de completado

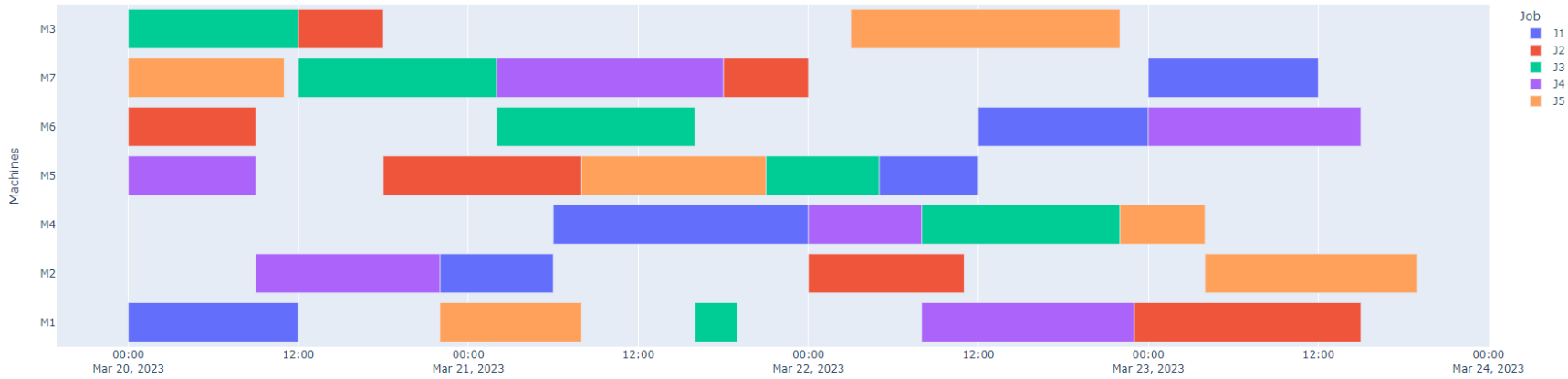


Ilustración 6: Secuencia para desviaciones con respecto a la fecha de entrega.

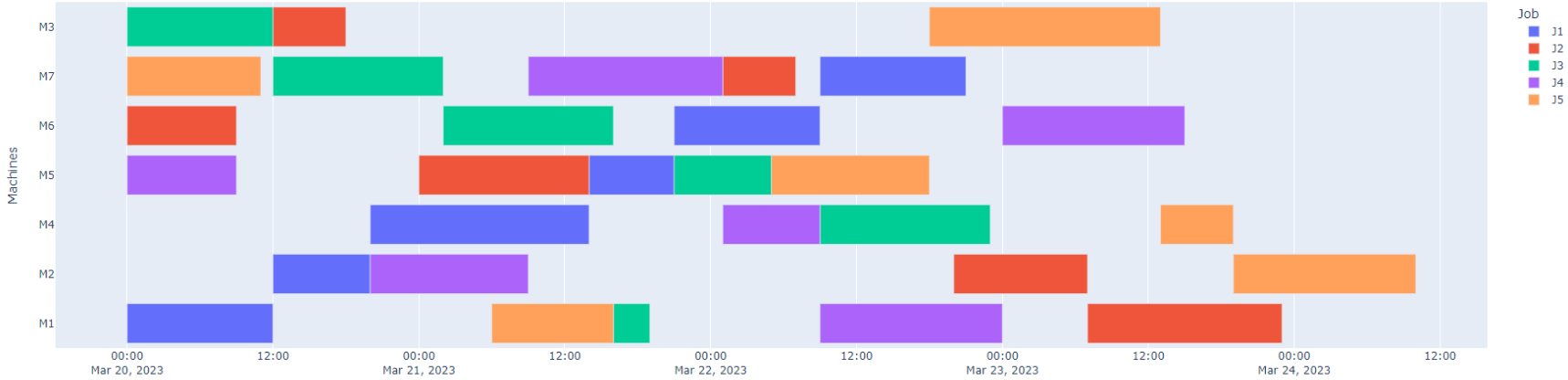


Ilustración 7: Secuencia para máxima demora.

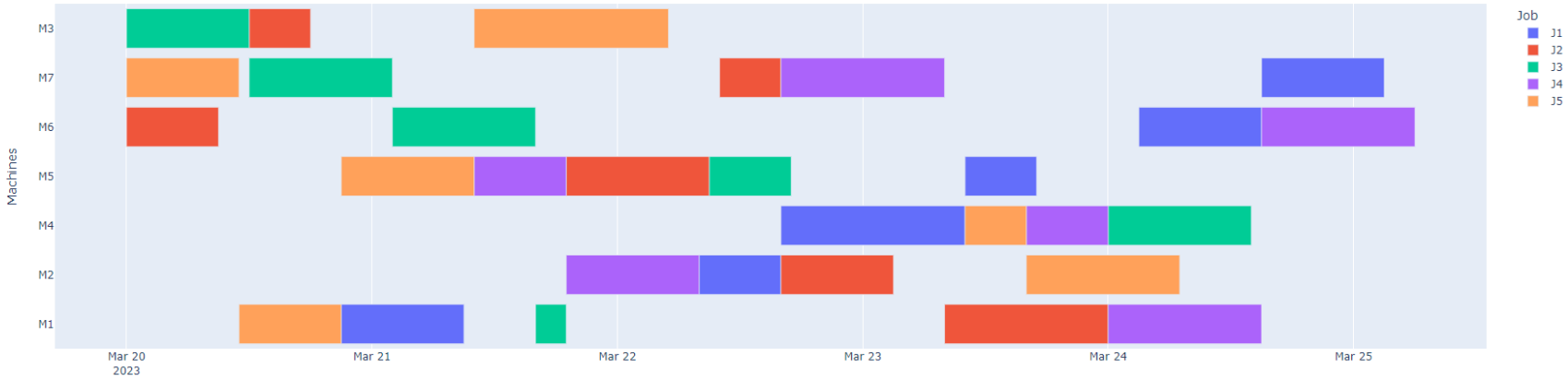


Ilustración 8: Secuencia para anticipación máxima.