Tezan Sahu
170100035
tezansahu@iitb.ac.in

Assignment 4
ME714: Computer Integrated Manufacturing

Due Date: 11/04/21

Question 1 *(Completed Individually)*

Following is the python code to solve the $n/2$ shop floor scheduling problem using *Johnson's algorithm*:

```python
import numpy as np

# Function to get the idle time Xi on machine B for all jobs
def get_idle_time(data, optimal_seq):
    # Store the Ai's & Bi's from the given data
    a = [data[0][i-1] for i in optimal_seq]
    b = [data[1][i-1] for i in optimal_seq]

    # This array will store the idle times on machine B (Xi) for each job i
    x = []

    for i in range(data.shape[1]):
        x.append(max(sum(a[:i+1]) - sum(b[:i]) - sum(x[:i]), 0.0))

    return x

# Function to obtain the optimal sequence for a n/2 scheduling problem using Johnson's algorithm
def johnsons_algo(x):
    # Create a placeholder to store the optimal sequence of jobs
    optimal_seq = np.zeros(x.shape[1], dtype=int)

    front = 0
    back = x.shape[1] - 1

    # Obtain the indices of the elememnts of the matrix after sorting in ascending order.
    # These indices will be used in scheduling
    sorted_indices = np.argsort(x.flatten())

    # Repeat the scheduling process until all the jobs are scheduled
    i = 0
    while front <= back:
        smallest_elem_index = np.unravel_index(sorted_indices[i], x.shape)

        # Select the job with the smallest Ai or Bi
        candidate_job = smallest_elem_index[1] + 1

        # Since the way we select the next smallest Ai or Bi does not guarantee that
        # a 'unique' (unscheduled) job is selected, we perform an extra check to include
        # this job only if it doesn't already exist in the array containing our optimal sequence
        if candidate_job not in optimal_seq:
            # If the smallest value belongs to machine A, add the Job to start of scheduling
            if smallest_elem_index[0] == 0:
                optimal_seq[front] = candidate_job
                front += 1
            # If the smallest value belongs to machine B, add the Job to end of scheduling
            elif smallest_elem_index[0] == 1:
                optimal_seq[back] = candidate_job
                back -= 1

        i += 1

    return optimal_seq


def shop_floor_scheduling(schedule_file):
    x = np.genfromtxt(schedule_file, delimiter=',')
    opt_seq = johnsons_algo(x)
    idle_time = get_idle_time(x, opt_seq)


    print("Optimal Sequence of Jobs:\t", *opt_seq)
    print("Idle Time Xi on Machine B for jobs:")
    for i, time in enumerate(idle_time):
        print(f"\tX{i+1} = {time}")
    print("Total idle time on Machine B:\t", sum(idle_time))
    print("Total processing time:\t\t", sum(idle_time) + sum(x[1][:]))



def main():
    shop_floor_scheduling("schedule.csv")

if __name__ == "__main__":
    main()
```

Listing 1: `a4_q1.py` Python Script to implement Johnson's Algorithm

The input to this file is a CSV file named `schedule.csv`, which contains a $2 \times n$ matrix in which an element at row i and column j depicts the duration of job j on machine i.

The printed outputs include:

- The optimal sequence of jobs
- Idle Time X_i on Machine B for jobs
- Total idle time on Machine B $= \sum_i X_i$
- Total processing time $= \sum_i (B_i + X_i)$

As indicated, we test the above code with the following input:

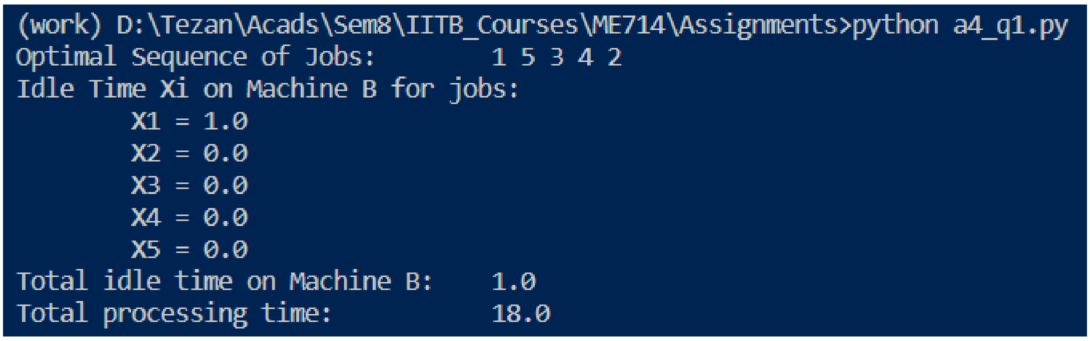
	Job 1	Job 2	Job 3	Job 4	Job 5
Machine A	1	3	2	4	1
Machine B	4	2	6	2	3

$n/2$ example problem with $n = 5$

```
1,3,2,4,1
4,2,6,2,3
```

`schedule.csv` file to be used as input

The following result is obtained on running the script with the above mentioned `schedule.csv` file as input:



```
(work) D:\Tezan\Acads\Sem8\IITB_Courses\ME714\Assignments>python a4_q1.py
Optimal Sequence of Jobs:        1 5 3 4 2
Idle Time Xi on Machine B for jobs:
        X1 = 1.0
        X2 = 0.0
        X3 = 0.0
        X4 = 0.0
        X5 = 0.0
Total idle time on Machine B:    1.0
Total processing time:           18.0
```

Figure 1: Output obtained by running the script `a4_q1.py` for the above mentioned input

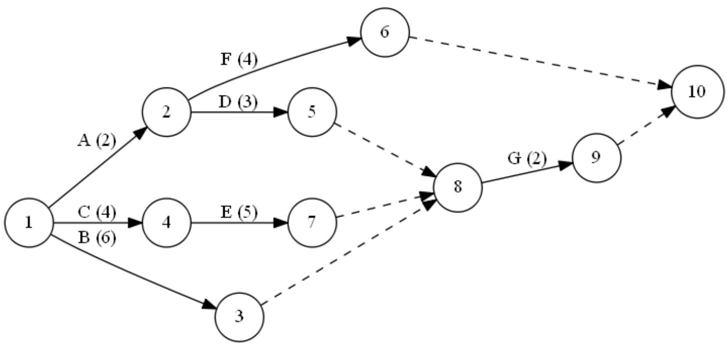
The actual python script can be accessed here.

Question 2

The table below show the project activities with their respective duration (in days). The figure alongside depicts the *preliminary* **activity-on-arrow** type graph produced by using the table.

Activity	Predecessor	Duration
A	-	2
B	-	6
C	-	4
D	A	3
E	C	5
F	A	4
G	B, D, E	2

Project Activities



Preliminary Activity-On-Arrow Type Graph

By eliminating some of the *dummy* elements from the above graph, we simplify the activity-on-arrow type graph as follows:

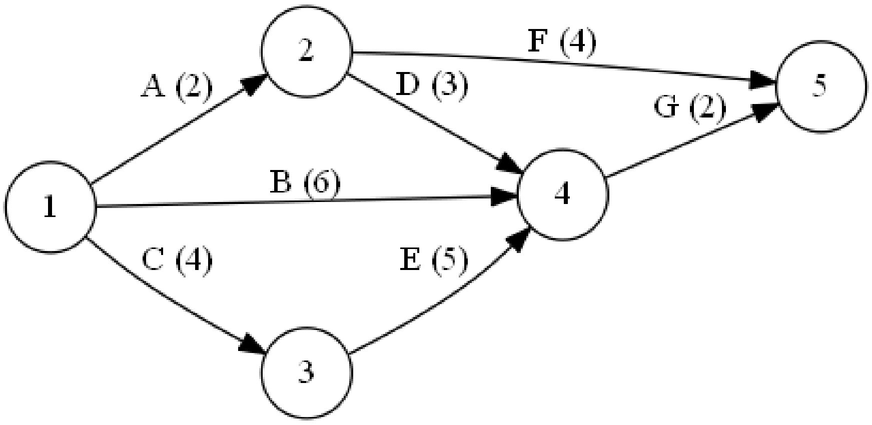


Figure 2: Simplified Activity-On-Arrow Type Graph

For every node 1-5 in the simplified graph, we obtain the *Early Start Time (ES)* & *Latest Finish Time (LF)* using the formulae mentioned below & tabulate them as follows:

$$ES_i = max(ES_{j-1} + D_j)$$
$$LF_i = min(ES_{j+1} - D_{j+1})$$

Node	Early Start Time (ES)	Latest Finish Time (LF)
1	0	$min(6-2, 9-6, 4-4) = 0$
2	$max(0+2) = 2$	$min(11-4, 9-3) = 6$
3	$max(0+4) = 4$	$min(9-5) = 4$
4	$max(2+3, 0+6, 4+5) = 9$	$min(11-2) = 9$
5	$max(2+4, 9+2) = 11$	11

Table 1: Early Start & Latest Finish Times for the various nodes in the simplified graph

On the **Critical Path**, we have $ES_i = LF_i$. From the table above, this corresponds to Nodes 1, 3, 4 & 5. Thus, the critical path includes activities **C**, **E** & **G**.

The **total duration of the project** is the sum of the duration of the activities on the critical path:

$$Total\ Project\ Duration = D_C + D_E + D_G = 4 + 5 + 2 = 11\ days$$

Now, we calculate & tabulate the *slack* for each activity as follows:

$$Slack_a = LF_{i+1} - ES_i - D_a\ where\ activity\ a\ goes\ from\ Node\ i\ to\ Node\ (i+1)$$

Activity	Nodes	Duration	Slack
A	1, 2	2	$6-0-2=4$
B	1, 4	6	$9-0-6=3$
C	1, 3	4	$4-0-4=0$
D	2, 4	3	$9-2-3=4$
E	3, 4	5	$9-4-5=0$
F	2, 5	4	$11-2-4=5$
G	4, 5	2	$11-9-2=0$

Table 2: Slack for Activities A-G during the Project

Clearly, we observe that the activities that lie on the Critical Path (C, E & G) have $Slack = 0$ necessarily, i.e., they cannot be delayed without delaying the overall project.

Note: *The code for generating the 2 graphs above can be found here.*

Question 3

- Number of samples (k) = 10
- Number of parts per sample (n) = 5

For the given data, we compute the $\bar{x}$ & R values as follows:

Sample No.	Collection Time	x_1	x_2	x_3	x_4	x_5	$\bar{x} = \sum_{i=1}^{5} x_i/5$	$R = Range(x_i)$
1	08:00 AM	202	206	204	206	206	204.8	4
2	09:00 AM	208	204	208	207	205	206.4	4
3	10:00 AM	208	206	207	205	208	206.8	3
4	11:00 AM	203	203	207	206	203	204.4	4
5	12:00 PM	201	204	209	208	204	205.2	8
6	01:00 PM	208	204	200	209	201	204.4	9
7	02:00 PM	207	203	200	202	206	203.6	7
8	03:00 PM	202	210	206	202	200	204.0	10
9	04:00 PM	207	208	200	202	210	205.4	10
10	05:00 PM	210	203	203	208	207	206.2	7

Table 3: Computation of $\bar{x}$ & R values for the various samples

Now, we compute the values of $\bar{\bar{x}}$ & $\bar{R}$ as follows:

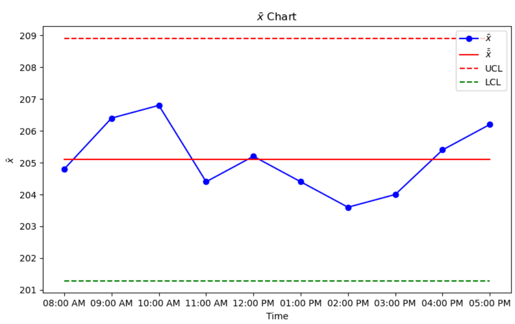
$$\bar{\bar{x}} = \frac{\sum_{k=1}^{10} \bar{x}_k}{10} = \frac{204.8 + 206.4 + 206.8 + 204.4 + 205.2 + 204.4 + 203.6 + 204.0 + 205.4 + 206.2}{10} = 205.1$$

$$\bar{R} = \frac{\sum_{k=1}^{10} R_k}{10} = \frac{4 + 4 + 3 + 4 + 8 + 9 + 7 + 10 + 10 + 7}{10} = 6.6$$

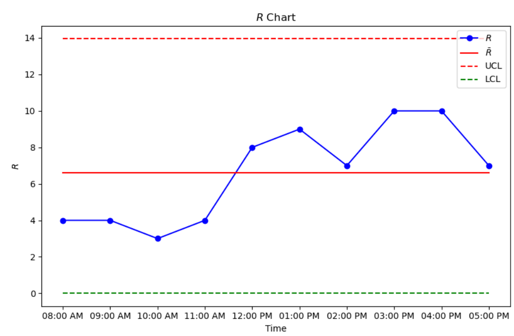
Since $n = 5$, we use $d_2 = 2.326$, $A_2 = 0.577$, $D_3 = 0$ & $D_4 = 2.114$ for calculation of the Upper & Lower Control Limits as follows:

- Control Limits for $\bar{x}$ chart:
  - $UCL_{\bar{x}} = \bar{\bar{x}} + A_2\bar{R} = 205.1 + (0.577 \times 6.6) = 208.91$
  - $LCL_{\bar{x}} = \bar{\bar{x}} - A_2\bar{R} = 205.1 - (0.577 \times 6.6) = 201.29$
- Control Limits for R chart:
  - $UCL_R = D_3\bar{R} = 0 \times 6.6 = 0$
  - $LCL_R = D_4\bar{R} = 2.114 \times 6.6 = 13.95$

The **Control Charts** for this data are shown below:



(a) $\bar{x}$ Chart



(b) R Chart

Clearly, all the points are within the respective control limits. Hence, we can conclude that manufacturing process is **in-control**. Moreover, since there are no deviations outside the control limits, we do not need to remove any points or revise the charts.

Note: *The code for generating the $\bar{x}$ & R control charts can be found here.*

Question 4

Given, required diameter of the ball (D) = 2.83 inches
Actual diameter of the balls made is uniformly distributed over the range of 2.75 inches to 2.90 inches. Thus, PDF $f(x)$ is given by:

$$\begin{cases} \frac{1}{0.15} & x \in [2.75, 2.90] \\ 0 & otherwise \end{cases}$$

i. Quadratic Loss Function

Let quadratic loss function be $L(x) = k(x - m)^2$

We know that $L(2.83) = 0 \Rightarrow m = 2.83$
Also, $L(2.90) = 10 \Rightarrow k(2.90 - 2.83)^2 = 10 \Rightarrow k = \frac{10}{0.07^2} = 2040.82$

Thus, loss function $L(x) = 2040.82(x - 2.83)^2$

We need to obtain the expected loss (average loss per ball), i.e., $\mathbb{E}[L(x)]$

$$\mathbb{E}[L(x)] = \int_x L(t)f(t)dt$$

For quadratic loss function $L(x)$ of the form $k(x - m)^2$, this simplifies to:

$$\mathbb{E}[L(x)] = k[s^2 + (\bar{x} - m)^2]$$

For the given *uniform distribution* for balls manufactured:

- $\bar{x} = \frac{2.90+2.75}{2} = 2.825$
- $s = \sqrt{\frac{(2.90-2.75)^2}{12}} = 0.0433$

Thus, $\mathbb{E}[L(x)] = 2040.82 \cdot [0.0433^2 + (2.825 - 2.83)^2] = 3.877$

The average loss per ball to the company is **INR 3.877**. *[Ans]*

ii. Non-Quadratic Loss Function

Based on the given information, we introduce the profit function $P(x)$ such that, $P(x)$ is represented as:

$$\begin{cases} -50 & x < 2.80 \\ 100 & x \in [2.80, 2.86] \\ 10 & x > 2.86 \end{cases}$$

The expected profit $\mathbb{E}[P(x)]$ is given by:

$$\mathbb{E}[P(x)] = \int_x P(t)f(t)dt = \frac{1}{0.15}\int_{2.75}^{2.90} P(t)dt = \frac{1}{0.15}\left[\int_{2.75}^{2.80} -50dt + \int_{2.80}^{2.86} 100dt + \int_{2.86}^{2.90} 10dt\right]$$

$$\therefore \mathbb{E}[P(x)] = \frac{1}{0.15}[-50(2.80 - 2.75) + 100(2.86 - 2.80) + 10(2.90 - 2.86)] = 26$$

Thus, the **expected profit** per ball for the company is **INR 26** *[Ans]*