# 40.012 MANUFACTURING AND SERVICE OPERATIONS

## Assignment 4

**Michael Hoon**
1006617

June 12, 2024

# Question 1

## (a) Silver-Meal Heuristic

We first define the given parameters, namely:

- Holding cost $h = \$1$ per unit per month
- Setup cost $K = \$40$ per order
- Initial Inventory $= 4$, Ending Inventory $= 8$

The 12-month anticipated demand is given as

| **Month** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 6 | 12 | 4 | 8 | 15 | 25 | 20 | 5 | 10 | 20 | 5 | 12 |
| **Adjusted Demand** | 2 | 12 | 4 | 8 | 15 | 25 | 20 | 5 | 10 | 20 | 5 | 20 |

Table 1: Anticipated Demand over 12 Months

We now define $C(T)$ as the average holding and setup cost per period if the current order spans the next $T$ periods. The general formula for $j$ periods is given by

$$C(j) = (K + hr_2 + 2hr_3 + \cdots + (j-1)hr_j)/j \tag{1}$$

We start the iterative algorithm first with month 1, considering that we have a current inventory of 4:

$$\begin{aligned} C_1(1) &= 40 \\ C_1(2) &= [40 + 1(12)]\,/2 = 26 \\ C_1(3) &= [40 + 1(12) + 2(1)(4)]\,/3 = 20 \\ C_1(4) &= [40 + 1(12) + 2(1)(4) + 3(1)(8)]\,/4 = 21 \end{aligned}$$

We terminate here as $C(4) > C(3)$, and set $y_1 = r_1 + r_2 + r_3 = 2 + 12 + 4 = 18$, adding this into our optimal order policy $\mathbf{y} = \big(18\big)$. We note that there is a way to streamline the calculations using a recursive formula instead:

$$C(j+1) = \left[\frac{j}{j+1}\right][C(j) + hr_{j+1}] \tag{2}$$

We continue the iterations by setting month 4 as the starting point:

$$\begin{aligned} C_4(1) &= 40 \\ C_4(2) &= [40 + 1(15)]\,/2 = 27.5 \\ C_4(3) &= [27.5(2) + 1(2)(25)]\,/3 = 35 \end{aligned}$$

Similarly, we stop here as the value increased, corresponding to the $6^{\text{th}}$ month. We set $y_4 = r_4 + r_5 = 23$, adding into our optimal policy $\mathbf{y} = \begin{pmatrix}18 & 23\end{pmatrix}$. Since there are way too many manual calculations involved in this Silver-Meal heuristic, we can come up with a recursive algorithm in Python using equation 2 to iteratively obtain the values for us (refer to Appendix). From the code, we obtain the following values:

$$\begin{aligned} C_6(1) &= 40 \\ C_6(2) &= 30 \\ C_6(3) &= 23.33 \\ C_6(4) &= 25 \end{aligned}$$

We stop here as $C_6(4) > C_6(3)$ corresponding to month 9, adding into our policy $y_6 = r_6 + r_7 + r_8 = 25 + 20 + 5 = 50$, so $\mathbf{y} = \begin{pmatrix}18 & 23 & 50\end{pmatrix}$. Continuing from month 9, we have

$$\begin{aligned} C_9(1) &= 40 \\ C_9(2) &= 30 \\ C_9(3) &= 23.33 \\ C_9(4) &= 26.50 \end{aligned}$$

We stop here as $C_9(4) > C_9(3)$ corresponding to month 12, adding into our policy $y_9 = r_9 + r_{10} + r_{11} = 10 + 20 + 5 = 35$, so $\mathbf{y} = \begin{pmatrix}18 & 23 & 50 & 35\end{pmatrix}$. As period 12 is the final period in the horizon, we only need to consider the setup cost for that month, where

$$C_{12}(1) = 40$$

We set $y_{12} = r_{12} = 20$, to meet the ending inventory of 8 units. Thus, the optimal order policy for this item based on the Silver-Meal heuristic is $\mathbf{y} = \begin{pmatrix}18 & 23 & 50 & 35 & 20\end{pmatrix}$, where the order from month 1 to 3 is 18 units, month 4 to 5 is 23 units, 6 to 8 is 50 units, 9 to 11 is 35 units, and the final month is 20 units. We also consider the holding cost of the ending inventory of 8 units. This will correspond to a total cost of

$$\begin{aligned} C_{\text{total}} &= C_1(3) \times 3 + C_4(2) \times 2 + C_6(3) \times 3 + C_9(3) \times 3 + C_{12}(1) \times 1 + 8 \times 1 \\ &= 20 \times 3 + 27.5 \times 2 + 23.33 \times 3 + 23.33 \times 3 + 40 \times 1 + 8 \\ &= \boxed{\$303} \end{aligned}$$

## (b) Least Unit Cost

Now, we will use the Least Unit Cost heuristic to evaluate the optimal policy for TMEs. Similar to the Silver-Meal heuristic as before, we modify the iterative algorithm such that instead of dividing the cost over $j$ periods by the same number of periods $j$, we divide it by the total number of units demanded through period $j$, $r_1 + r_2 + \cdots + r_j$. We choose the order horizon that **minimises the cost per unit of demand** rather than the cost per period. The general formula here for $j$ periods is

$$C(j) = \left[K + hr_2 + hr_3 + \cdots + (j-1)hr_j\right] / (r_1 + r_2 + \cdots + r_j) \tag{3}$$

The costs for the first iteration using a similar algorithm in Python gives:

$$\begin{aligned}
C_1(1) &= 40/2 = 20 \\
C_1(2) &= (40 + (1)(12))/(2 + 12) = 3.71 \\
C_1(3) &= 3.33 \\
C_1(4) &= 3.23 \\
C_1(5) &= 3.51
\end{aligned}$$

Terminate here as $C_1(5) > C_1(4)$, where $y_1 = r_1 + \ldots r_4 = 26$, corresponding to month 5. Continuing the algorithm,

$$\begin{aligned}
C_5(1) &= 2.67 \\
C_5(2) &= 1.62 \\
C_5(3) &= 1.75
\end{aligned}$$

Terminate here as $C_5(3) > C_5(2)$, where $y_5 = r_5 + r_6 = 40$, corresponding to month 7. Continuing the algorithm,

$$\begin{aligned}
C_7(1) &= 2 \\
C_7(2) &= 1.8 \\
C_7(3) &= 1.86
\end{aligned}$$

Terminate here as $C_7(3) > C_7(2)$, where $y_7 = r_7 + r_8 = 25$, corresponding to month 9. Continuing the algorithm,

$$\begin{aligned}
C_9(1) &= 4 \\
C_9(2) &= 2 \\
C_9(3) &= 2 \\
C_9(4) &= 2.255
\end{aligned}$$

Terminate here as $C_9(4) > C_9(3)$, where $y_9 = r_9 + r_{10} + r_{11} = 35$, corresponding to month 12. We finally terminate at the end of the planning horizon, with a final order at month 12:

$$C_{12}(1) = \frac{40}{20} = 2$$

where $y_{12} = r_{12} = 20$. Thus, the optimal order policy is given as $\mathbf{y} = \begin{pmatrix} 26 & 40 & 25 & 35 & 20 \end{pmatrix}$ with order points at months 1, 5, 7, 9, and 12. Similarly, we also consider the holding cost of the 8 units of ending inventory. The total cost is calculated with

$$\begin{aligned}
C_{\text{total}} &= C_1(4) \times 26 + C_5(2) \times 40 + C_7(2) \times 25 + C_9(3) \times 35 + C_{12}(1) \times 20 + 8 \times 1 \\
&= \boxed{\$312}
\end{aligned}$$

## (c) Part Period Balancing

Now, we consider the part period balancing heuristic. Here, we set the order horizon equal to the number of periods that most closely matches the total holding cost with the setup cost $K = \$40$ over that period. We start the algorithm with month 1, where the total holding cost is $0 + hr_2 + 2hr_3 + \cdots + (j-1)hr_j$:

| **Month** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Demand** | 6 | 12 | 4 | 8 | 15 | 25 | 20 | 5 | 10 | 20 | 5 | 12 |
| **Adjusted Demand** | 2 | 12 | 4 | 8 | 15 | 25 | 20 | 5 | 10 | 20 | 5 | 20 |

Table 2: Anticipated Demand over 12 Months

| **Order Horizon** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Total Holding Cost** | 0 | 12 | 20 | 44 |

Table 3: Total Holding Costs over Order Horizon

Since $44 > K = 40$, we terminate the algorithm. As 44 is closer to 40 than 20, the first order horizon is 4 periods, $y_1 = r_1 + r_2 + r_3 + r_4 = \mathbf{26}$. We now start the next iteration with the $5^{\text{th}}$ month:

| **Order Horizon** | 1 | 2 | 3 |
|---|---|---|---|
| **Total Holding Cost** | 0 | 25 | 65 |

Table 4: Total Holding Costs over Order Horizon

Since $65 > K = 40$, we terminate the algorithm. As 25 is closer to 40 than 65, the second order horizon is 2 periods, $y_2 = r_5 + r_6 = \mathbf{40}$. We now start the next iteration with the $7^{\text{th}}$ month:

| **Order Horizon** | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **Total Holding Cost** | 0 | 5 | 25 | 85 |

Table 5: Total Holding Costs over Order Horizon

Since $85 > K = 40$, we terminate the algorithm. As 25 is closer to 40 than 85, the third order horizon is 3 periods, $y_3 = r_7 + r_8 + r_9 = \mathbf{35}$. We now start the next iteration with the $10^{\text{th}}$ month:

| **Order Horizon** | 1 | 2 | 3 |
|---|---|---|---|
| **Total Holding Cost** | 0 | 5 | 45 |

Table 6: Total Holding Costs over Order Horizon

Since $45 > K = 40$, we terminate the algorithm. As 45 is closer to 40 than 5, the fourth order horizon is 3 periods, $y_{10} = r_{10} + r_{11} + r_{12} = \mathbf{45}$. Since we have covered the entire order horizon of 12 months, we terminate the algorithm and conclude that the optimal order policy is $\mathbf{y} = \begin{pmatrix} 26 & 40 & 35 & 45 \end{pmatrix}$ with the order months 1, 5, 7, and 9. The cost associated with this heuristic is given by the sum of the holding costs from the tables above, and the setup cost for 4 orders. We also account for the holding cost of the **8 units of ending inventory**, so we have

$$\begin{aligned} C_{\text{total}} &= 44 + 25 + 25 + 45 + 4(40) + 8(1) \\ &= \boxed{\$307} \end{aligned}$$

## (d) Lowest Cost

From the total costs of each heuristic, we see that

$$C_{\text{S-M}} = \$\mathbf{303} \qquad C_{\text{LUC}} = \$312 \qquad C_{\text{PPB}} = \$307$$

where the Silver-Meal Heuristic corresponds to the lowest cost for the 12 periods.

# Question 2

## (a)

If the retailer is not happy with the outcomes of the approaches above, we will need to use a more exact method, which gives us the true optimal lot sizes. Optimal in this context means that the policy that minimises the total holding and setup cost over the planning horizon. This can be determined by casting the problem as a shortest-path problem, using Dynamic Programming (DP). An exhaustive search over all the feasible policies is impossible here, as we have $2^{12-1}$ policies for 12 planning periods. To solve this problem, we can use the Wagner-Whitin algorithm from the textbook, and solve it using a backwards DP approach.

DP is based on the principle of optimality, where if a problem consists of exactly $n$ stages and there are $r < n$ stages remaining, the optimal policy for the remaining stages is independent of the policy adopted in the previous stages.

Here, we can formulate the problem as a one-way network with the number of nodes equal to exactly one more than the number of periods, i.e. 13 nodes. Every path through the network corresponds to a specific exact requirements policy. For any pair $(i, j)$ with $i < j$, if the arc $(i, j)$ is on the path, then ordering takes place in period $i$ and the order size is equal to the sum of the requirements in periods $i, i + 1, \ldots, j - 1$. Period $j$ is the next period of ordering. All paths will end at period $n + 1 = 13$.

We now assign a value to each arc in the network, $c_{(i,j)}$ for path $(i, j)$. It is defined as the setup and holding cost of ordering in period $i$ to meet requirements through period $j - 1$. Finally, we would like to determine the minimum-cost production schedule, i.e. the shortest path through the network via DP. Define $f_k$ as the minimum cost starting at node $k$, assuming that an order is placed in period $k$. The principle of optimality for this problem results in the following system of equations

$$f_k = \min_{j>k}(c_{kj} + f_j) \qquad \text{for } k = 1, \ldots, n \tag{4}$$

with initial condition $f_{n+1} = f_{13} = 0$. In the backwards approach, we will define $C(t)$ as the minimum cost from period $t$ to the end.

```python
import numpy as np

# Parameters
D = [2, 12, 4, 8, 15, 25, 20, 5, 10, 20, 5, 20]
h = 1
s = 40
N = len(D)

# Initialize the cost array and order policy array
C = np.zeros(N + 1)
order_policy = np.zeros(N, dtype=int)

# Initialize the cost matrix for visualization
cost_matrix = np.full((N, N), np.inf)

# Backward Dynamic Programming
for t in range(N-1, -1, -1):
    min_cost = float('inf')
    best_k = t
    cumulative_demand = 0
    holding_cost = 0

    for k in range(t, N):
        cumulative_demand += D[k]
        holding_cost += (k - t) * D[k]

        cost = s + holding_cost + C[k + 1]
        cost_matrix[t, k] = cost

        if cost < min_cost:
            min_cost = cost
            best_k = k

    C[t] = min_cost
    order_policy[t] = best_k + 1

# Reconstruct the optimal order policy and quantities
optimal_orders = []
quantities = []
t = 0
while t < N:
    optimal_orders.append(t + 1)
    order_quantity = sum(D[t:order_policy[t]])
    quantities.append(order_quantity)
    t = order_policy[t]

optimal_cost = C[0]
optimal_orders, quantities, optimal_cost

# np.set_printoptions(precision=2, suppress=True)
# print("Cost Matrix:")
# print(cost_matrix)

✓ 0.0s
([1, 4, 6, 9, 12], [18, 23, 50, 35, 20], 295.0)
```

Figure 1: Python Approach to solving Dynamic Programming for optimal solution

From here, we see that we obtain the best possible order quantities to be

$$\mathbf{y} = (18, 0, 0, 23, 0, 50, 0, 0, 35, 0, 0, 20)$$

corresponding to the order period of months $(1, 4, 6, 9, 12)$. The optimal cost calculated here is is the **minimum cost** $C(t) = 295$, which when added with the ending inventory cost of 8 units, we get \$303, which is the same as the Silver-Meal heuristic, and is the optimum lowest possible cost.

## (b)

Since the shipping has a limitation in sending more than 40 TMEs per month, we now have a lot sizing problem with capacity constraints, a variant of the original problem. In each period, we also need to consider the production capacities $(c_1, \ldots, c_n)$, which will be 40 units for every month. Hence, we wish to find the optimal production quantities $(y_1, \ldots, y_n)$ subjected to the constraints $y_i \leq c_i$, for $1 \leq i \leq n$. Since we have a large solution space here for a total of 12 periods, instead of using the manual process similar to the examples in the textbook, we will augment the previous DP algorithm approach with an additional capacity constraint to solve and find the solution by a more efficient manner. The code is shown in Figure 2 below.

```python
import numpy as np

# Parameters
D = [2, 12, 4, 8, 15, 25, 20, 5, 10, 20, 5, 20]
h = 1
s = 40
capacity = 40  # Maximum number of TMEs that can be shipped per month
N = len(D)

# Initialize the cost array and order policy array
C = np.zeros(N + 1)
order_policy = np.zeros(N, dtype=int)

# Initialize the cost matrix for visualization
cost_matrix = np.full((N, N), np.inf)

# Backward Dynamic Programming with capacity constraint
for t in range(N-1, -1, -1):
    min_cost = float('inf')
    best_k = t
    cumulative_demand = 0
    holding_cost = 0

    for k in range(t, N):
        cumulative_demand += D[k]
        holding_cost += (k - t) * D[k]

        # Check if the order quantity exceeds the capacity
        if cumulative_demand > capacity:
            break

        cost = s + holding_cost + C[k + 1]
        cost_matrix[t, k] = cost

        if cost < min_cost:
            min_cost = cost
            best_k = k

    C[t] = min_cost
    order_policy[t] = best_k + 1

# Reconstruct the optimal order policy and quantities
optimal_orders = []
quantities = []
t = 0
while t < N:
    optimal_orders.append(t + 1)
    order_quantity = sum(D[t:order_policy[t]])
    quantities.append(order_quantity)
    t = order_policy[t]

optimal_cost = C[0]
optimal_orders, quantities, optimal_cost
```

✓ 0.0s

```
([1, 5, 7, 10, 12], [26, 40, 35, 25, 20], 299.0)
```

Figure 2: Incorporation of capacity constraint to DP

In this code, we introduce an additional variable `capacity` initiatied as 40 units, to act as the maximum number of TMEs that can be shipped. We see that the additional constraint

posed to the original DP algorithm is the conditional statement which checks whether the current `cumulative demand` exceeds the `capacity` of 40 units. If it does, the code breaks out of the loop and continues for other iterations, ensuring no order quantity surpasses the limit. With this, we obtain a new optimal order quantity to be

$$\mathbf{y} = (26, 0, 0, 0, 40, 0, 35, 0, 0, 25, 0, 20)$$

corresponding to the order months of $(1, 5, 7, 10, 12)$, and an optimal cost of \$299. Considering the final inventory of 8 units, we have a total final cost for the capacity constraint subproblem to be \$299+\$8 = \$307, which is the same as the Part Period Balancing Heuristic.

# Question 3

We are given the following information:

| **Job** | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 6 | 8 | 4 | 2 | 1 | 7 |
| **Due Date** | 4 | 8 | 12 | 15 | 11 | 25 | 21 |

Table 7: Sequencing for Jobs

## Mean Flow Time

The Mean Flow Time is given by

$$F' = \frac{1}{n}\sum_{i=1}^{n} F_i \tag{5}$$

In order to minimise the Mean Flow Time, we use the theorem:

**Theorem 0.1.** The scheduling rule that minimises the mean flow time $F$' is the Shortest Processing Time (SPT). The SPT rule prioritises jobs with shorter processing times, which generally leads to a lower mean flow time.

We now order the jobs in Table 7 by the SPT rule:

| **Job** | 6 | 5 | 1 | 4 | 2 | 7 | 3 | **Total** |
|---|---|---|---|---|---|---|---|---|
| **Processing Time** | 1 | 2 | 3 | 4 | 6 | 7 | 8 | |
| **Completion Time** | 1 | 3 | 6 | 10 | 16 | 23 | 31 | **90** |
| **Due Date** | 25 | 11 | 4 | 15 | 8 | 21 | 12 | |
| **Tardiness** | 0 | 0 | 2 | 0 | 8 | 2 | 19 | 31 |

Table 8: Sequencing for Jobs

Here we have a mean flow time of $90/7 = \mathbf{12.857}$, with the sequence of jobs being 6, 5, 1, 4, 2, 7, 3 respectively.

## Number of Tardy Jobs

To minimise the number of Tardy Jobs, we need to use Moore's algorithm. We first sequence the job via Earliest Due Date scheduling:

| **Job** | 1 | 2 | 5 | 3 | 4 | 7 | 6 | **Total** |
|---|---|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 6 | 2 | 8 | 4 | 7 | 1 | |
| **Completion Time** | 3 | 9 | 11 | 19 | 23 | 30 | 31 | **125** |
| **Due Date** | 4 | 8 | 11 | 12 | 15 | 21 | 25 | |
| **Tardiness** | 0 | 1 | 0 | 7 | 8 | 9 | 6 | |

Table 9: Earliest Due Date Scheduling

We see that the first tardy job here is job 2, and there are 5 tardy jobs in total. We now consider jobs 1 and 2, and reject the job with the longest processing time, which is job 2. Now, the new sequence is

| **Job** | 1 | 5 | 3 | 4 | 7 | 6 | **Total** |
|---|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 2 | 8 | 4 | 7 | 1 | |
| **Completion Time** | 3 | 5 | 13 | 17 | 24 | 25 | 87 |
| **Due Date** | 4 | 11 | 12 | 15 | 21 | 25 | |
| **Tardiness** | 0 | 0 | 1 | 2 | 3 | 0 | |

Table 10: Minimise Tardy Jobs, Iteration 1

At this point, the first tardy job is job 3, also with the longest processing time, so we reject it. The new sequence is now

| **Job** | 1 | 5 | 4 | 7 | 6 | **Total** |
|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 2 | 4 | 7 | 1 | |
| **Completion Time** | 3 | 5 | 9 | 16 | 17 | 50 |
| **Due Date** | 4 | 11 | 15 | 21 | 25 | |
| **Tardiness** | 0 | 0 | 0 | 0 | 0 | |

Table 11: Minimise Tardy Jobs, Iteration 2

Now, there are clearly no more tardy jobs, so we terminate the algorithm. We see that the optimal sequence of jobs to minimise the number of tardy jobs is (1, 5, 4, 7, 6, 2, 3) respectively.

## Maximum Lateness

To minimise maximum lateness, the jobs should be sequenced according to their due dates. That is, $d_1 \leq d_2 \leq \cdots \leq d_n$. As such, we reference Table 9 for Earliest Due Date (EDD) scheduling:

| **Job** | 1 | 2 | 5 | 3 | 4 | 7 | 6 | **Total** |
|---|---|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 6 | 2 | 8 | 4 | 7 | 1 | |
| **Completion Time** | 3 | 9 | 11 | 19 | 23 | 30 | 31 | **125** |
| **Due Date** | 4 | 8 | 11 | 12 | 15 | 21 | 25 | |
| **Tardiness** | 0 | 1 | 0 | 7 | 8 | 9 | 6 | |

Table 12: Sequencing for Maximum Lateness with EDD scheduling

From the table, we see that the optimal sequence of jobs to minimise Maximum Lateness is **(1, 2, 5, 3, 4, 7, 6)** respectively.

## Makespan

The makespan is the flow time of the job that is completed last, and thus it is also the time required to complete all 7 jobs. In this example, there is no possible way to minimise the makespan since we are only considering one single machine here, and thus the makespan is just the sum of the processing time of all jobs in Table 7.

| **Job** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **Total** |
|---|---|---|---|---|---|---|---|---|
| **Processing Time** | 3 | 6 | 8 | 4 | 2 | 1 | 7 | **31** |
| **Due Date** | 4 | 8 | 12 | 15 | 11 | 25 | 21 | |

Table 13: Minimising Makespan

From the table, we have the makespan to be

$$\text{Makespan} = 3 + 6 + 8 + 4 + 2 + 1 + 7 = 31$$

Thus, the makespan is 31 units of time, and will be the same irrespective of the job sequence for a one-machine scenario, and every job needs to be processed sequentially. We can just sequence it by index for example, namely (1, 2, 3, 4, 5, 6, 7).

# Question 4

## First-Come-First-Served

We are given the following information in Table form. For a First-Come-First-Served (FCFS) schedule, we have the sequence $A - B - C - D$ as follows, starting from 1:00pm:

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Completion Time** | **Flow Time** | **Tardiness** |
|---|---|---|---|---|---|
| A | 20 | 1:25pm | 1:20pm | 20 | 0 |
| B | 14 | 1:45pm | 1:34pm | 34 | 0 |
| C | 35 | 1:50pm | 2:09pm | 69 | **19** |
| D | 10 | 1:30pm | 2:19pm | 79 | **49** |
| **Total** | | | | 202 | 68 |

Table 14: First Come First Served Schedule

From the table, we can compute the mean flow time as

$$\begin{aligned} F' &= \frac{1}{4}\sum_{i=1}^{4} F_i \\ &= \frac{1}{4}\left(20 + 34 + 69 + 79\right) \\ &= 50.5 \end{aligned}$$

Thus the mean flow time is **50.5 minutes**. The average tardiness can be found via

$$\begin{aligned} \text{Average Tardiness} &= \frac{0 + 0 + 19 + 49}{4} \\ &= 17 \text{ minutes} \end{aligned}$$

and clearly the number of tardy jobs here is **2.**

## Shortest Processing Time

Here, we sequence the schedules based on increasing order of processing (unloading) times, specifically $D - B - A - C$. Similarly, if we start from 1:00pm:

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Completion Time** | **Flow Time** | **Tardiness** |
|---|---|---|---|---|---|
| D | 10 | 1:30pm | 1:10pm | 10 | 0 |
| B | 14 | 1:45pm | 1:24pm | 24 | 0 |
| A | 20 | 1:25pm | 1:44pm | 44 | **19** |
| C | 35 | 1:50pm | 2:19pm | 79 | **29** |
| **Total** | | | | 157 | 68 |

Table 15: Shortest Processing Time Schedule

From the table, we can compute the mean flow time as

$$\begin{aligned} F' &= \frac{1}{4}\sum_{i=1}^{4} F_i \\ &= \frac{1}{4}\left(10 + 24 + 45 + 79\right) \\ &= 39.25 \end{aligned}$$

Thus the mean flow time is **39.25 minutes**. The average tardiness can be found via

$$\begin{aligned} \text{Average Tardiness} &= \frac{0 + 0 + 19 + 29}{4} \\ &= 12 \text{ minutes} \end{aligned}$$

and clearly the number of tardy jobs here is also **2.**

## Earliest Due Date

Here, we sequence the schedules based on the increasing order of the due times, specifically $A - D - B - C$. Similarly, if we start from 1:00pm:

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Completion Time** | **Flow Time** | **Tardiness** |
|---|---|---|---|---|---|
| A | 20 | 1:25pm | 1:20pm | 20 | 0 |
| D | 10 | 1:30pm | 1:30pm | 30 | 0 |
| B | 14 | 1:45pm | 1:44pm | 44 | 0 |
| C | 35 | 1:50pm | 2:19pm | 79 | **29** |
| **Total** | | | | 173 | 29 |

Table 16: Earliest Due Date Schedule

From the table, we can compute the mean flow time as

$$\begin{aligned} F' &= \frac{1}{4}\sum_{i=1}^{4} F_i \\ &= \frac{1}{4}\left(20 + 30 + 44 + 79\right) \\ &= 43.25 \end{aligned}$$

Thus the mean flow time is **43.25 minutes**. The average tardiness can be found via

$$\begin{aligned} \text{Average Tardiness} &= \frac{0 + 0 + 0 + 29}{4} \\ &= 7.25 \text{ minutes} \end{aligned}$$

and clearly the number of tardy jobs here is now only **1.**

## Critical Ratio

After a job has been processed, we sequence the schedules based on a critical ratio, namely

$$\text{Critical Ratio (CR)} = \frac{\text{Processing Time}}{\text{Due Time} - \text{Current Time}} \tag{6}$$

We schedule the next job in order to minimise the value of the critical ratio, and the idea behind this is to provide a balance between the SPT (only considers processing time), and EDD (only considers due dates). The ratio will grow larger as the current time approaches the due date, and more priority will be given to those jobs with longer processing times. Treating start time 1:00pm as $t = 0$, we have

$$\begin{aligned} \text{CR}_A &= \frac{20}{25 - 0} \\ &= 0.80 \end{aligned} \qquad \begin{aligned} \text{CR}_B &= \frac{14}{45 - 0} \\ &= 0.311 \end{aligned} \qquad \begin{aligned} \text{CR}_C &= \frac{35}{50 - 0} \\ &= 0.7 \end{aligned} \qquad \begin{aligned} \text{CR}_D &= \frac{10}{30 - 0} \\ &= 0.333 \end{aligned}$$

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Critical Ratio** |
|---|---|---|---|
| A | 20 | 25 | **0.8** |
| B | 14 | 45 | 0.311 |
| C | 35 | 50 | 0.7 |
| D | 10 | 30 | 0.333 |

Table 17: Critical Ratio Schedule, $t = 0$

From here, we see that the vehicle with the largest ratio is A, so it is performed first. As it requires 20 minutes to process, we must update all the critical ratios in order to determine

the next job to process, by setting $t = 20$ now and recomputing the critical ratios:

$$\text{CR}_B = \frac{14}{45-20} \quad = 0.56 \qquad \text{CR}_C = \frac{35}{50-20} \quad = 1.167 \qquad \text{CR}_D = \frac{10}{30-20} \quad = 1.0$$

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Critical Ratio** |
|---|---|---|---|
| B | 14 | 45 | 0.56 |
| C | 35 | 50 | **1.167** |
| D | 10 | 30 | 1.0 |

Table 18: Critical Ratio Schedule, $t = 20$

From here, we see that C has the highest ratio, so it is next in line of the sequence. Since C has a processing time of 35 minutes, we update the time $t = 55$ and recompute the critical ratios:

$$\text{CR}_B = \frac{35}{50-55} \quad = -7 \qquad \text{CR}_D = \frac{10}{30-55} \quad = -0.4$$

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Critical Ratio** |
|---|---|---|---|
| B | 14 | 45 | -7 |
| D | 10 | 30 | -0.4 |

Table 19: Critical Ratio Schedule, $t = 55$

From here, we see that both critical values are negative, indicating that both B and C are now late. Late values here are scheduled in SPT order, so they are done in the sequence D and then B. Thus, the sequence here is $A - C - D - B$.

| **Delivery Vehicle** | **Unloading Time** | **Due Time** | **Completion Time** | **Flow Time** | **Tardiness** |
|---|---|---|---|---|---|
| A | 20 | 1:25pm | 1:20pm | 20 | 0 |
| C | 35 | 1:50pm | 1:55pm | 55 | **5** |
| D | 10 | 1:30pm | 2:05pm | 65 | **35** |
| B | 14 | 1:45pm | 2:19pm | 79 | **34** |
| **Total** | | | | 219 | 29 |

Table 20: Earliest Due Date Schedule

From the table, we can compute the mean flow time as

$$\begin{aligned} F' &= \frac{1}{4}\sum_{i=1}^{4} F_i \\ &= \frac{1}{4}(20 + 55 + 65 + 79) \\ &= 54.75 \end{aligned}$$

Thus the mean flow time is **54.75 minutes**. The average tardiness can be found via

$$\begin{aligned} \text{Average Tardiness} &= \frac{0 + 5 + 35 + 34}{4} \\ &= 18.5 \text{ minutes} \end{aligned}$$

and clearly the number of tardy jobs here is now **3.** A summary of the values are given in the table below:

| **Rule** | **Mean Flow Time** | **Average Tardiness** | **Number of Tardy Jobs** |
|---|---|---|---|
| FCFS | 50.5 | 17 | 2 |
| SPT | 39.25 | 12 | 2 |
| EDD | 43.25 | 7.25 | 1 |
| CR | 54.74 | 18.5 | 3 |

Table 21: Summary of Results for Four Scheduling Rates

# Question 5

The context of the problem appears to be a scheduling problem for 5 jobs on two machines, with each job being one subject and the two machines being Laurel and Hardy. We rely on the crucial assumption here that each job must be processed in the order machine 1 (Laurel) then machine 2 (Hardy). Furthermore, we also assume that the optimization criterion here is to minimise the makespan. We consider the theorem:

**Theorem 0.2.** The optimal solution for scheduling $n$ jobs on two machines is always a permutation schedule.

Although the question **unintentionally left out details** on the order of the scheduled jobs, we will proceed with the previously stated assumptions regardless. We note that the total number of permutation schedules here is exactly $n! = 5! = 120$, and we can use the Johnson's Algorithm to efficiently solve the optimal schedule. Following Johnson's notation in the textbook, we first define the two machines as A and B (Laurel and Hardy) respectively. It is assumed that the jobs must be processed first on machine A then on B. Suppose that the jobs are labelled $i$, for $1 \leq i \leq 5$, and define

$$\begin{aligned} A_i = &\text{ Processing time of job } i \text{ on machine A.} \\ B_i = &\text{ Processing time of job } i \text{ on machine B.} \end{aligned}$$

Johnson's result is that the following rule is optimal for determining an order in which to process the jobs on the two machines:

$$\text{Rule: Job } i \text{ precedes job } i+1 \text{ if } \min(A_i, B_{i+1}) < \min(A_{i+1}, B_i)$$

To implement this rule, we follow the steps:

1. List the values of $A_i$ and $B_i$ in two columns
2. Find the smallest remaining element in the two columns. If it appears in column A, then schedule that job next. If it appears in column B, then schedule that job last.
3. Cross of the jobs as they are scheduled. Stop when all jobs have been scheduled.

We have the following information in the table:

| **Job** | **Laurel (A)** | **Hardy (B)** |
|---|---|---|
| 1 | 40 | 20 |
| 2 | 15 | 30 |
| 3 | 25 | 10 |
| 4 | 15 | 35 |
| 5 | 20 | 25 |

Table 22: Processing Times for each Job

Following the algorithm, we get the following schedule:

| Iteration 1 | | | Iteration 2 | | | Iteration 3 | | | Iteration 4 | | | Iteration 5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job | Laurel (A) | Hardy (B) | Job | Laurel (A) | Hardy (B) | Job | Laurel (A) | Hardy (B) | Job | Laurel (A) | Hardy (B) | Job | Laurel (A) | Hardy (B) |
| 1 | 40 | 20 | 1 | 40 | 20 | 1 | 40 | 20 | 1 | 40 | 20 | 1 | 40 | 20 |
| 2 | 15 | 30 | 2 | 15 | 30 | 2 | 15 | 30 | 2 | 15 | 30 | 2 | 15 | 30 |
| 3 | 25 | 10 | 3 | 25 | 10 | 3 | 25 | 10 | 3 | 25 | 10 | 3 | 25 | 10 |
| 4 | 15 | 35 | 4 | 15 | 35 | 4 | 15 | 35 | 4 | 15 | 35 | 4 | 15 | 35 |
| 5 | 20 | 25 | 5 | 20 | 25 | 5 | 20 | 25 | 5 | 20 | 25 | 5 | 20 | 25 |

Figure 3: Johnson Algorithm Iterations

This sequence gives us the following optimal sequencing schedule: [HASS, ESA, AE, SMT, MSO]. To calculate the start and end times of each of the subjects, we schedule them accordingly for machine A and B:

| Start | 0 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 125 | 130 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| End | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 125 | 130 | 135 |
| Laurel (A) | HASS | | | ESA | | | AE | | | | SMT | | | | | | | | MSO | | | | | | | | |
| Hardy (B) | | | | HASS | | | | | | | ESA | | | | | | | AE | | | | | SMT | | | | MSO |

Figure 4: Time Schedule for each subject

We note that there is no idle time here between each subject, even though machine B starts 15 minutes later than machine A. This is a feature of all optimal schedules.