For the 3rd class students / Operations Research II
Mathematics Department / College of Science for Women

Machine Scheduling Problem

Suppose that m machines M_i $(i = 1,\dots,m)$ have to process n jobs j $(j = 1,\dots,n)$. A **schedule** is for each job an allocation of one or more-time intervals to one or more machines. A schedule is **feasible** if at any time, there is at most one job on each machine; each job is run on at most one machine. A schedule is optimal if it minimizes (or maximizes) a given optimality criterion. A scheduling problem type can be specified using three- field classification α / β / γ composed of machine environment, the job characteristics, and the optimality criterion.

Job Data : Let $\boldsymbol{n}$ denote the number of jobs. The following data is specified for each job $\boldsymbol{j}$ $(\boldsymbol{j} = \mathbf{1}, \mathbf{2}, \dots, \boldsymbol{n})$:

p_{ij}	A processing time of its i^{th} operation, $i = 1, 2, \dots, m_j$, where m_j is the number of operations on job j. If $m_j = 1$, we shall write p_i instead of p_{ij}
r_j	A release date on which job j become available for processing.
d_j	A due date, the time by which job j ideally be completed.
$\bar{d}_j$	A deadline, the time by which j must be completed.
w_j	The weight of job j representing the importance of job j relative to another job.
f_j	A non-decreasing real cost function measuring the cost $f_j(t)$ incurred if job j completed at time t.

In general, p_{ij} , d_j, r_j, d_j and w_j are given positive integer constants.

The first field $\boldsymbol{\alpha}$: There are various possible cases of machines, all of which become available to process at time zero. There are four classes of machine environment.

a. Single Machine Scheduling: It can be defined as sequencing number of n jobs operating on a single machine.

b. Parallel Machines Scheduling: Parallel machines scheduling involves scheduling a set of jobs on two or more machines that work in parallel with each other. In this type of problem, the jobs are assigned to either machine for processing, and the flow between machines is not allowed.

c. Flow Shop Scheduling: A flow shop scheduling consists of two or more machines and a set of jobs that must be processed on each of these machines. This arrangement is called a flow shop because the products flow along a specific unidirectional path. Each product must be processed on each machine in the same order e.g. 1st -machine 1, 2nd – machine 2,…, mth – machine m. The processing times for each job can vary from machine to machine and the processing times on each machine can vary from job to job.

d. Job Shop Scheduling: A job shop consists of two or more than two machines that perform specific operations, and a set of jobs that must be processed on some or all of these machines. Unlike the flow shop, there is no fixed path that the products must follow through the system therefore the order of operations is not fixed. This type of layout is typically used when the product variety is high and the product volume is low.

The second field β is a list of the variables and the constraints indicates job characteristics:

- r_j a parameter defines that jobs have specified release dates, on which job j becomes available for processing.
- d_j a parameter indicates that jobs have specified due dates. This is a significant parameter of the job, because after which job j is consider late.
- $\boldsymbol{s_f}$ is used when jobs have specified sequence dependent setup times.
- If ptmn is present, then preemptions are allowed; the processing of any job may be interrupted at no cost and resumed at later time.
- If a prec is present, then there is a precedence relation $\prec$ among the jobs, i.e., if $J_j \prec J_k$, then J_j must be completed before J_k can be started.
- $d_j = d$ indicates that all the due dates are identical.
- $P_j = p$ indicates that all the processing times are identical.

Optimality Criteria

The third field $\boldsymbol{\gamma}$ defines the optimality criterion or the objective, the value which is to be optimized (minimized). Given a schedule, the following can be computed for each job j:

C_j	The completion time, the time at which the processing of job j is completed.
F_j	The flow time, the time job j spends in the system, $F_j = C_j - r_j$.
L_j	The lateness, $L_j = C_j - d_j$, the amount of time by which the completion time of job j exceed its due date. Lateness can be negative if job j finishes earlier than its due date.
T_j	The tardiness, $T_j = max\{L_j, 0\}$.
E_j	The earliness, $E_j = max\{-L_j, 0\}$.
F_j	The flow time $F_j = C_j - r_j$
U_j	The unit penalty, a unit penalty of job j if it fails to meet its deadline. $U_j = 0$ if $C_j \leq d_j$, $U_j = 1$ otherwise.

The cost f_j for each job j usually takes one of the variables described above or the product of the weight w_j with one of the variables. The optimality criterion can be any function of the costs $f_j, j = 1,2,3,..,n$. Common optimality criteria are usually in the form

1) $f = f_{max} = max\{f_j | j = 1,2,..,n\}$.
2) $f = \sum f_j$.

The following objective functions have frequently been chosen to be minimized.
$f = \sum(w_j)C_j$: The total (weighted) completion time.

Introducing due dates d_j $(j = 1, \dots, n)$ we have the following objective functions:

$f = C_{max}$	: The maximum completion time (makespan)
$f = L_{max} = max\{L_j\}$	: The maximum lateness
$f = T_{max} = max\{T_j\}$	: The maximum tardiness.
$f = \sum T_j$	: The total tardiness.
$f = \sum U_j$	: The total number of late jobs.

We may also choose to minimize:

$f = \sum w_j T_j$	: The total weighted tardiness.
$f = \sum w_j U_j$	: The total weighted number of late jobs.
$f = \sum w_j E_j$	: The total weighted earliness.

Example (1):

$1/r_j \ / \sum w_j C_j$	is the problem of minimizing the total weighted completion time on single machine subject to non-trivial release date.
$P_3/pmtn, prec \ / \ L_{max}$	is the problem of minimizing maximum lateness on three identical parallel machines subject to general precedence constraint, allowing preemption.

Fundamental Theorems and Algorithms:

Theorem (Jackson): The $1//L_{\max}$ problem is minimized by sequencing the jobs according to the earliest due date (EDD) rule, that is, in order of non-decreasing d_j (i.e. $d_1 \le d_2 \le \dots \le d_n$), this rule also minimize T_{max} for the $1//\ T_{max}$ problem

Theorem (Smith): The $1// \sum_{j=1}^{n} C_j$ problem is minimized by sequencing the jobs according to the shortest processing time (SPT) rule, that is, in order of non-decreasing p_j(i.e. $p_1 \le p_2 \le \dots \le p_n$),

Theorem (Hoogeveen): The $1//E_{\ max}$ problem is solved by sequencing the jobs according to the minimum slack time (MST) rule, that is, in order non-decreasing $s_j = d_j - p_j$, (i.e. $s_1 \le s_2 \le \dots \le s_n$).

Theorem (Moore): The $1//\Sigma U_j$ problem, is minimized as follows: repeatedly adds jobs in EDD order to the end of partial schedule of on time jobs. If the addition of job j results in this job being complete after time d_j , then a job in the partial schedule with the largest processing time is removed and declared late.

Theorem (Lawler): The 1// f_{max} problem, fmax is minimized as follows: while there are unassigned jobs, assign the job that has minimum cost when scheduled in the last unassigned position in that position.

Lawler algorithm (LA) which solves the 1/prec/ f_{max} problem or 1// f_{max} problem where $f_{max} \in \{L_{max}, T_{max}, V_{max}, V^w_{max}, T^w_{max}\}$.

Lawler algorithm

Step (1): Let $N = \{1, \dots, n\}, \sigma = (\varphi)$ and F set of all jobs with no successors.

Step (2): Let j^* such that $f_{j^*}(\sum_{j\in N} p_j) = min_{j\in F}\{f_j(\sum_{j\in N} p_j)\}$.

Set $N = N - \{j^*\}$ and sequence j^* in $\sigma, i.e., \sigma = (j^*, \sigma)$. Modify F to represent the new set of schedulable jobs.

Step (3): If $N = \varphi$ stop, otherwise go to step (2).

Job priority is determined as a function of job parameters, the machine parameters or the characteristics of the store (store characteristics). When prioritizing each job, the jobs are sorted, and then the job with the highest priority is selected for first processing. Below are some scheduling rules that have been developed, studied, and implemented by many researchers and practitioners:

- Longest Processing Time (LPT): the job with the largest operation processing time is processed first. this lends to increase work in progress and make short jobs late.

- First Come, First Served (FCFS), or Smallest Ready Time (FCFS or SORT): The job which arrives first at the machine will be served first. often seen as a fair rule, especially by the customer, but in practice, it leads to overall inefficiency.

- Last Come, First Served (LCFS): The job which arrives last will be served first

Dominance Rules for MSP: Dominance rules (DR) are used efficiently in the reduction of existing sequences. DR is usually use to indicate whether a particular node in the BAB method can be deleted before computation (LB). These rules are useful when a node has a suboptimal solution and can be removed. If the nodes in the BAB method are dominated by others, DRs can also be used to solve these nodes. These developments can significantly reduce the number of nodes in the search for the optimal solution. DRs can also be applied to such problems. Let ND denote the number of jobs dominated by it and the number of jobs not dominated by NND

Definition: The graph G represents a finite number of nodes or vertices V and a finite number of edges connecting two vertices, and the edge connecting the vertex to itself is called a loop.

Definition: A directed graph, also known as a graph, is a graph with a finite number of directed edges, each connecting an ordered pair of vertices Strang, (2009).

Definition: If n vertices make up a graph called G, then $A(G) = [a_{ij}]$ be the matrix (which is called adjacency matrix), whose i^{th} and j^{th} element is *1* if there is at least one edge between two vertices v_1 and v_2 and zero otherwise Strang, (2009),

$$a_{ij} = \begin{cases} 0, \text{if } i = j \text{ } or \text{ } i \nrightarrow j \\ 1, \text{if } i \rightarrow j \\ a_{ij} \text{ } and \text{ } \bar{a}_{ij}, \quad i \leftrightarrow j \end{cases}$$

$A(G)$ can be formulated as follows:

$$A(G) = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & 0 & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & 0 & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & 0 \end{bmatrix}.$$

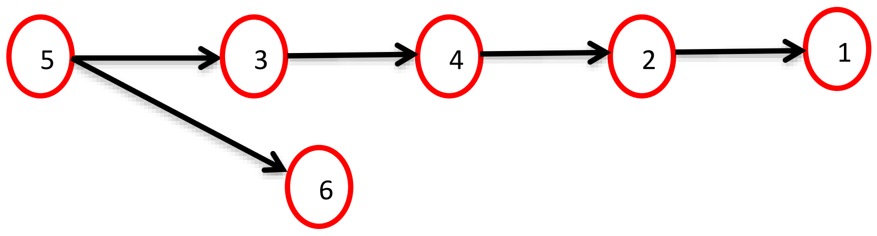
Emmon's Theorem (1): For the $1//\sum T_j$ problem, if $p_i \leq p_j$ and $d_i \leq d_j$ then there exists an optimal sequencing in which job i sequencing before job j

Remark (1): For problem $1//\ E_{max}$ there exists an optimum solution s.t. job i is sequenced before job j if $p_i \leq p_j$ and $s_i \leq s_j$

Example (2): Let's use MSP with 6 jobs and the following processing time, due date:

	job_1	job_2	job_3	job_4	job_5	job_6
p_j	7	7	4	6	1	9
d_j	19	16	15	16	10	14
$s_j = d_j - p_j$	12	9	9	10	9	5

The DRs by using theorem (1) is illustrated in Figure (1).



Notice that there are (11) DRs: 2→1, 3→ 1,3→2, 3→4, 4→1,4→2,5→ 1, 5→2, 5→ 3, 5→ 4,5→6, with the number of jobs not dominated (4): 3 ↔ 6,

4 ↔ 6,2 ↔ 6,1 ↔ 6. The adjacency matrix A is as followings:

$$A(G)=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & a_{16} \\ 1 & 0 & 0 & 0 & 0 & a_{26} \\ 1 & 1 & 0 & 1 & 0 & a_{36} \\ 1 & 1 & 0 & 0 & 0 & a_{46} \\ 1 & 1 & 1 & 1 & 0 & 1 \\ a_{61} & a_{62} & a_{63} & a_{64} & 0 & 0 \end{bmatrix}.$$

These DRs may be useful for finding a good solution to some problems such as $1//\sum T_j$.

Example (3): Let's use MSP with 6 jobs and the following processing time, due date:

	job_1	job_2	job_3	job_4	job_5	job_6
p_j	16	12	12	4	8	10
d_j	29	24	22	6	12	18
$s_j = d_j - p_j$	13	12	10	2	4	8

The DRs by using remark (1) is illustrated in Figure (2).

$$\begin{tikzcd} 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1 \end{tikzcd}$$

Figure (2): The DRs for the example (3)

Notice that there are (14) DRs: 4→5, 4→ 6,4→3, 4→2, 4→1,5→6,5→ 3, 5→2, 5→1, 6→3, 6→1, 3→2, 3→1, 2→1, with adjacency matrix A is as followings:

$$A(G)=\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

These DRs may be useful for finding a good solution to some problems such as $1//E_{max}$.

Methods for Solving MSP: The optimal solution of the MSP is to find a processing sequence for the jobs on each machine so that a given production level reaches its optimal value. The goal of the MSP is to find an optimal schedule from a finite number of feasible schedules. To find the one with the smallest value of the objective function, all possibilities can be searched. When all schedules are compared, this search for a finite set of schedules should eventually end and the smallest value (optimal solution) should be found. In this case, if there are n jobs and a machine scheduling problem, there are $(n!)$ different sequence. Thus, for the corresponding m-machine problem, there are possible processing orders $(n!)^m$ and which is very large even for small values of (n) and (m). The most popular methods for solving MSPs are generally categorized into types. The first method leads to optimal solutions and is called “exact methods”, the second method leads to near-optimal solutions and is called "approximate methods" or "heuristic methods". The following figure (3) shows the solution methods used to solve MSP.

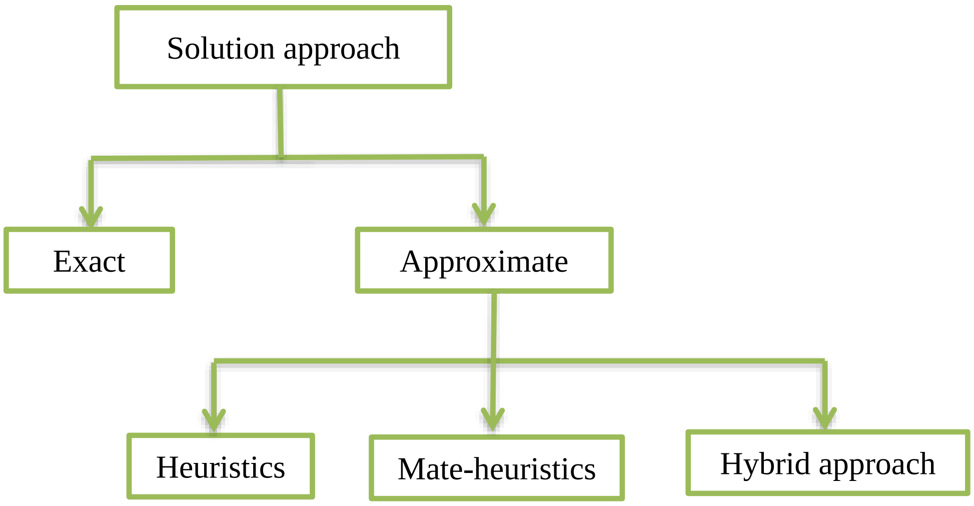


Figure (3): The solution methods to solve MSP

Exact Methods: There are several exact methods to find the optimal solution for MSP. This study gives a brief introduction to two methods (Complete Enumeration, Branch and Bound and Dynamic Programming).

Complete Enumeration Method: The complete enumeration method (CEM) generates the permutation or ordering of the objects one by one to find the optimal solution and list all possible permutations, then delete the non-optimal permutations from the list, leaving the ones that are optima. Searching for the optimal permutation among all possible permutations using the complete enumeration is not suitable even for problems of small size. Thus, CEM can only be used for problems with small sizes $n < 10$ or 11. For example, there are $(n!)$ different sequences obtained for SMSP for n jobs, and there are $(n!)^m$ different sequences obtained for n jobs and case m machines scheduling problem.

Branch and Bound Method: Branch and Bound method (BAB) is provided an exact solution to the NP-hard MSP problem. BAB applies implicit enumeration techniques to find an optimal solution by automatically testing some feasible solutions. BAB is based on finding trees of nodes related to these solutions. Each node has contained a subsequence of jobs. This technique contains $(n-1)$ levels. For level zero there are no open nodes yet, while for level one there are nodes to be checked. The first 1^{th} position in the specified sequence is used by one job in numerical order. Thus, each node in the level $(n-1)^{th}$ must branch to $(n-2)$. This processing contains only one fragment. To reduce the calculations, lower bounds (LBs) are calculated at each level for each node. The LB is calculated by a formula derived from the objective function of the MSP. Which LB is the minimum from which we branch from this node, and this describes why the computation is low. In the general description of the BAB methods, all possible sets are divided into different subgroups. For each subset, the LB is calculated, this means the cost of sequential jobs (depending on the objective function) and the cost of non-sequential jobs (depending on the derived LB). If the LB of this subset is greater than or equal to the upper bound (UB), this subset is ignored. the Upper Bound UB is usually defined as the minimum value of all currently existing feasible solutions, since any subset with a value less than the UB can only exist in the remaining subsets. These remaining subsets must be considered one after the other. According to a search strategy, one of these subsets is selected from which the branch is accessed. This subset is then subdivided into individual smaller subsets. As soon as, one of these subsets contains only one element, a complete sequence of functions should be available. This sequence is evaluated and if its value is smaller than the current maximum, this upper bound is changed accordingly. Then, the procedure is repeated until all subsets (nodes) have been taken into account. The upper bound at the end of this BAB procedure is optimal solution for a particular problem. Depending on the problem, the search tree node can also be eliminated using the sovereignty properties or feasibility conditions developed for the particular problem. The steps of BAB method introduced in the following algorithm (1):

Algorithm (1): Branch and Bound (BAB)

Step (1)	Compute upper bound (UB), partitions subsets, by using a branching procedure.
Step (2)	Compute lower bounds (LB) using LB procedure.
Step (3)	Exclude the subsets which are note include any optimal by using steps1 and steps2.
Step (4)	Determine active node, with smallest LB.
Step (5)	If $\text{LB} > \text{UB}$ for a particular subset, then this subset is ignored (UB is the minimum. of the values of all f solutions currently found). The remaining subsets with $(\text{LB} \leq \text{UB})$ have to be considered one at a time. One of these subsets is chosen according to some search strategy from which to branch.

Step (6)	When the branching end at a complete sequence of jobs, this sequence is evaluated and if its value is less than the current UB, then this UB is reset to take that value.
Step (7)	Repeat the procedure until all nodes have been considered (i.e., LB of all nodes in the search tree greater than or equal UB), a feasible solution with this UB is an optimal solution.

Figure (4) shows the BAB flow chart for three jobs

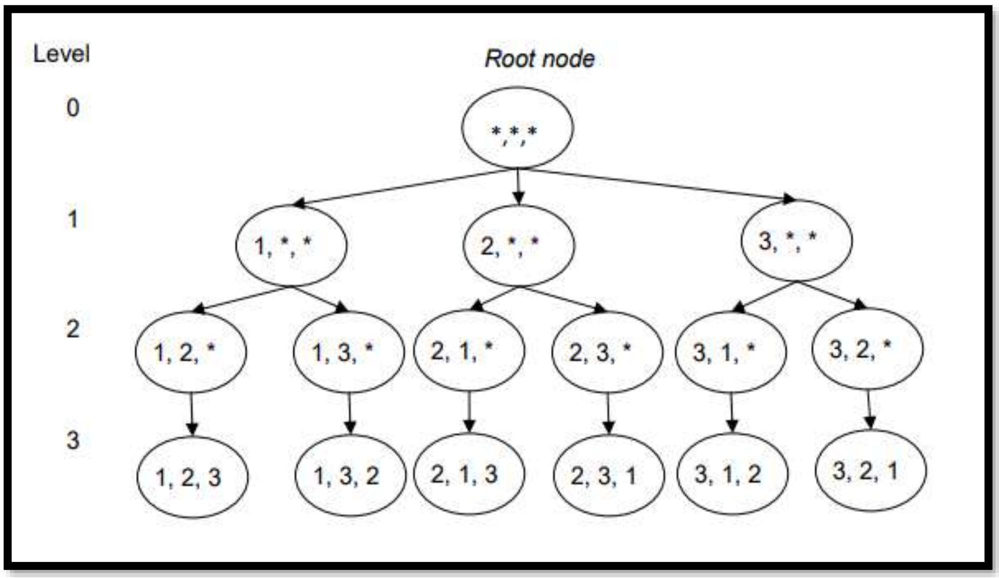


Figure (4): Flowchart of the BAB

Dynamic Programming: In the dynamic optimization approach, complex problems are broken down into a series of simpler sub-problems, each of which is solved only once before the data structure is created. The next time the same subproblem is encountered, the solution is not recalculated, but a previously calculated answer is performed. This reduces computation time at the expense of at least a little more storage space. To facilitate the search process, all solutions of subproblems are indexed in some way, usually based on the values of the input parameters). However, there are some difficulties in doing so: finding a good way to divide the problem into a number of subproblems, and the need for a large storage unit for information in the computer due to a number of quantities that are calculated from the iterative equations during the phases and that need to be stored during the calculations. Therefore, it is difficult to solve problems where there are many $n \leq 30$ jobs.

Heuristic Methods: Exact methods take a long time to find the optimal solution and have failed for large. On the other hand, Heuristic methods (HMs) are used to find the optimal solutions that are impractical for more than a few objects. HMs can replace these enumeration methods and attempt to find an optimal (or near-optimal) solution to difficult problems. In addition, it can be useful when the search for the optimal solution takes more time than is practical or cannot be found. Generally, HMs can be categorized into scheduling rules constructive and improvement heuristics. The constructive heuristics create a schedule from a list of planned jobs. Improvement heuristics, start with a problem and work their way up. In constructive heuristics, once a schedule has been created, it cannot be changed, whereas in improvement heuristics, an original solution is iteratively improved.

Example (4):

Consider the following schedule:

j	1	2	3	4	5	6	7	8
P_j	7	3	2	9	5	1	2	6
d_j	5	13	20	5	30	21	29	25

Then to calculate the total completion time, maximum lateness, total earliness, total tardiness, and the total number of late jobs:

j	1	2	3	4	5	6	7	8
P_j	7	3	2	9	5	1	2	6
d_j	5	13	20	5	30	21	29	25
C_j	7	10	12	21	26	27	29	35
L_j	2	−3	−8	16	−4	6	0	10
E_j	0	3	8	0	4	0	0	0
T_j	2	0	0	16	0	6	0	10

Then: $\sum C_j = 7 + 10 + 12 + 21 + 26 + 27 + 29 + 35 = 167,\ L_{max} = 16, \sum E_j = 15, \sum T_j = 34, \sum U_j = 4.$

Single Machine Scheduling Problems: $1 / / \sum C_j$ Problem

This is the problem of sequencing n jobs on a single machine to minimize the total completion time. This problem is solved by the SPT (shortest processing time) rule. The jobs are sequenced in non-decreasing order of processing times P_j.

Example (5):

Solve the following $1//\sum C_j$ problem:

To minimize $\sum C_j$, we use the SPT rule as follows:

j	1	2	3	4	5	6	7	8
P_j	7	3	2	9	5	1	2	6

j	6	3	7	2	5	8	1	4
P_j	1	2	2	3	5	6	7	9
d_j	1	3	5	8	13	19	26	35

Then by SPT rule: $\sum C_j$=1+ 3 + 5 + 8 + 13 + 19 + 26 + 35 =110. That is the optimal schedule is s= (6,3,7,2,5,8,1,4) with $\sum C_j$=110.

$1 / / \sum w_j C_j$ Problem

This is the problem of sequencing n jobs on a single machine to minimize the weighted total completion time. This problem is solved by the SWPT (shortest weighted processing time) rule. The jobs are sequenced in non-decreasing order of processing times P_j / w_j

Example (6):

Consider the following schedule:

To minimize $\sum w_j C_j$, we must first find P_j / w_j for each job j:

j	1	2	3	4	5
P_j	6	10	12	18	4
w_j	2	4	3	3	4

j	1	2	3	4	5
P_j	6	10	12	18	4
w_j	2	4	3	3	4
P_j / w_j	3	2.5	4	6	1

Then, use the SWPT rule as follows:

j	5	2	1	3	4
P_j / w_j	1	2.5	3	4	6
P_j	4	10	6	12	18
w_j	4	4	2	3	3
C_j	4	14	20	32	50
$w_j C_j$	16	56	40	96	150

Then by SWPT: $\sum \boldsymbol{w_j C_j}$= 358. That is the optimal schedule is s= (5,2,1,3,4) with $\sum \boldsymbol{w_j C_j}$= 358 ($\sum \boldsymbol{w_j C_j}$= 498 for the original sequence).

1/ / L_{max} Problem

This is the problem of sequencing n jobs on a single machine to minimize the maximum lateness. This problem is solved by the EDD (earliest due date) rule. The jobs are sequenced in non-decreasing order of due dates d_j.

Example (7):

Consider the following schedule:

j	**1**	**2**	**3**	**4**
P_j	4	5	3	2
d_j	7	8	5	4

j	**4**	**3**	**1**	**2**
P_j	2	3	4	5
d_j	4	5	7	8
C_j	2	5	9	14
L_j	−2	0	2	6

To minimize L_{max} we use the EDD rule:

∴ L_{max} = 6 (for the original schedule L_{max} = 10).

The optimal schedule is s = (4,3,1,2) with L_{max} = 6.

1/ / $\sum U_j$ Problem

This is the problem of sequencing n jobs on a single machine to minimize the number of late jobs (minimize the total unit penalties). This problem is solved by Moore algorithm. Let E denote the set of early jobs and L denote the set of late jobs. The jobs of E are sequenced in EDD rule followed by the jobs of L.

Moore (and Hodgson) Algorithm

Step 1: Number the jobs in EDD order. Set $E = \phi, L = \phi, k = 0, t = 0$.

Step 2: Let $k = k + 1$. If $k > n$ go to step 4.

Step 3: Let $t = t + P_k$ and $E = E \cup \{k\}$. If $t < d_k$ go to step 2. If $t > d_k$ find $j \in E$ with P_j as large as possible and let $t = t - P_j$, $E = E - \{j\}, L = L \cup \{j\}$. Go to step 2.

Step 4: E is the set of early jobs and L is the set of late jobs.

Example (8):

Minimize $\sum U_j$ for the following schedule:

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
P_j	5	3	1	8	4	7	5	3

d_j	12	32	10	18	23	27	15	24

To minimize $\sum U_j$ we use Moore algorithm:

j	3	1	7	4	5	8	6	2
P_j	1	5	5	8	4	3	7	3
d_j	10	12	15	18	23	24	27	32
C_j	1	6	11	19				
C_j	1	6	11	*	15	18	25	28

$\therefore \sum U_j$=1, E= {3,1,7,5,8,6,2}, L = {4}. The optimal schedule is: s = (3,1,7,5,8,6,2,4) (in the original schedule $\sum U_j$ =3).

Example (9):

Minimize $\sum U_j$ for the following schedule:

j	1	2	3	4	5	6	7	8
P_j	4	2	7	6	4	7	5	5
d_j	12	27	10	15	30	22	8	28

Solution:

To minimize $\sum U_j$ we use Moore's algorithm:

j	7	3	1	4	6	2	8	5
P_j	5	7	4	6	7	2	5	4
d_j	8	10	12	15	22	27	28	30
C_j	5	12						
C_j	5	*	9	15	22	24	29	
C_j	5	*	9	15	*	17	22	26

Remark: 5th job (Job 6) is selected although it is early since it has the greatest P_j among all jobs in E.

$\therefore \sum U_j = 2$, $E = \{7,1,4,2,8,5\}$, $L = \{3,6\}$. The optimal schedule is: s= (7,1,4,2,8,5,3,6). Also, s= (7,1,4,2,8,5,6,3) is an optimal schedule.

Example (10):

Minimize $\sum U_j$ for the following schedule:

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
P_j	4	3	1	5	2	3	1	3
d_j	7	6	4	7	9	6	4	5

Solution:

To minimize $\sum U_j$ we use Moore's algorithm:

j	**3**	**7**	**8**	**2**	**6**	**1**	**4**	**5**
P_j	1	1	3	3	3	4	5	2
d_j	4	4	5	6	6	7	7	9
C_j	1	2	5	8				
C_j	1	2	*	5	8			
C_j	1	2	*	*	5	9		
C_j	1	2	*	*	5	*	10	
C_j	1	2	*	*	5	*	*	7

$\therefore \sum U_j = 4$, $E = \{3,7,6,5\}$, $L = \{8,2,1,4\}$. The optimal schedule is: s = (3,7,6,5,8,2,1,4).

بسم الله الرحمن الرحيم

قل هو الله أحد ۝ الله الصمد ۝ لم يلد ولم يولد ۝
ولم يكن له كفوا احد ۝

بسم الله الرحمن الرحيم

قل أعوذ برب الفلق

من شر ما خلق ۝ ومن شر
غاسق إذا وقب ۝ ومن
شر النفاثات في العقد ۝
ومن شر حاسد إذا حسد ۝

قل أعوذ برب الناس

ملك الناس ۝ إله الناس ۝
من شر الوسواس الخناس ۝
الذي يوسوس في صدور
الناس ۝ من الجنة والناس ۝