Scheduling Algorithms

Reading:

1. Survey: Scheduling Algorithms (1997)
David Karger, Cliff Stein, Joel Wein

2. Scheduling: Theory, Algorithms, and Systems
by Michael Pinedo

3. Scheduling Algorithms by Peter Brucker.

Scheduling Theory

In general:

A set of jobs needs to be processed by a set of machines. The jobs need to be scheduled on the machines in a way that satisfies some objective function.

Scheduling Theory

Example 1a:

Given a set of jobs, each job has known processing-time and deadline. How do we schedule the jobs on a single machine in a way that minimizes the number of late jobs (those completed after their deadline).

Example 1b:

The same on two identical machines.

Example 1c:

The same on m machines each having a different processing rate.

Scheduling Theory

Example 2a:

Each exam needs to be marked by three teachers (each checking a different question). The order in which the questions are marked is not important.

For each exam and question, we know how much time it takes to mark it. What is the best schedule if we want to minimize the completion time of the whole marking process?

Example 2b: The same, but the questions must be marked in some fixed order.

Example 2c: The same, but now it takes some (known) time to transfer a set of exams from one teacher to another.

Scheduling Theory - Notations

J_j – The j^{th} job.

p_j – length of J_j = how many processing units it requires.

r_j – release time of J_j = when does J_j is available for execution.

d_j – deadline (or due-date) for J_j = when do we need to complete its execution.

C_j – the completion time of J_j in a given schedule.

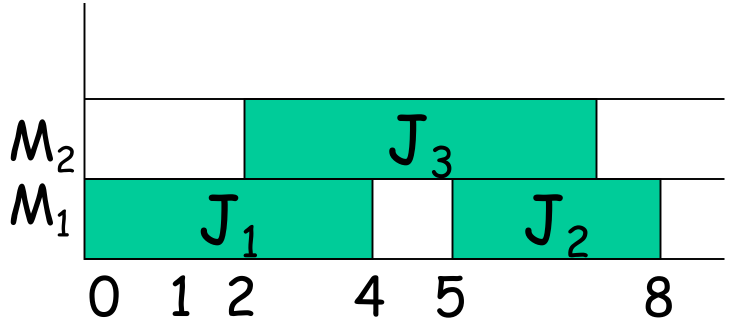
Example

Consider an instance with n=3 jobs J_1, J_2,J_3, on two machines M_1 and M_2.

$p_1 = 4$, $p_2 = 3$, $p_3 = 5$.

$r_1 = 0$, $r_2 = 5$, $r_3 = 1$.

A possible schedule:



$C_1 = 4$, $C_2 = 8$, $C_3 = 6$.

$C_{max} = 8$

Possible objective functions

Denote by K_j the cost associated with job j (can be many things).

There are two main types of objective functions:

1. Minimize the maximal cost: Min K_{max} = min $\max_i K_i$

Example ($K_j = C_j$): Min C_{max} : Minimize the makespan (last completion time).

2. Minimize the sum of costs: Min $\sum_j K_j$

Example: $\sum_j C_j$ – sum of completion times (same objective as average).

Objective functions based on the completion times

- min C_{max} (the makespan)
- min $\sum_j C_j$ (sum of completion times)

Sometimes each job is associated with weight, w_j – cost of spending one unit of time in the system.

- min $\sum_j w_j C_j$ (weighted sum of completion times)

When jobs have release times, the service time is defined as $F_j = C_j - r_j$

- min F_{max}, min $\sum_j F_j$, min $\sum_j w_j F_j$

Out of the service time, the waiting time is $W_j = F_j - p_j$

- min W_{max}, min $\sum_j W_j$, min $\sum_j w_j W_j$

Objective functions based on the due-dates

d_j – due-date of J_j

$L_j = C_j - d_j$ (Lateness)

$T_j = \max(0, L_j)$ (Tardiness) – more practical.

Possible objectives: Minimizing $T_{max}, L_{max}, \sum_j T_j, \sum_j L_j, \sum_j w_j T_j, \sum_j w_j L_j$

U_j - lateness indicator (=0 if $C_j \leq d_j$; 1 if $C_j > d_j$).

Possible objectives: minimize the number of late jobs given by $\sum_j U_j$. Other: $\sum_j w_j U_j$

Objective functions based on the system efficiency

$N_p(t)$ – number of jobs processed at time t.

$N_w(t)$ – number of jobs waiting at time t.

I_k - idle time of machine k during the interval $[0, C_{max}]$.

$Avg(N_p)$ = average number of processed jobs over the time interval $[0,C_{max}]$

$Avg(N_w)$ = average number of waiting jobs over the time interval $[0,C_{max}]$

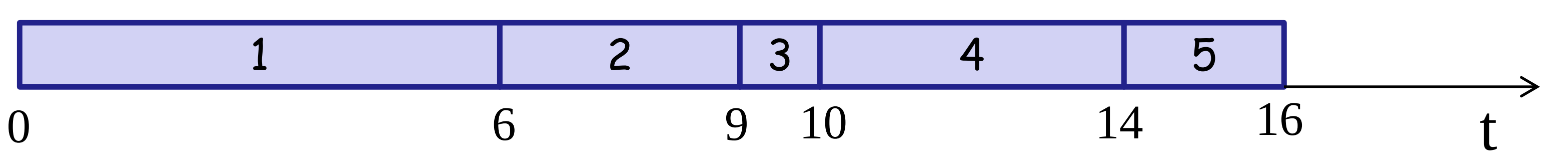
Possible objectives: Minimizing $Avg(N_w)$, $\sum_k I_k$.

Maximizing $Avg(N_p)$.

Example - in class exercise

An instance consists of 5 jobs and a single machine. Consider the schedule J_1-J_2-J_3-J_4-J_5 with no intended idle, and fill in the table.

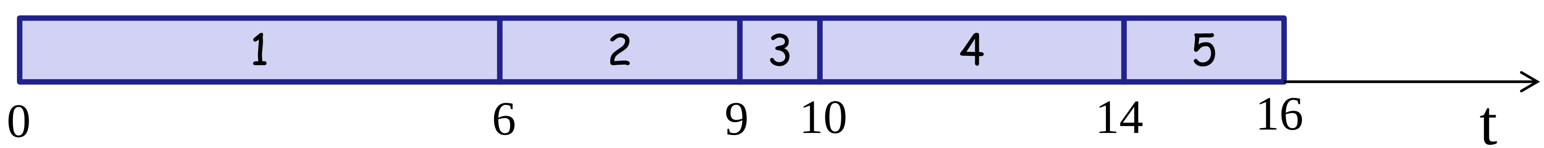
j	p_j	w_j	d_j	C_j	w_jC_j	L_j	T_j	U_j	w_jT_j	w_jU_j
1	6	1.5	7							
2	3	1	9							
3	1	1	6							
4	4	2	10							
5	2	1	11							
sum	16	----	----							
max	----	----	----							



Example - in class exercise

An instance consists of 5 jobs and a single machine. Consider the schedule J_1-J_2-J_3-J_4-J_5 with no intended idle, and fill in the table.

j	p_j	w_j	d_j	C_j	w_jC_j	L_j	T_j	U_j	w_jT_j	w_jU_j
1	6	1.5	7	6	9	-1	0	0	0	0
2	3	1	9	9	9	0	0	0	0	0
3	1	1	6	10	10	4	4	1	4	1
4	4	2	10	14	28	4	4	1	8	2
5	2	1	11	16	16	5	5	1	5	1
sum	16	----	----	55	72	12	13	3	17	4
max	----	----	----	16	28	5	5	1	8	2



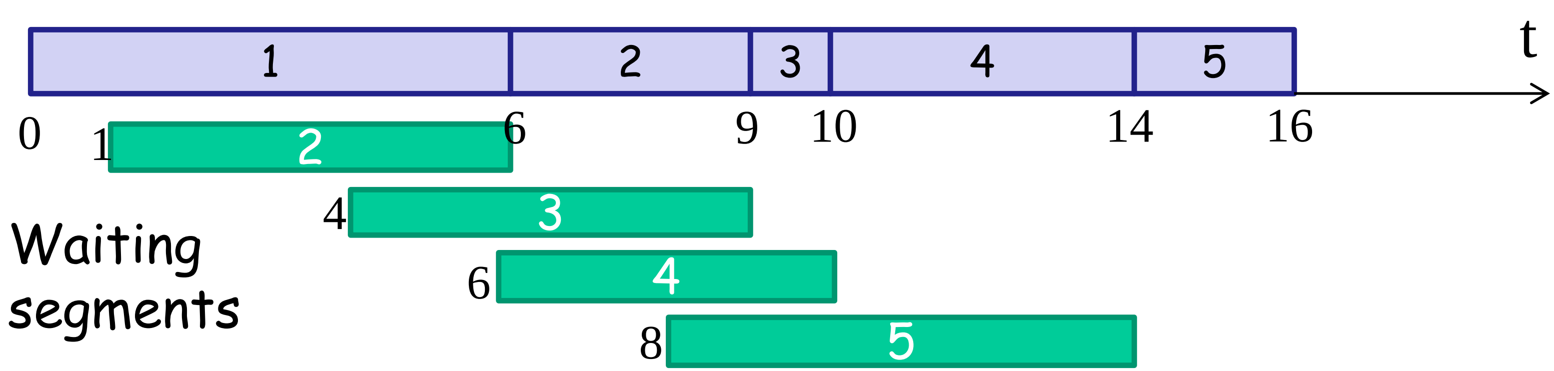
Example - in class exercise

Assume release times as given in the table.

What is Avg(N_w) in this schedule?

j	p_j	r_j	C_j
1	6	0	6
2	3	1	9
3	1	4	10
4	4	6	14
5	2	8	16

Total waiting time:
0+5+5+4+6=20.
Avg(N_w) = 20/16=1.25



Relations between different objective functions

Theorem 1: The following objective functions are equivalent (a schedule that is optimal for one is optimal also for the others):

(i) Min C_{max} (makespan)

(ii) Max Avg(N_p) (average # of processed jobs)

(iii) Min $\sum_k I_k$ (total idle time)

Proof: In class

Relations between different objective functions

Theorem 2: The following objective functions are equivalent (a schedule that is optimal for one is optimal also for the others):

(i) Min $\sum_j C_j$ (completion time)

(ii) Min $\sum_j F_j$ (service time)

(iii) Min $\sum_j W_j$ (waiting time)

(iv) Min $\sum_j L_j$ (lateness)

Proof: In class

Relations between different objective functions

Theorem 3: The following objective functions are equivalent (a schedule that is optimal for one is optimal also for the others):

(i) Min $\sum_j w_j C_j$ (completion time)

(ii) Min $\sum_j w_j F_j$ (service time)

(iii) Min $\sum_j w_j W_j$ (waiting time)

(iv) Min $\sum_j w_j L_j$ (lateness)

Proof: Similar to the proof of Theorem 2.

Scheduling Theory – more notations

A scheduling problem is defined by a triplet $\alpha|\beta|\gamma$.

Some possibilities for α:

1 - a single machine

P - identical parallel machines

Q - parallel machines with different rates.

R - unrelated parallel machines (specific processing time for each job and machines).

O – Open-shop scheduling

F – Flow-shop scheduling

J – Job-shop scheduling

Scheduling Theory – more notations

β - additional assumptions or constraints.

For example:

pmtn- preemptions allowed

prec- precedence constraints

r_j – release times

$t_j=1$ - all jobs have the same proc. time.

Set-up – there is a delay when jobs are switched on a machine (fixed or given)

γ- the objective function.

Examples

$1||\sum_j C_j$ - A single machine. Minimize average complete time.

$1|prec|L_{max}$ – A single machine, precedence constraints exist, minimize the maximal lateness.

$Q|pmtn|C_{max}$ – Parallel machines with different rates, minimize the makespan, jobs can be preempted.

Algorithms for a Single Machine

We will see that simple greedy algorithms are optimal for some scheduling problems on a single machine.

Other problems, some of them look really simple, are NP-hard.

Shortest Processing Time (SPT) Rule

The problem: $1||\sum_j C_j$ (average completion time)

SPT Rule: Sort the jobs such that $p_1 \leq p_2 \leq \ldots \leq p_n$.
Process the jobs according to this order.

Example: 5 jobs of lengths 9, 6, 3 ,8, 1

$\sum_j C_j$ in original order: 9+15+18+26+27= 95

SPT order: 1, 3, 6, 8 , 9

$\sum_j C_j$ in SPT order: 1+4+10+18+27= 60

Shortest Processing Time (SPT) Rule

The problem: $1||\sum_j C_j$ (average completion time)

Theorem: SPT is optimal for $1||\sum_j C_j$

Proof:

$C_1 = p_1$; $C_2=p_1+p_2$; $C_j = \sum_{i \leq j} p_i$

$\sum_j C_j = np_1 + (n-1)p_2 + (n-2)p_3 + \ldots + p_n = \boxed{\sum_j (n-j+1)p_j .}$

This is a product of two vectors. The first one (n,n-1,…,1) is decreasing. To get a minimal result, the other vector needs to be non-decreasing.

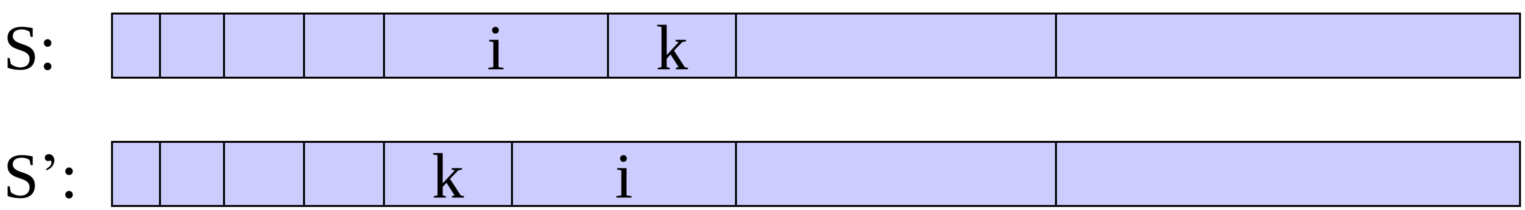
Optimality of SPT for $1||\sum_i C_j$

An alternative proof (using exchanging argument):

Assume that S is an optimal schedule which is not according to SPT.

For some pair J_i, J_k of adjacent jobs, J_k is scheduled after J_i while $p_i > p_k$.

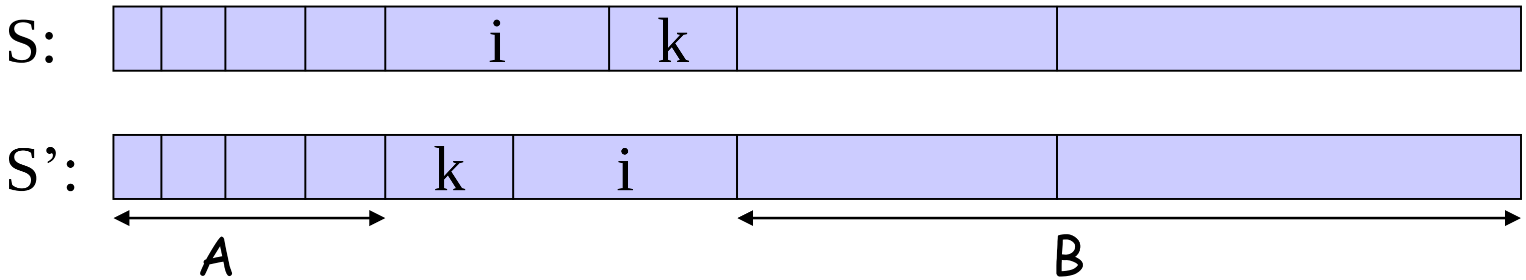
Build a new schedule S' in which we swap the schedules of J_i,J_k (the other jobs are as in S).



We show that S' is a better schedule:

Optimality of SPT for $1||\sum_i C_j$

Claim: In S′ $\sum_j C_j(S') < \sum_j C_j(S)$.



Proof: A: jobs starting before J_i and J_k.

B: jobs starting after J_i and J_k.

$$\sum_j C_j(S) = \sum_{j\in A} C_j + \sum_{j\in B} C_j + C_i + C_k =$$

$$= \sum_{j\in A} C_j + \sum_{j\in B} C_j + (p_A+p_i)+(p_A+p_i+p_k)$$

$$\sum_j C_j(S') = \sum_{j\in A} C'_j + \sum_{j\in B} C'_j + C'_i + C'_k =$$

$$= \sum_{j\in A} C_j + \sum_{j\in B} C_j + (p_A+p_k)+(p_A+p_k+p_i) =$$

$$= \sum_j C_j(S) + p_k - p_i < \sum_j C_j(S) \text{ (since } p_i > p_k).$$

■

Variants of SPT

1. The problem: $1|r_j, pmtn|\sum_j C_j$

Shortest Remaining Processing Time (SRPT) Rule:

At each moment, process the job with the shortest remaining processing time (can preempt a currently processed job).

Complexity: Should keep a sorted list of all available jobs. $O(\log n)$ for any released job + $O(\log n)$ for any preempted job. The total number of preemptions is at most n. → $O(n \log n)$ in total.

Theorem: SRPT rule is optimal for $1|r_j, pmtn|\sum_j C_j$

Proof: Exchanging argument.

Variants of SPT

2. The problem: $1||\sum_j w_jC_j$

Weighted Shortest Processing Time (WSPT) Rule:

Sort the jobs such that $p_1/w_1 \leq p_2/w_2 \leq \ldots \leq p_n/w_n$.

Process the jobs on the machine according to this order.

Theorem: WSPT is optimal for $1||\sum_j w_jC_j$.

Proof: Exchange argument.

Single Machine. Set-up times.

The problem: $1|\text{set-up}|\, C_{max.}$

For each pair of jobs, s_{ij} is the set-up time required between processing i and j.

Note: without set-up times, or with identical set-up times ($\forall i,j\ s_{ij} = s$), any order is optimal ($C_{max} = \sum_j p_j + (n-1)s$).

For arbitrary set-up times. The problem is NP-hard.

Proof: Reduction from the traveling salesman problem.

EDD for Minimizing Tardiness.

For an instance with due-dates and a given schedule

$L_j = C_j - d_j$ (Lateness)

$T_j = \max(0, L_j)$ (Tardiness)

Possible objectives: Minimizing T_{max}, L_{max}, $\sum_j T_j$, $\sum_j L_j$

EDD Rule (earliest due-date): Sort the jobs such that $d_1 \le d_2 \le \dots \le d_n$. Process the jobs on the machine according to this order.

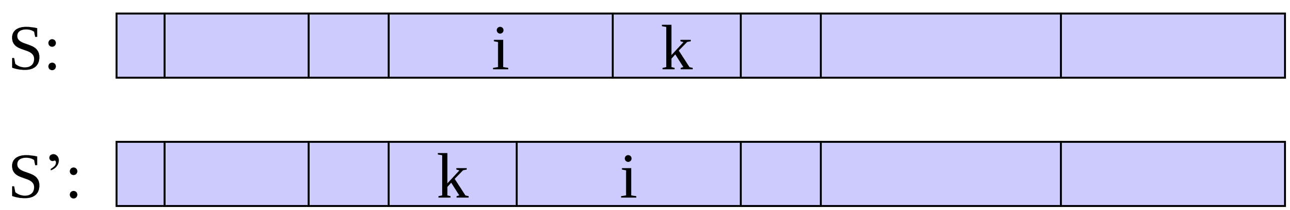
Theorem: EDD is optimal for $1||T_{max}$ and $1||L_{max}$

Optimality of EDD for $1||L_{max}$

Assume that S is an optimal schedule which is not according to EDD.

For some pair J_i, J_k of adjacent jobs, J_k is scheduled after J_i while $d_i > d_k$.

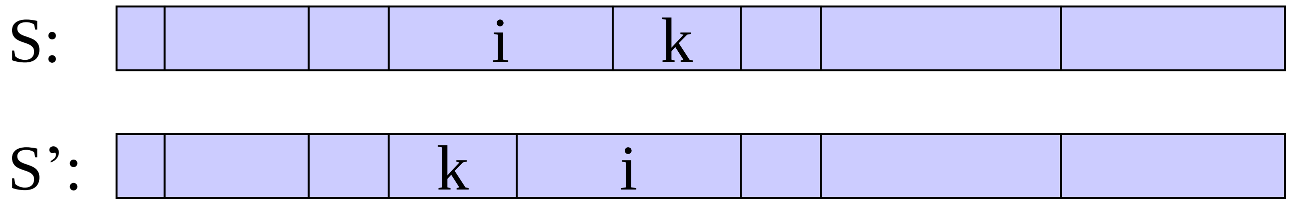
Build a new schedule S' in which we swap the schedules of J_i,J_k (the other jobs are as in S).



We show that $L_{max}(S') \leq L_{max}(S)$:

Optimality of EDD for $1||L_{max}$

Claim: In S' $L_{max}(S') \leq L_{max}(S)$:



Proof:

- $L_k(S') = C_k(S')-d_k < C_k(S)-d_k = L_k(S)$.
- $L_i(S') = C_i(S')-d_i = C_k(S)-d_i < C_k(S)-d_k = L_k(S)$

$\rightarrow \max(L_k(S'), L_i(S')) < \max(L_k(S), L_i(S))$

Let $L = \max\{L_j \mid j \text{ is not } i \text{ or } k\}$.

L is identical in S and S'.

$L_{max} = \max\{L, L_i, L_k\}$.

$L_{max}(S') \leq L_{max}(S)$ ∎

Optimality of EDD for $1||T_{max}$

Theorem: EDD is optimal for $1||T_{max}$

Proof:

Using the same exchange argument as in the previous proof. We get:

$T_{max}(S) = \max(0, L_{max}(S)) \geq \max(0, L_{max}(S')) = T_{max}(S')$.

$\blacksquare$

EDD for Minimizing Tardiness.

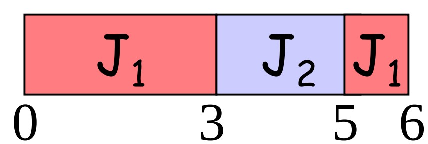
The problem: $1|r_j, pmtn|T_{max}$.

EDD rule: Process the job with minimal due-date among the jobs that are available.

- When a new job with an early due-date is released we might preempt the currently processed job.

Example: $r_1=0$, $p_1=4$, $d_1=6$.
$r_2=3$, $p_2=2$, $d_2=5$.

The EDD schedule:



No late jobs.

Theorem: EDD is optimal for $1|r_j, pmtn|T_{max}$

and for $1|r_j, pmtn|\sum_j T_j$

Proof: Exchange argument.

Minimizing Tardiness.

The problem: $1|r_j|\sum_j T_j$

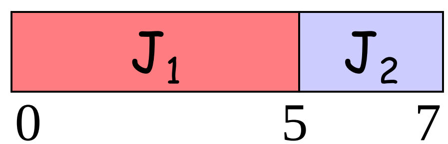
Preemptions are not allowed (preempted jobs must be rescheduled).

The problem is NP-hard.

Intended idle might be useful.

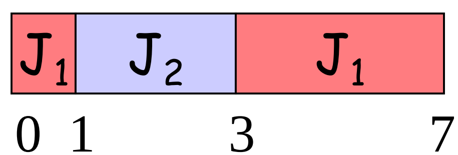
Example: $r_1=0,\ p_1=5,\ d_1=7.$
$r_2=1,\ p_2=2,\ d_2=3.$

With no intended idle



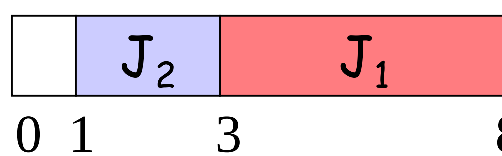
$\sum_j T_j = 4$

With preemptions



$\sum_j T_j = 0$

With idle, no pmtn



$\sum_j T_j = 1$

Minimizing Tardiness with Release Dates and No Preemptions.

Theorem: $1|r_j|T_{max}$ is NP-hard.

Proof: A reduction from Partition.

The partition problem:

Input: a set of n numbers, $A = \{a_1, a_2, \ldots, a_n\}$, such that $\sum_{j \in A} a_j = 2B$.

Output: Is there a subset A' of A such that $\sum_{j \in A'} a_j = B$?

Example: $A=\{5, 5, 7, 3, 1, 9, 10\}$; $B=20$

A possible partition: $A'=\{10,5,5\}$, $A-A'=\{7,3,1,9\}$

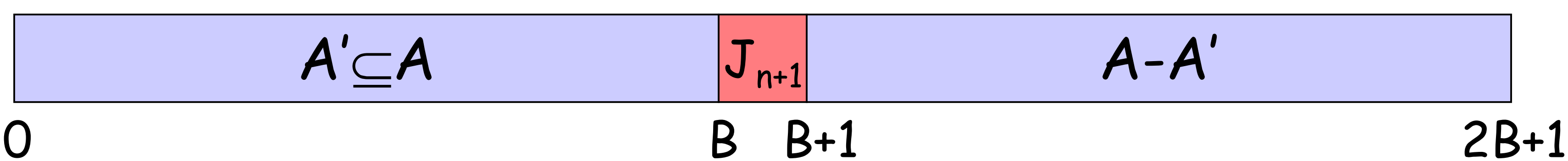
Minimizing Tardiness with Release Dates and No Preemptions.

Hardness proof for $1|r_j|T_{max}$:

Given an instance for partition, $A = \{a_1, a_2, \ldots, a_n\}$, s.t. $\sum_j a_j = 2B$, we build an instance for $1|r_j|T_{max}$ such that $T_{max} = 0$ if and only if A has a partition:

For each item, $a_j \in A$, we have a job j with $p_j = a_j$, $r_j = 0$, and $d_j = 2B+1$. In addition, we have the job, J_{n+1}, with $p_{n+1} = 1$, $r_{n+1} = B$, and $d_{n+1} = B+1$.

- To achieve $T_{n+1} = 0$, J_{n+1} must be scheduled in $[B, B+1]$



- The schedule of J_{n+1} induces a partition

Moore's Algorithm for $1||\sum_j U_j$

The objective: minimize the number of late jobs.

U_j is the lateness indicator (=0 if $C_j \leq d_j$; 1 if $C_j > d_j$).

The problem: $1||\sum_j U_j$

An optimal algorithm (Moore):

1. Order the jobs according to EDD rule (into A^*). The set R^* is empty.
2. If no job in A^* is late. A^*R^* is an optimal order.
3. Else, let k be the first job to be late in A^*.
4. Move to R^* the longest job among the first k jobs in A^*.
5. Update the completion times of jobs in A^*. Go to step 2.

Moore's Algorithm for $1||\sum_j U_j$ (example)

1. Order the jobs according to EDD rule (into A^*). The set R^* is empty.

j	p_j	d_j
1	1	2
2	5	7
3	3	8
4	7	11
5	9	13

A^*= {1-2-3-4-5} , R^* = { }.

2. According to this order, J_3 is the first to be late (C_3=9).

3. The longest job among the first three is J_2.

4. We move J_2 to R^*.

A^*= {1-3-4-5}, R^* = {2}

Moore's Algorithm for $1||\sum_j U_j$ (cont')

j	p_j	d_j
1	1	2
3	3	8
4	7	11
5	9	13

$A^* = \{1\text{-}3\text{-}4\text{-}5\}$, $R^* = \{2\}$

2. According to this order, J_5 is to first to be late ($C_5=20$).

3. The longest job among the first four (J_5 is the 4^{th} in A^*) is J_5.

4. We move J_5 to R^*.

j	p_j	d_j
1	1	2
3	3	8
4	7	11

$A^* = \{1\text{-}3\text{-}4\}$, $R^* = \{2, 5\}$

Now, no job in A^* is late.

$A^*R^* = 1\text{-}3\text{-}4\text{-}2\text{-}5$ is optimal

$\sum_j U_j = 2$