IO543: Scheduling

Lecture 3

Brian Denton
Associate Professor, IOE

IOE Bld 2893
btdenton@umich.edu

Today

Goals:

- Application of Complexity
- Solve $1| \ |\sum w_j C_j$ and $1| \, prec \, |\sum w_j C_j$

Complexity Relationships

- There are many situations where an algorithm for one problem can be applied to another

- One problem **reduces** to another if it is a **special case** of the other

<u>Example:</u> $1|\ |\sum C_j$ reduces to $1|\ |\sum w_j C_j$ by setting $w_j = 1$ for all j. This is expressed as

$$1|\ |\sum C_j \ \propto\ 1|\ |\sum w_j C_j$$

In-Class Assignment

Work in groups of 2-3. Draw the complexity hierarchy for these problems (without looking at Pinedo). Provide a sentence explaining each relationship

- $1| \ |C_{max}$
- $P2| \ |C_{max}$
- $F2| \ |C_{max}$
- $Jm| \ |C_{max}$
- $FFc \ | \ |C_{max}$

Hand in your answers (all names on one sheet)

Weighted Completion Time

Consider the generalization:

$$1|\ |\sum C_j \rightarrow 1|\ |\sum w_j C_j$$

with the following processing times and job weights:

Jobs	1	2	3	4
w_j	2	4	8	12
p_j	6	4	2	1

What is the optimal sequence for this instance?

Single Machine Scheduling

The Weighted Shortest Processing Time first (WSPT) rule sorts jobs in decreasing order of w_j/p_j

<u>Theorem 3.1.1 (Pinedo)</u> The WSPT rule is optimal for $1|\ |\sum w_j C_j$

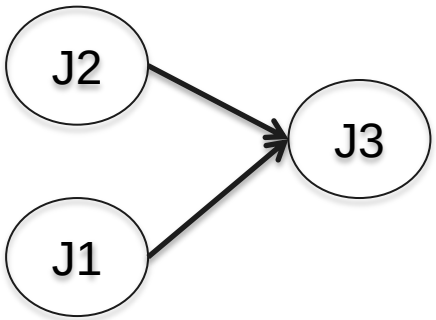
Proof: Completed in class

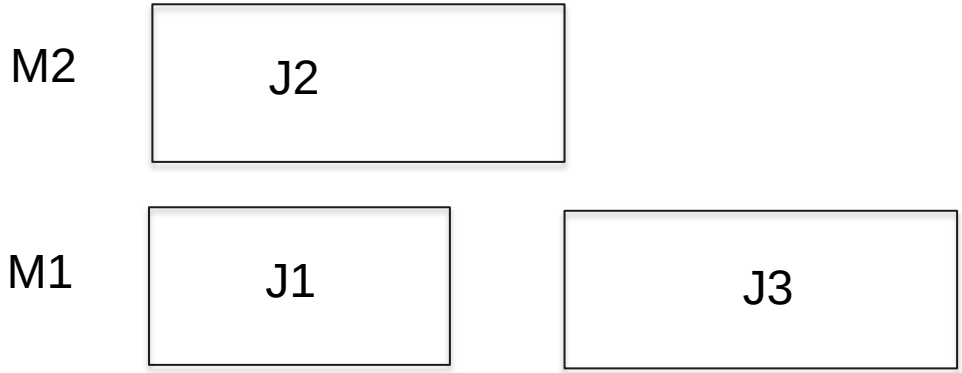
Therefore the computation time for $1|\ |\sum w_j C_j$ is *O(nlog(n)).*

Types of Schedules

- **Delay** vs **Non-Delay** Schedules
  - A schedule is non-delay if no machine is intentionally kept idle while an operation is waiting for processing
  - Example: Non-delay schedule

Precedence constraints:



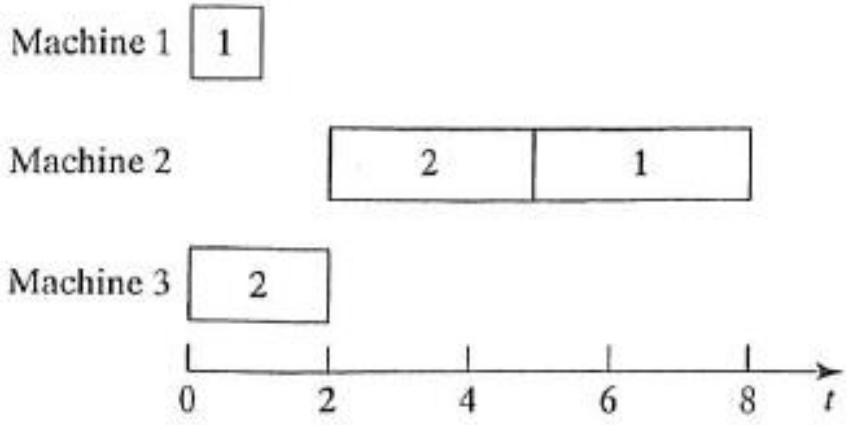


Types of Schedules

Active Schedules

- A schedule is **active** if it is not possible to construct another schedule by reordering jobs on a machine with at least one job finishing earlier and no job finishing later

- Example of an active schedule:

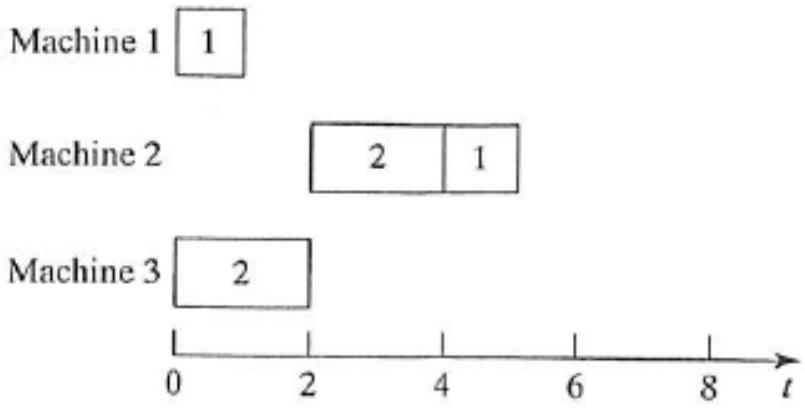


Types of Schedules

Semi-Active Schedules

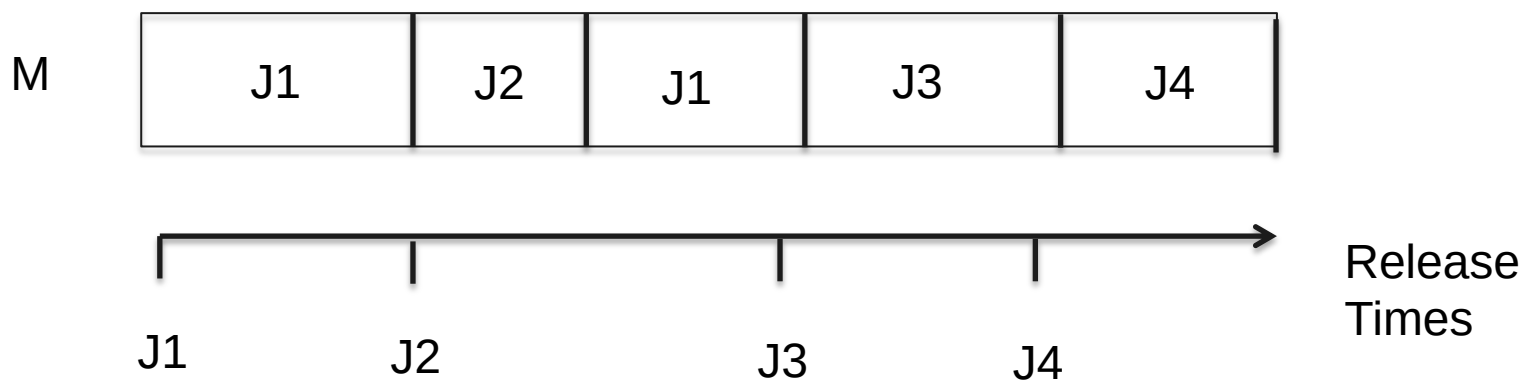
- A schedule is **semi-active** if no job can be completed earlier without changing the order of jobs

- Example of a semi-active schedule that is not active:



Types of Schedules

- **Preemptive** vs **Non-Preemptive** Schedules
  - A schedule is non-preemptive if every job that starts is completed before any other job starts
  - Example of a preemptive schedule



Question

What is an application context for which the following could arise?

1) A preemptive schedule

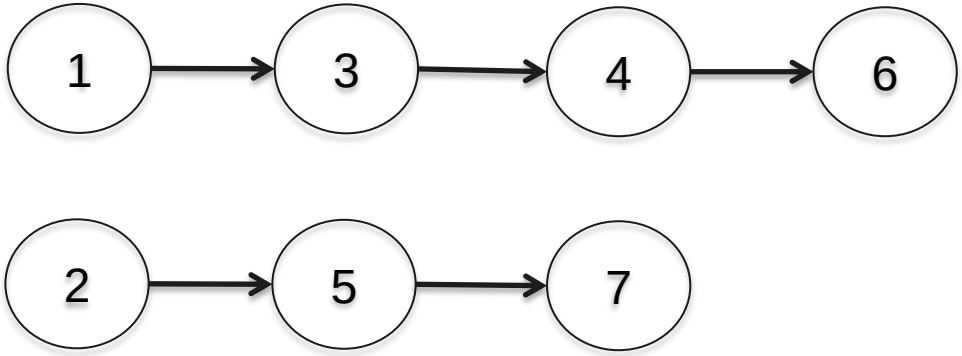
2) A delay schedule

Scheduling with Chains

Consider $1| \ |\sum w_j C_j$ with precedence constraints based on **chains**.

Definition: A **chain** is a sequence of jobs with serial precedence constraints

Example:



Scheduling with Chains

If you have to complete one chain before starting another…

<u>Lemma (3.1.2 Pinedo): If</u>

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

Then it is optimal to process the chain {1,…,k} before the chain {k+1, …,n}.

Scheduling with Chains

<u>Definition:</u> ρ(1,…,k) is referred to as the **ρ –factor** of chain {1,…,k} defines as:

$$\rho(1,\dots,k) = max_{1\le l\le k}\left\{\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j}\right\}$$

The job that determines the ρ –factor is denoted by l^*

Scheduling with Chains

The following is useful for designing an algorithm to solve $1| \ |\sum w_j C_j$ with chain precedence constraints

<u>Lemma (3.1.3 Pinedo)</u>: If job l^*determines $\rho(1, \ldots, k)$, then there exists an optimal sequence that processes jobs $1, \ldots, l^*$ one after another without interruption.

Scheduling with Chains

The following algorithm minimizes total weighted completion time in the presence of chains.

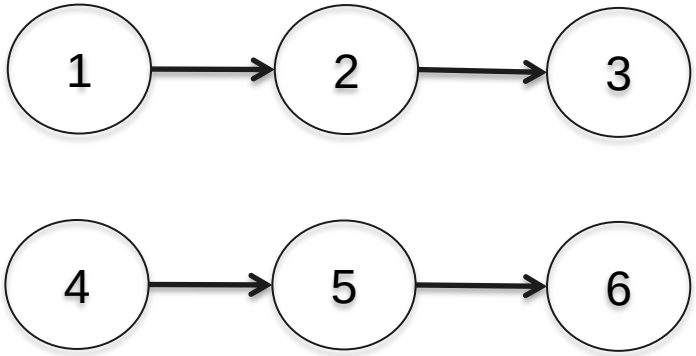
<u>Algorithm 3.1.4:</u>
Step 1: compute the ρ-factor for all pending chains; If no pending chains then stop.

Step 2: Select the chain with the highest ρ-factor and process all jobs up to l^* in the chain without interruption; return to Step 1.

Example: Scheduling with Chains

Solve the following instance of $1|\,prec\,|\sum w_j C_j$

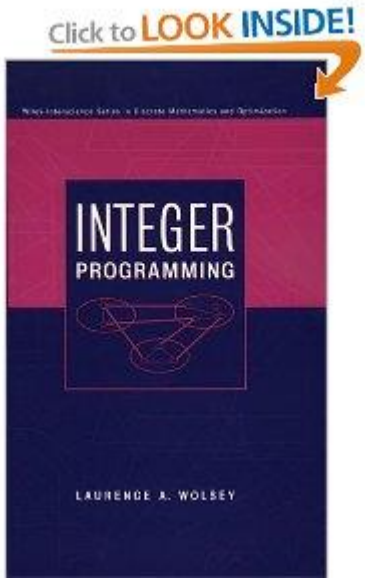
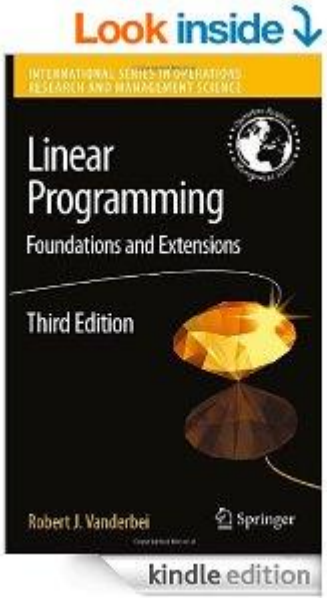
Precedence Relationships:



Weights and Processing Time:

Jobs	1	2	3	4	5	6
w_j	3	1	2	5	7	2
p_j	3	2	2	4	5	3

For Next Time

Read section 2.4 of Pinedo on complexity hierarchies

Review integer programming and **branch-and-bound**