• Today
  — Scheduling: Classification
    Ref: “Scheduling Algorithm” Peter Brayker Book
  — Multiprocessor Scheduling: List PTAS
    Ref: “job scheduling” by uwe schweigelshohn
• Tomorrow
  — Distributed Scheduling
  — Cilk Programming and Work Stealing
  — Scheduling in 2D NOC and 3D NOC

Scheduling Problems
• Find time slots in which activities (or jobs) should be processed under given constraints.
• Constraints
  — Resource constraints
  — Precedence constraints between activities.
• A quite general scheduling problem is
  — Resource Constrained Project Scheduling Problem (RCPSP)

Resource Constraints Project Scheduling Problem
• We have
  — Activities $j = 1, ..., n$ with processing times $p_j$.
  — Resources $k = 1, ..., r$. A constant amount of $R_k$ units of resource $k$ is available at any time.
  — During processing, every activity $j$ occupies $r_k$ units of resource $k$ for $k = 1, ..., r$.
  — Precedence constraints $i \rightarrow j$ between some activities $i, j$ with the meaning that activity $j$ cannot start before $i$ is finished.

RCPSP
• Objective: Determine starting times $S_j$ for all activities $j$ in such a way that
  — at each time $t$ the total demand for resource $k$ is not greater than the availability $R_k$ for $k = 1, ..., r$.
  — the given precedence constraints are fulfilled, i.e. $S_i + p_i \leq S_j$ if $i \rightarrow j$.

• Some objective function $f(C_1, ..., C_n)$ is minimized where $C_j = S_j + p_j$ is the completion time of activity $j$.
• The fact that activities $j$ start at time $S_j$ and finish at time $S_j + p_j$ implies that the activities $j$ are not preempted.
• We may relax this condition by allowing preemption (activity splitting).
RCPSP: An Example

- Consider a project with \( n = 4 \) activities, \( r = 2 \)
- resources with capacities \( R_1 = 5 \) and \( R_2 = 7 \), a
- precedence relation \( 2 \rightarrow 3 \) and the following data:

\[
\begin{array}{c|cccc}
 i & 1 & 2 & 3 & 4 \\
 \rho_i & 4 & 3 & 5 & 8 \\
r_{ij} & 2 & 1 & 2 & 2 \\
r_{ji} & 3 & 5 & 3 & 4 \\
\end{array}
\]

A corresponding schedule with minimal makespan

Applications of Scheduling

- Production scheduling
- Robotic cell scheduling
- Computer processor scheduling
- Timetabling
- Personnel scheduling
- Railway sc
- Air traffic control, Etc.

Machine Scheduling Problems and their Classification

- Most machine scheduling problems are special cases of the RCPSP.
  - Single machine problems,
  - Online Problem: FCFS, SJF, SRF, RR...
  - Parallel machine problems, and
  - Shop scheduling problems.

Single machine problems

- We have \( n \) jobs \( j = 1, \ldots, n \) to be processed on a single machine. Additionally precedence constraints between the jobs may be given.
- This problem can be modeled by an RCPSP with \( r = 1 \), \( R_1 = 1 \), and \( r_{ji} = 1 \) for all jobs \( j \).

Parallel Machine Problems

- We have jobs \( j \) as before and \( m \) identical machines \( M_1, \ldots, M_m \).
- The processing time for \( j \) is the same on each machine.
- One has to assign the jobs to the machines and to schedule them on the assigned machines.
- This problem corresponds to an RCPSP with \( r = 1 \), \( R_1 = m \), and \( r_{ji} = 1 \) for all jobs \( j \).
Parallel Machine Problems

- For unrelated machines the processing time $p_{jk}$ depends on the machine $M_k$ on which $j$ is processed.
- The machines are called uniform if $p_{jk} = \frac{p_j}{r_k}$.
- In a problem with multi-purpose machines a set of machines $\mu_j$ is associated with each job $j$ indicating that $j$ can be processed on one machine in $\mu_j$ only.

Shop Scheduling Problems

- A job-shop problem is a general shop scheduling problem with chain precedence constraints of the form $O_{ij} \rightarrow O_{ij} \rightarrow \ldots \rightarrow O_{n(j)i}$.
- A flow-shop problem is a special job-shop problem with $n(j) = m$ operations for $j = 1, \ldots, n$ and $\mu_{ij} = M_i$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Classification of Scheduling Problems

Classes of scheduling problems can be specified in terms of the three-field classification $\alpha | \beta | \gamma$ where

- $\alpha$ specifies the machine environment,
- $\beta$ specifies the job characteristics, and
- $\gamma$ describes the objective function(s).
Machine Environment: α

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Single Machine</td>
</tr>
<tr>
<td>P</td>
<td>Parallel Identical Machine</td>
</tr>
<tr>
<td>Q</td>
<td>Uniform Machine</td>
</tr>
<tr>
<td>R</td>
<td>Unrelated Machine</td>
</tr>
<tr>
<td>MPM</td>
<td>Multipurpose Machine</td>
</tr>
<tr>
<td>J</td>
<td>Job Shop</td>
</tr>
<tr>
<td>F</td>
<td>Flow Shop</td>
</tr>
</tbody>
</table>

If the number of machines is fixed to m we write

Pm, Qm, Rm, MPMm, Jm, Fm, Om.

Job Characteristics: β

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmtn</td>
<td>preemption</td>
</tr>
<tr>
<td>rj</td>
<td>release times</td>
</tr>
<tr>
<td>dj</td>
<td>deadlines</td>
</tr>
<tr>
<td>pm</td>
<td>p_j = 1 or p_j = p or p_j \in {1, 2} restricted processing times</td>
</tr>
<tr>
<td>prec</td>
<td>arbitrary precedence constraints</td>
</tr>
<tr>
<td>intree</td>
<td>intree (or outtree) precedences</td>
</tr>
<tr>
<td>intree</td>
<td>intree (or outtree) precedences</td>
</tr>
<tr>
<td>chains</td>
<td>chain precedences</td>
</tr>
<tr>
<td>series-parallel</td>
<td>a series-parallel precedence graph</td>
</tr>
</tbody>
</table>

Objective Functions: γ

Two types of objective functions are most common:

- **bottleneck objective functions**
  
  \[ \max \{ f_j(C_j) \mid j = 1, \ldots, n \} , \text{ and} \]

- **sum objective functions**
  \[ \sum f_j(C_j) = f_1(C_1) + f_2(C_2) + \ldots + f_n(C_n) . \]

  \[ C_j \text{ is completion time of task } j \]

Objective Functions: γ

- **Σ U_j (number of late jobs) and Σ ω_j U_j (weighted number of late jobs) where U_j = 1 if C_j > d_j and U_j = 0 otherwise.**

- **Σ T_j (sum of tardiness) and Σ ω_j T_j (weighted sum of tardiness/lateness) where the tardiness of job j is given by**

  \[ T_j = \max \{ 0, C_j - d_j \} . \]

Examples of Scheduling Problem

- 1 | prec; \( p_j = 1 \) | Σ \( \omega_j C_j \)
- P2 | | \( C_{\text{max}} \)
- P | \( p_j = 1 \); \( r_j \) | Σ \( \omega_j U_j \)
- R2 | | \( C_{\text{max}} \)
- J3 | \( n = 3 \) | \( C_{\text{max}} \)
- F | \( p_j = 1 \); \( \text{outtree} \); \( r_j \) | Σ \( C_j \)
- Om | \( p_j = 1 \) | Σ \( T_j \)
Polynomial algorithms

- A problem is called polynomially solvable if it can be solved by a polynomial algorithm.

Example
1 | | \( \sum \alpha_i C_i \) can be solved by scheduling the jobs in an ordering of non-increasing \( \alpha_i/p_i \) - values. Complexity: \( O(n \log n) \)

Precedence constraints (\( prec \))

Before certain jobs are allowed to start processing, one or more jobs first have to be completed.

Definition
- Successor
- Predecessor
- Immediate successor
- Immediate predecessor
- Transitive Reduction

In-tree (Out-tree)

- In-forest (Out-forest)
- Opposing forest
  - Interval orders
  - Series-parallel orders
  - Level orders
• Processor Environment
  – m identical processors are in the system.

• Job characteristics
  – Precedence constraints are given by a precedence graph;
  – Preemption is not allowed;
  – The release time of all the jobs is 0.

• Objective function
  – $C_{max}$: the time the last job finishes execution.
  – If $c_j$ denotes the finishing time of $J_j$ in a schedule $S$,
    $$C_{max} = \max_{1 \leq j \leq n} c_j$$

**Classification**

Due to the number of processors

• Number of processors is a variable (m)
  $P_m | \text{prec}, p_j = 1 | C_{max}$

• Number of processors is a constant (k)
  $P_k | \text{prec}, p_j = 1 | C_{max}$

**Theorem 1**

$P_m | \text{prec}, p_j = 1 | C_{max}$ is NP-complete.

1. Ullman (1976)
   3SAT $\leq P_m | \text{prec}, p_j = 1 | C_{max}$

2. Lenstra and Rinnoy Kan (1978)
   $k$-clique $\leq P_m | \text{prec}, p_j = 1 | C_{max}$

**Corollary 1.1**

The problem of determining the existence of a schedule with $C_{max} \leq 3$ for the problem $P_m | \text{prec}, p_j = 1 | C_{max}$ is NP-complete.

**Theorem 2**

$P_m | p_j = 1, \text{SP} | C_{max}$ is NP-complete.

SP: Series - parallel

**Theorem 3**

$P_m | p_j = 1, \text{OF} | C_{max}$ is NP-complete.

OF: Opposing - forest

**PTAS Algorithms: Pk | prec, pj = 1 | C_{max}**

• PTAS: Polynomial Time Approximation Scheme
• Approximation List scheduling policies
  – Graham's list algorithm
  – HLF algorithm
  – MSF algorithm
**List scheduling policies**

- Set up a priority list $L$ of jobs.
- When a processor is idle, assign the first ready job to the processor and remove it from the list $L$.

$L = (J_3, J_5, J_4, J_2, J_6, J_1, J_0)$

**Graham’s list algorithm**

- Graham first analyzed the performance of the simplest list scheduling algorithm.
- List scheduling algorithm with an arbitrary job list is called Graham’s list algorithm.
- Approximation ratio for $P_k \mid \text{prec}, p_j = 1 \mid C_{\text{max}}$
  \[ \delta = 2 - \frac{1}{k}. \] (Tight bound!)
  - Approximation ratio is $\delta$ if for each input instance, the makespan produced by the algorithm is at most $\delta$ times of the optimal makespan.

**HLF/CP algorithm**

- T. C. Hu (1961), Critical Path Algorithm or Hu’s algorithm
- Algorithm
  - Assign a level $h$ to each job.
    - If job has no successors, $h(j)$ equals 1.
    - Otherwise, $h(j)$ equals one plus the maximum level of its immediate successors.
  - Set up a priority list $L$ by nonincreasing order of the jobs’ levels.
  - Execute the list scheduling policy on this level based priority list $L$.

$L = (J_{15}, J_{13}, J_{11}, J_{10}, J_9, J_8, J_7, J_6, J_5, J_4, J_3, J_2, J_1)$

**HLF/CP algorithm**

- Time complexity
  $O(|V| + |E|)$ (where $|V|$ is the number of jobs and $|E|$ is the number of edges in the precedence graph)
- Theorem (Hu, 1961)
  The HLF algorithm is optimal for $P_k \mid p_j = 1, \text{in-tree (out-tree)} \mid C_{\text{max}}$.
  
  **Corollary**
  The HLF algorithm is optimal for $P_k \mid p_j = 1, \text{in-forest (out-forest)} \mid C_{\text{max}}$.

- N.F. Chen & C.L. Liu (1975)
  The approximation ratio of HLF algorithm for the problem with general precedence constraints:
  \[ \delta_{\text{HLF}} \leq 4/3. \] (Tight!)
  If $k = 2$, $\delta_{\text{HLF}} \leq 4/3$.
  If $k \geq 3$, $\delta_{\text{HLF}} \leq 2 - 1/(k-1)$. 
**Most Successors First Algorithm (MSF)**

- **Algorithm:**
  - Set up a priority list $L$ by nonincreasing order of the jobs' successors numbers.
    - (i.e. the job having more successors should have a higher priority in $L$ than the job having fewer successors)
  - Execute the list scheduling policy based on this priority list $L$.

**Time complexity** $O(V+E)$

**Theorem** (Papadimitriou and Yannakakis, 1979)
The MSF algorithm is optimal for $P_k | p_j = 1$, interval $| C_{\text{max}}$.

**Theorem** (Moukrim, 1999)
The MSF algorithm is optimal for $P_k | p_j = 1$, quasi-interval $| C_{\text{max}}$.

Prove out of Syllabus

---

**Linear Programming Solution to Scheduling Problem**

- $L = \{ j \}$ whether job $j$ is scheduled in machine $i$
- $p_i, \ldots, p_j$ for each job $j$
- $p_i, \ldots, p_j$ for each machine $i$
- $\sum_i p_i$ for each machine $i$
- $\sum_j p_i$ for each job $j$, machine $i$

- Each job is scheduled in one machine.
- Each machine can finish its jobs by time $T$.

---

**Minimum makespan scheduling: Arbitrary List**

- **Algorithm**
  - 1. Order the jobs arbitrarily.
  - 2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.

- Above algorithm achieves an approximation guarantee of 2 for the minimum makespan problem.
Load Balancing: List Scheduling

- **Machine 1**
  - A
  - F
  - H

- **Machine 2**
  - B
  - C
  - I

- **Machine 3**
  - D
  - E
  - J

---

Load Balancing: List Scheduling

- **Machine 1**
  - B
  - G
  - J

- **Machine 2**
  - C

- **Machine 3**
  - D
  - E

---

Load Balancing: List Scheduling

- **Machine 1**
  - A

- **Machine 2**
  - B
  - C

- **Machine 3**
  - D
  - E

---

Load Balancing: List Scheduling

- **Machine 1**
  - A

- **Machine 2**
  - B
  - C

- **Machine 3**
  - D
  - E

---

Load Balancing: List Scheduling

- **Machine 1**
  - A
  - E

- **Machine 2**
  - B
  - D

- **Machine 3**
  - C
**List Scheduling is "2-approximation"** (Graham, 1966)

**Algorithm:** List scheduling

Basic idea: In a list of jobs, schedule the next one as soon as a machine is free

- a: machine 1
- b: machine 2
- c: machine 3
- d: machine 4

Good or bad?

**LS achieves a perf. ratio 2-1/m.**

So all machines are busy from time 0 through $A(I)-t_k$

Consequently,

$$T_t z m A(I)-t_j \rightarrow T_t z m A(I)-m t_k$$

$$T_z m A(I) \rightarrow T_t z m A(I)$$

$$A(I) \leq T_t z m A(I)-t_k$$

$S$ $A$ job f finishes last, at time A

**LS achieves a perf. ratio 2-1/m.**

- **Proof:**
  - Let $T = \sum_t, i=1,2...n$, the sum of all processing times to be accommodated.
  - We know that the total processing time available in an optimal schedule on the machines is $m.OPT(I)$. So, $OPT(I) \leq T/m$.
  - Moreover, $OPT(I) \geq T_k$ for every k.

- Let $A(I)$ be the makespan of the schedule produced by LS.
  - By definition there must be a job $k$, with processing time $t_k$ that ends at the makespan time.
  - No machine can end its operation before $A(I)-t_k$, because then job $k$ would have been scheduled on that machine, thus reducing the makespan.

**Example: Worst Case**

- m x m
- m x 1
- makespan: 2m
- m x 1
- makespan: m+1
List with LPT

- List scheduling can do badly if long jobs at the end of the list spoil an even division of processing times.
- We now assume that the jobs are all given ahead of time, i.e. the LPT rule works only in the off-line situation. Consider the “Largest Processing Time first” or LPT rule that works as follows.

LPT(I)
1 sort the jobs in order of decreasing processing times: \( t_1 \geq t_2 \geq \ldots \geq t_n \)
2 execute list scheduling on the sorted list
3 return the schedule so obtained.

- The LPT rule achieves a performance ratio 4/3-1/(3m). **Prove out of Syllabus**