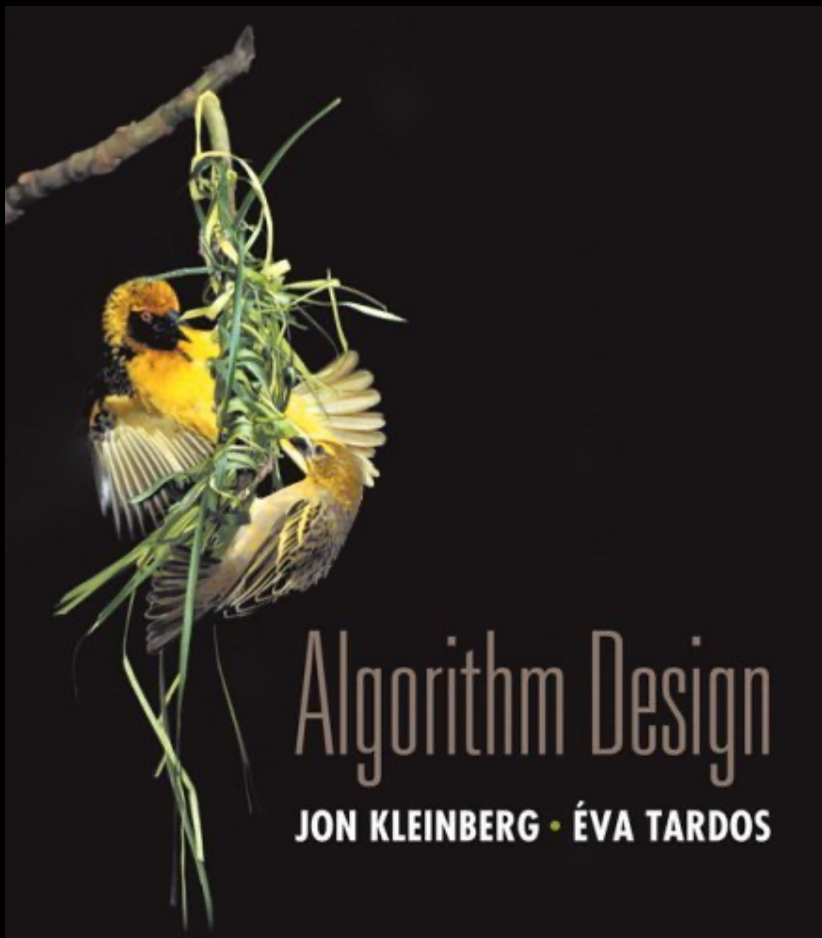


Chapter 11

Approximation Algorithms

Slides by Kevin Wayne.

Approximation Algorithms

Q. Suppose I need to solve an **NP-hard** problem. What should I do?
A. Theory says you're <u>unlikely</u> to find a **poly-time** algorithm.

Must sacrifice one of **three** desired features.
- Solve problem to optimality. **
- Solve problem in poly-time.
- Solve arbitrary instances of the problem.

** → **r-approximation algorithms**.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio **r** of true optimum.

Analysis Challenge. Need to prove a solution's value is **close** to optimum, without even knowing what optimum value is!

11.1 Load Balancing

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

- Job j must run contiguously on **one** machine.
- A machine can process **at most one job** at a time.

Def (Feasible Solutions). Let $J(i)$ be the subset of jobs assigned to machine i. The **load** of machine i is $L_i = \Sigma_{j \in J(i)} t_j$

Def(Cost of a Solution). The **makespan** is the **maximum** load on any machine $L = \max \{ L_i : i = 1, \ldots, m \}$

The Load Balancing Problem (It is NP-HARD):

Assign each job j to a machine i to **minimize** **makespan**.

Homework: Give a formal definition of a generic feasible solution.

Load Balancing: Greedy Approach

List-Scheduling (LS) algorithm.

- Consider n jobs in **any** fixed order.
- Assign job j to a machine whose load is the **smallest** one so far.

```
List-Scheduling(m, n, t_1,t_2,…,t_n) {
    for i = 1 to m {
        L_i ← 0          ← load on machine i
        J(i) ← φ         ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k L_k          ← machine i has smallest load
        J(i) ← J(i) ∪ {j}         ← assign job j to machine i
        L_i ← L_i + t_j           ← update load of machine i
    }
    return J(1), …, J(m)
}
```

Implementation. **O(n log m)** using a **priority queue**.

Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] LS algorithm is **2-approximation.**

- First **worst-case analysis** of an approximation algorithm.
- Need to compare LS solution with **optimal** makespan L^*.

Lemma 1. The optimal **makespan** $L^* \geq \max_j t_j$.

Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal **makespan** $L^* \geq (1/m) \Sigma_j t_j$

Pf.

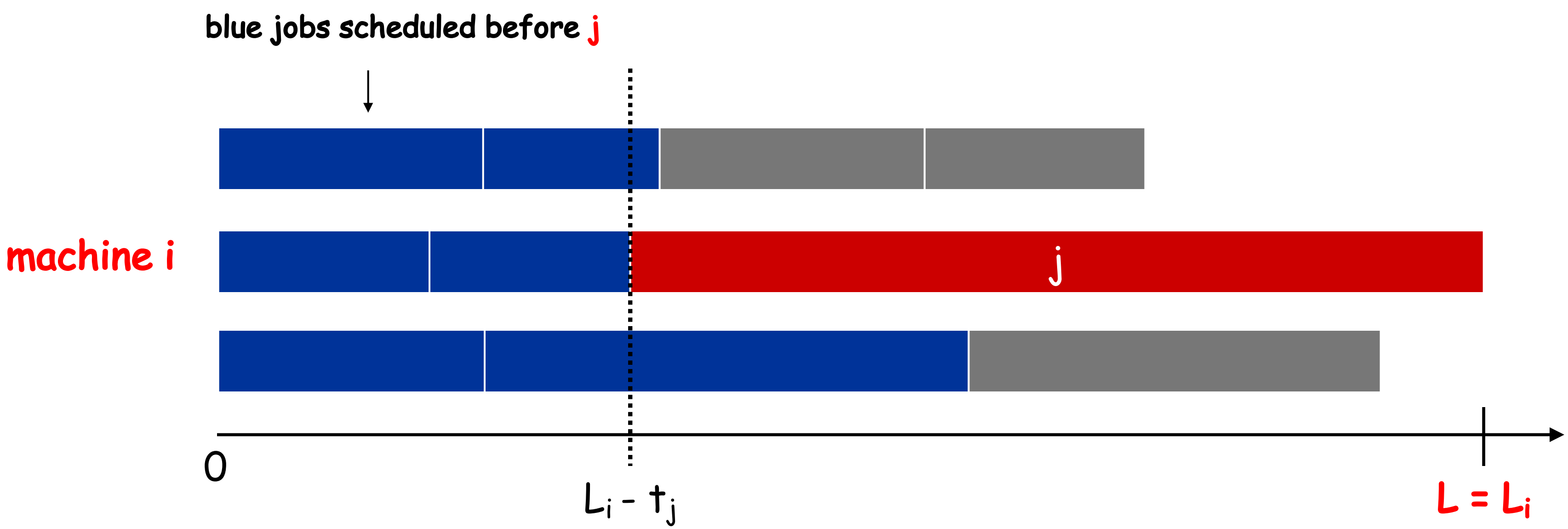
- The total processing time (i.e. total **work**) is $\Sigma_j t_j$.
- One of the m machines must do at least a $1/m$ fraction of total **work**. ▪

Load Balancing: List Scheduling Analysis

Theorem. LS algorithm is **2-approximation.**

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had **smallest load**. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ **for all** $1 \leq k \leq m$.



Load Balancing: List Scheduling Analysis

Theorem. LS algorithm is **2-approximation**.

Pf. Consider load L_i of bottleneck machine **i**.

- Let **j** be last job scheduled on machine i.
- When job **j** assigned to **machine i,** i had **smallest** load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \le L_k$ for all $1 \le k \le m$.
- Sum inequalities over all **k** and **divide** by **m**:
- $m(L_i - t_j) \le \Sigma L_k$

$$\begin{aligned} L_i - t_j &\le \tfrac{1}{m}\textstyle\sum_k L_k \\ &= \tfrac{1}{m}\textstyle\sum_k t_k \\ \text{Lemma 1} \longrightarrow \quad &\le L^* \end{aligned}$$

- **Now** $L_i = \underbrace{(L_i - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*.$ ▪

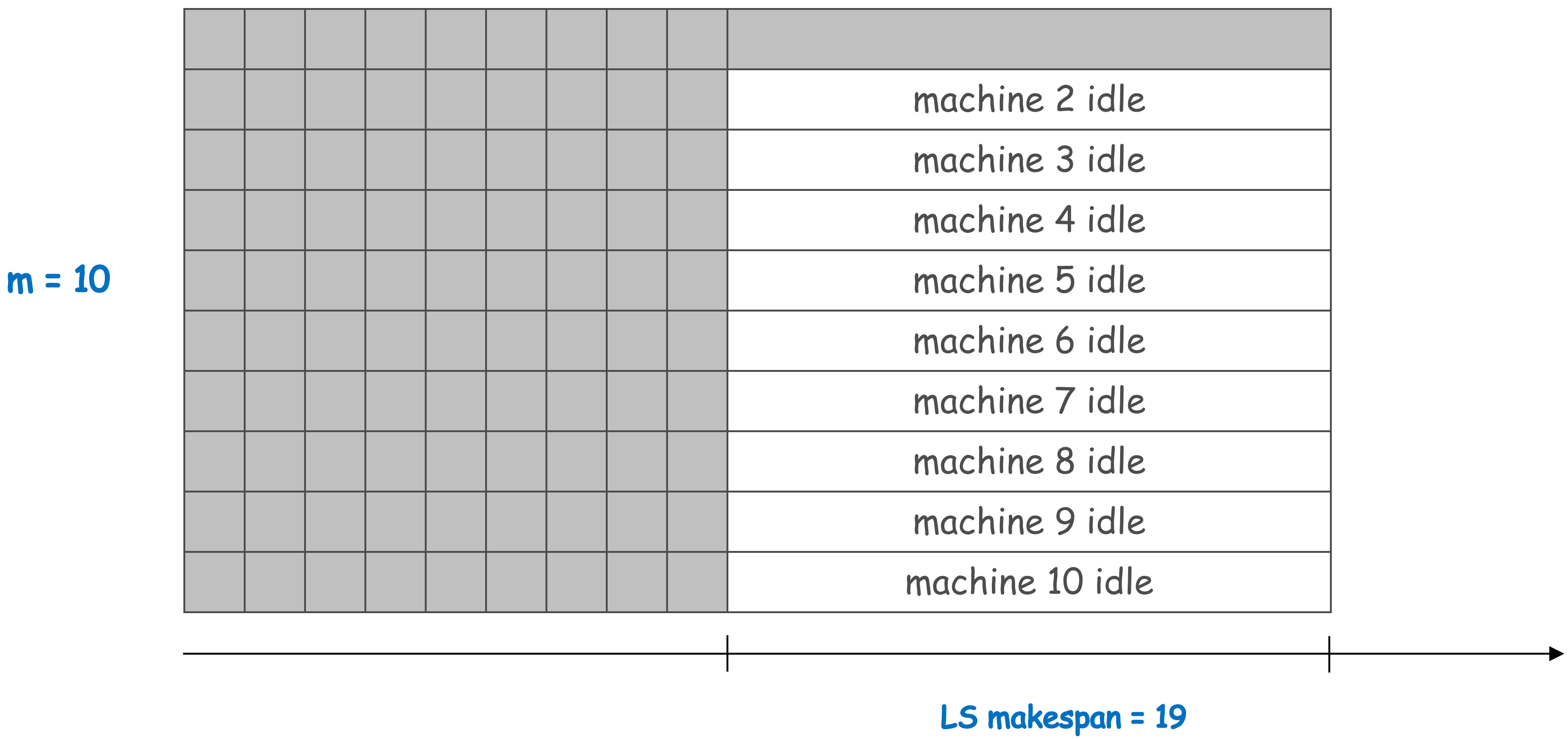
Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis **tight**?
A. Essentially yes.

Ex: m machines, $n = m(m-1)$ jobs: length **1** jobs, one job of length m

- LS SOLUTION:



Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

OPTIMAL SOLUTION: