

Algorithm Design and Analysis

CSE
565

LECTURES 39, 40
Approximation Algorithms
• Load Balancing

Sofya Raskhodnikova

11/28/2007

S. Raskhodnikova; based on slides by K. Wayne

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?
A. Theory says you're unlikely 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.

ρ -approximation algorithm.

- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

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

Load Balancing Problem

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

Def. The **makespan** is the maximum load on any machine $L = \max_i L_i$.

Goal. Assign each job to a machine to minimize makespan.

Load Balancing: List Scheduling

List-scheduling algorithm.

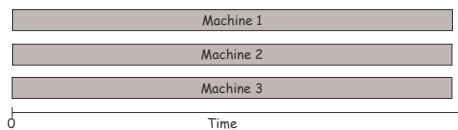
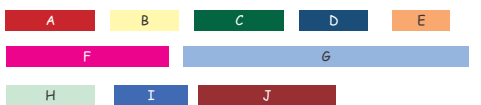
- Consider n jobs in some fixed order.
- Assign job j to machine whose load is smallest so far.

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

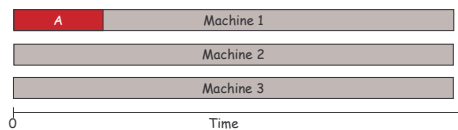
  for  $j = 1$  to  $n$  {
     $i = \operatorname{argmin}_k L_k$      ← machine  $i$  has smallest load
     $J(i) \leftarrow J(i) \cup \{j\}$  ← assign job  $j$  to machine  $i$ 
     $L_i \leftarrow L_i + t_j$    ← update load of machine  $i$ 
  }
  return  $J$ 
}
```

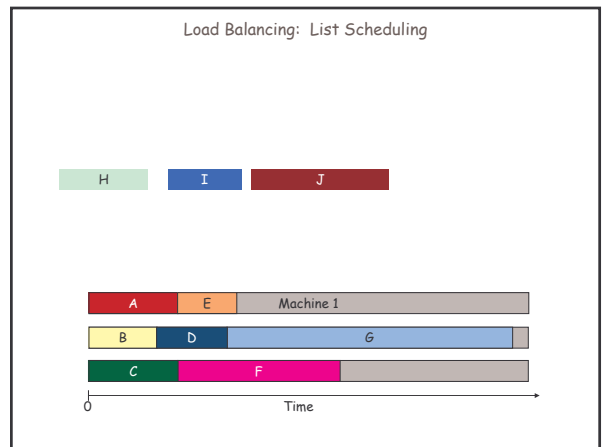
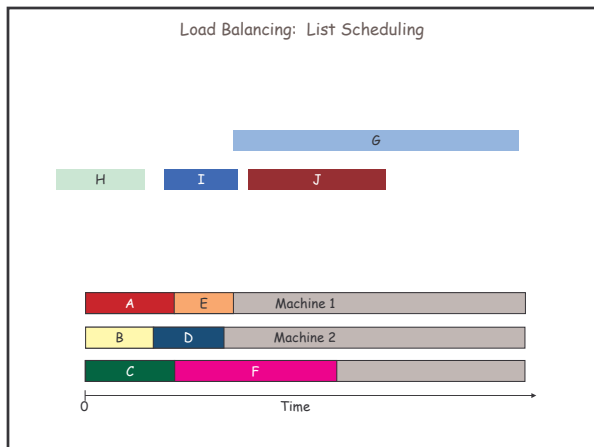
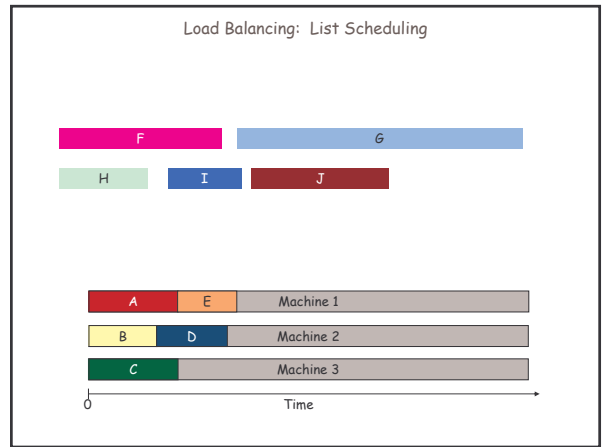
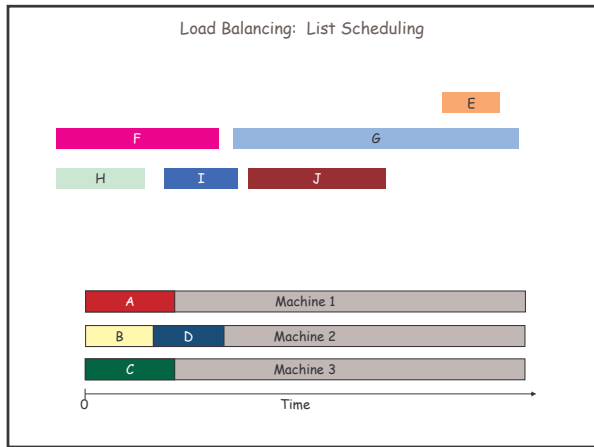
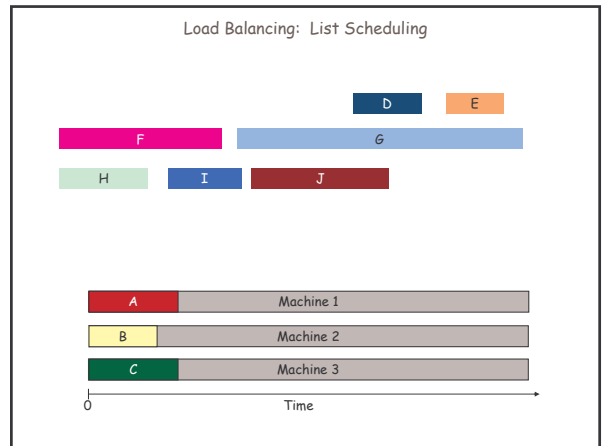
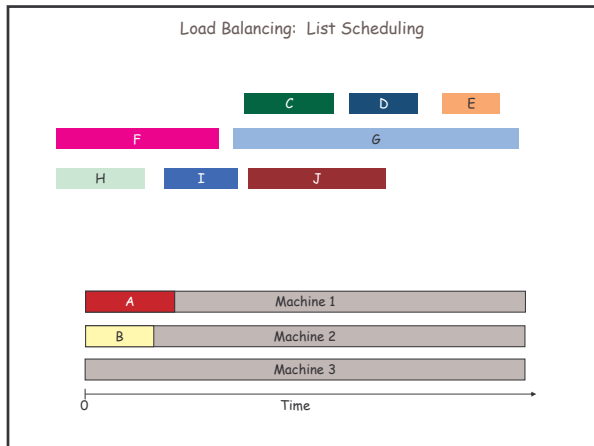
Implementation. $O(n \log n)$ time using a priority queue.

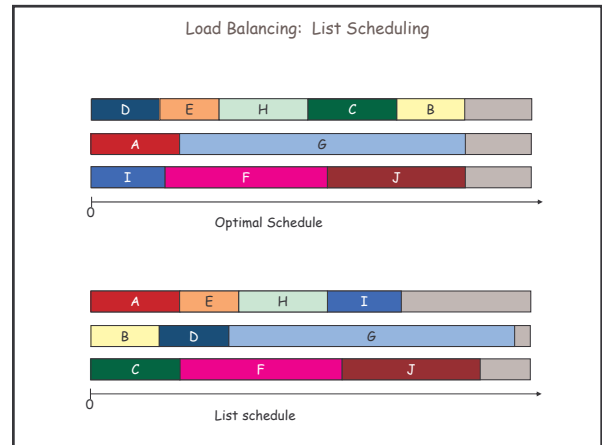
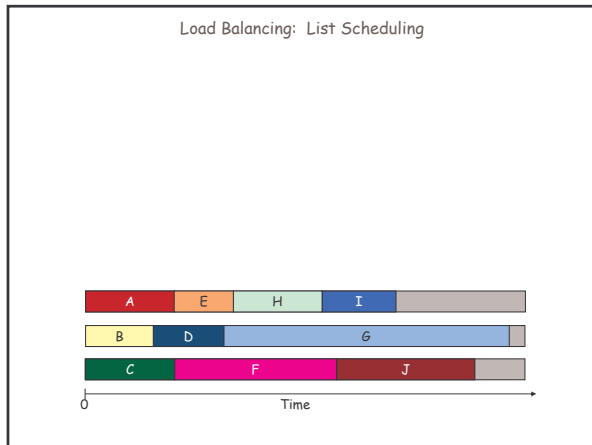
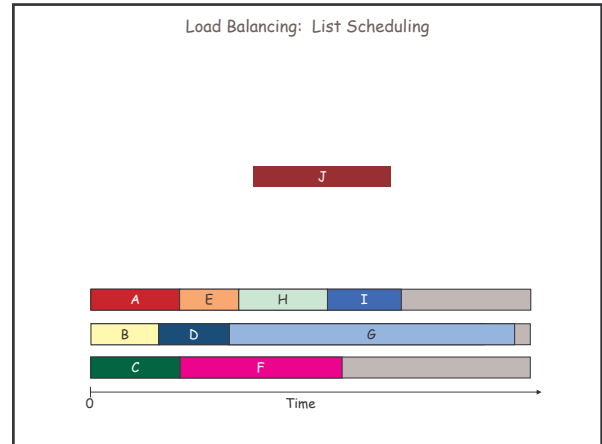
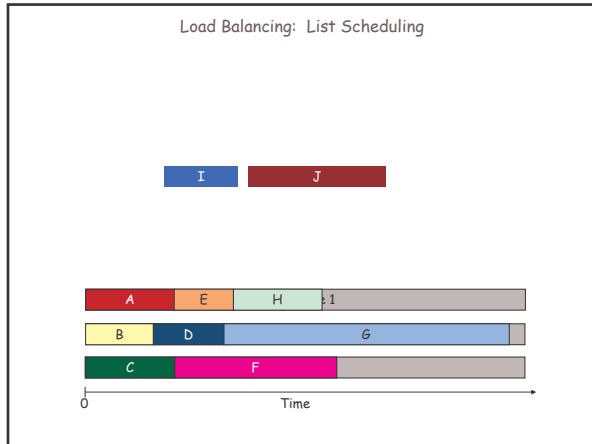
Load Balancing: List Scheduling



Load Balancing: List Scheduling







Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] Greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare resulting 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 $L^* \geq \frac{1}{m} \sum_{j=1}^n t_j$.

Pf.

- The total processing time is $\sum_j t_j$.
- One of m machines must do at least a $1/m$ fraction of total work. •

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 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. Greedy algorithm is a 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 $1 \leq k \leq m$.
- Sum inequalities over all k and divide by m :

$$\begin{aligned} L_i - t_j &\leq \frac{1}{m} \sum_{k=1}^m L_k \\ &= \frac{1}{m} \sum_{j=1}^n t_j \end{aligned}$$

$$\text{Lemma 1} \rightarrow \leq L^*$$

$$\bullet \text{ Now } L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq L^*} \leq 2L^* \quad \bullet$$

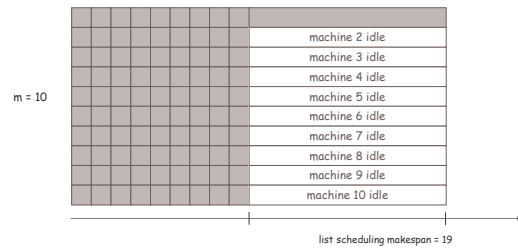
Lemma 2

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

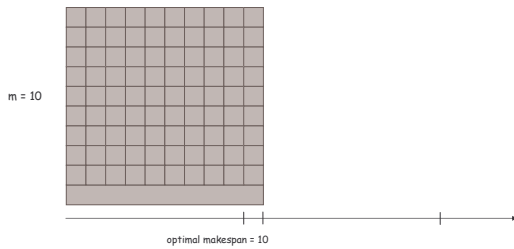


Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

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



Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in descending order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling( $m, n, t_1, t_2, \dots, t_n$ ) {
  Sort jobs so that  $t_1 \geq t_2 \geq \dots \geq t_n$ 

  for  $i = 1$  to  $m$  {
     $L_i \leftarrow 0$       ← load on machine  $i$ 
     $J(i) \leftarrow \emptyset$  ← jobs assigned to machine  $i$ 
  }

  for  $j = 1$  to  $n$  {
     $i = \text{argmin}_k L_k$       ← machine  $i$  has smallest load
     $J(i) \leftarrow J(i) \cup \{j\}$  ← assign job  $j$  to machine  $i$ 
     $L_i \leftarrow L_i + t_j$  ← update load of machine  $i$ 
  }
  return  $J$ 
}
```

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine. •

Lemma 3. If there are more than m jobs, $L^* \geq 2t_{m+1}$.

Pf.

- Consider first $m+1$ jobs t_1, \dots, t_{m+1} .
- Since the t_i 's are in descending order, each takes at least t_{m+1} time.
- There are $m+1$ jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. •

Theorem. LPT rule is a 3/2 approximation algorithm.

Pf. Same basic approach as for list scheduling.

$$L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{2}{3}L^*} \leq \frac{3}{2}L^* \quad \bullet$$

Lemma 3
(by observation, can assume number of jobs $> m$)

Load Balancing: LPT Rule

Q. Is our 3/2 analysis tight?

A. No.

Theorem. [Graham, 1969] LPT rule is a 4/3-approximation.

Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's 4/3 analysis tight?

A. Essentially yes.

Ex: m machines, $n = 2m+1$ jobs, 2 jobs of each length $m+1, m+2, \dots, 2m-1$ and one job of length m .

Load Balancing: State of the Art

Load Balancing is NP-complete even for 2 machines.

Exercise. Prove this statement. Hint: to prove NP-hardness, reduce from the problem in Exercise 26 in Chapter 8.

Polynomial Time Approximation Scheme (PTAS). $(1 + \epsilon)$ -approximation algorithm for any constant $\epsilon > 0$.

- Load balancing. [Hochbaum-Shmoys 1987]

Consequence. PTAS produces arbitrarily high quality solution, but trades off accuracy for time.