

## Lecture 13, Dec 5

### Load Balancing Problem

- Given  $m$  machines and  $n$  jobs.
- Each job  $j$  has a processing time of  $t_j$ .
- Want to assign the jobs to the machines so that all the machines finish as soon as possible.
- Intuitively, this is achieved by “balancing” the work between the machines.

## Parameters

- The machines  $M_1, M_2, \dots, M_m$ .
- The set of jobs assigned to  $M_i$  is  $A(i)$ .
- The  $T_i$  time  $M_i$  needs:  
this is the load on  $M_i$ . 
$$T_i = \sum_{j \in A(i)} t_j,$$
- Want to minimize  
makespan: 
$$T = \max_i T_i.$$
- What if  $m > n$ ?

## Simple greedy algorithm

Greedy-Balance

Start with no jobs assigned

Set  $T_i = 0$  and  $A(i) = \emptyset$  for all machines  $M_i$

For  $j = 1, \dots, n$

    Let  $M_i$  be a machine that achieves the minimum  $\min_k T_k$

    Assign job  $j$  to machine  $M_i$

    Set  $A(i) \leftarrow A(i) \cup \{j\}$

    Set  $T_i \leftarrow T_i + t_j$

EndFor

## Simple greedy algorithm – cont'd

- Why doesn't it always work?
- OK, it doesn't give the optimal solution, but maybe the solution it outputs is not much worse than the optimal one.
- Denote the output of the algorithm by  $T$  and the optimal solution by  $T^*$ .

## Simple greedy algorithm – cont'd

- OK, it doesn't give the optimal solution, but maybe the solution it outputs is not much worse than the optimal one.
- To find out, we try to bound  $T^*$  from below.
- If  $T^*$  is not too small, then  $T$  is not much worse than  $T^*$ .

## Simple greedy algorithm – cont'd

- Lower bound I: even if the balancing is perfect,  $T^*$  must be at least the average load on the machines.

(11.1) *The optimal makespan is at least*

$$T^* \geq \frac{1}{m} \sum_j t_j.$$

## Simple greedy algorithm – cont'd

- Lower bound II: the longest job must be scheduled on some machine.

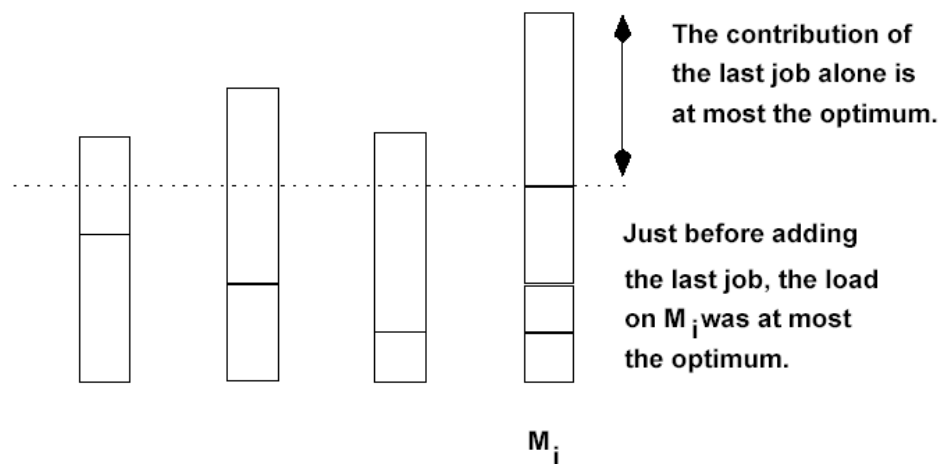
(11.2) *The optimal makespan is at least  $T^* \geq \max_j t_j$ .*

## Simple greedy algorithm – cont'd

- Now we can prove:

(11.3) *Algorithm Greedy-Balance produces an assignment of jobs to machines with makespan  $T \leq 2T^*$ .*

### Proof idea:

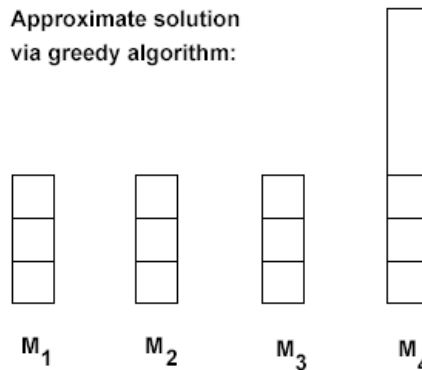


## Bad Examples

- The prove gives us a hint on how to construct “bad” examples.
- The last job should be very heavy, the others balanced.

## Bad Examples

- $m(m-1)$  jobs of length 1, one job of length  $m$ .
- $T = (m-1) + m = 2m - 1$ .



## Bad Examples

- $m(m-1)$  jobs of length 1, one job of length  $m$ .

- $T^*=m$ .

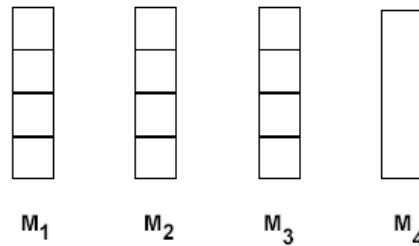
- Approx ratio:

$$(2m-1)/m =$$

$$2 - 1/m.$$

Actually, precise.

Optimal solution:



## Improving the approximation

- Idea: the “bad” examples had a huge job arriving at the end.
- Sort the jobs to settle the heavy jobs first.

## Sorted-Balance

Sorted-Balance

Start with no jobs assigned,

Set  $T_i = 0$  and  $A(i) = \emptyset$  for all machines  $M_i$

Sort jobs in decreasing order of processing times  $t_j$ .

Assume that  $t_1 \geq t_2 \geq \dots \geq t_n$

For  $j = 1, \dots, n$

    Let  $M_i$  be the machine that achieves the minimum  $\min_k T_k$

    Assign job  $j$  to machine  $M_i$

    Set  $A(i) \leftarrow A(i) \cup \{j\}$

    Set  $T_i \leftarrow T_i + t_j$

EndFor

## Improving the approximation

- Can strengthen

(11.2) *The optimal makespan is at least  $T^* \geq \max_j t_j$ .*

to

(11.4) *If there are more than  $m$  jobs, then  $T^* \geq 2t_{m+1}$ .*

because there will be a machine with at least  
jobs scheduled

## Improving the approximation

- We get

(11.5) *Algorithm Sorted-Balance produces an assignment of jobs to machines with makespan  $T \leq \frac{3}{2}T^*$ .*

Bad examples?

## Ex 27, Ch 7 in the book

Some of your friends with jobs out West decide they really need some extra time each day to sit in front of their laptops, and the morning commute from Woodside to Palo Alto seems like the only option. So they decide to carpool to work.

Unfortunately, they all hate to drive, so they want to make sure that any carpool arrangement they agree upon is fair, and doesn't overload any individual with too much driving. Some sort of simple round-robin scheme is out, because none of them goes to work every day, and so the subset of them in the car varies from day to day.

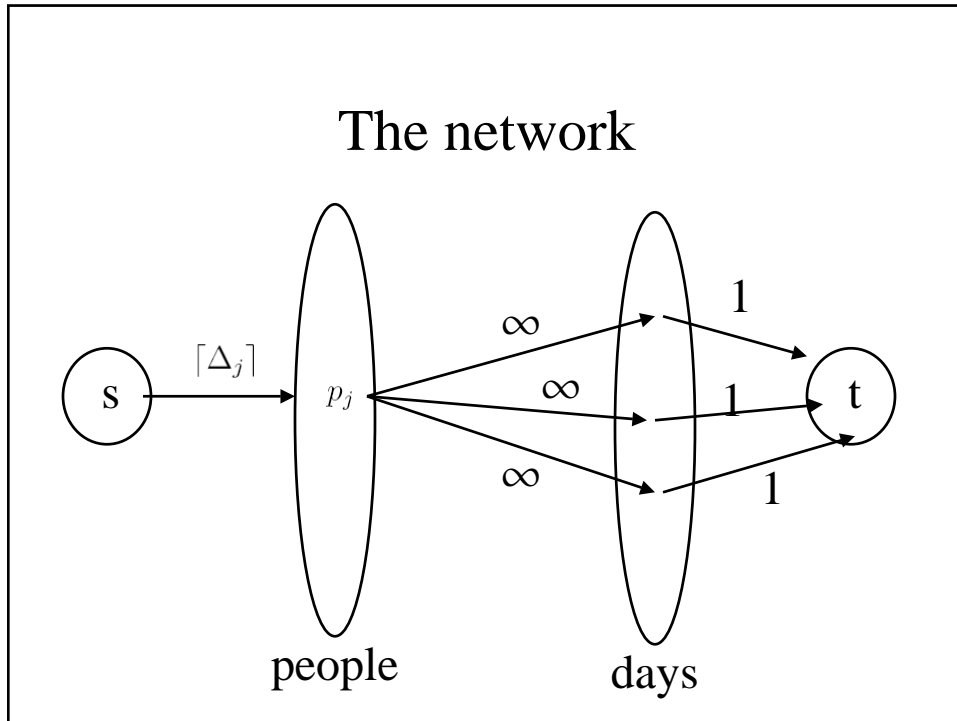
Here's one way to define *fairness*. Let the people be labeled  $S = \{p_1, \dots, p_k\}$ . We say that the *total driving obligation* of  $p_j$  over a set of days is the expected number of times that  $p_j$  would have driven, had a driver been chosen uniformly at random from among the people going to work each day. More concretely, suppose the carpool plan lasts for  $d$  days, and on the  $i^{\text{th}}$  day a subset  $S_i \subseteq S$  of the people go to work. Then the above definition of the total driving obligation  $\Delta_j$  for  $p_j$  can be written as  $\Delta_j = \sum_{i:p_j \in S_i} \frac{1}{|S_i|}$ . Ideally, we'd like to require that  $p_j$  drives at most  $\Delta_j$  times; unfortunately,  $\Delta_j$  may not be an integer.

So let's say that a *driving schedule* is a choice of a driver for each day — i.e. a sequence  $p_{i_1}, p_{i_2}, \dots, p_{i_d}$  with  $p_{i_t} \in S_t$  — and that a *fair driving schedule* is one in which each  $p_j$  is chosen as the driver on at most  $\lceil \Delta_j \rceil$  days.

- (a) Prove that for any sequence of sets  $S_1, \dots, S_d$ , there exists a fair driving schedule.
- (b) Give an algorithm to compute a fair driving schedule with running time polynomial in  $k$  and  $d$ .

## Sol'n outline

- We first think about how we would solve (b) if we knew that such a schedule exists.
- Create a network with drivers on one side and days on the other.
- Need to match drivers to days.
- Connect the source  $s$  to a driver  $p_j$  with weight  $\lceil \Delta_j \rceil$
- Connect each day  $d_i$  to the sink  $t$  with weight 1.
- Connect a driver  $p_j$  to days on which he goes to work with infinite capacity.
- Find a max-flow in the network.

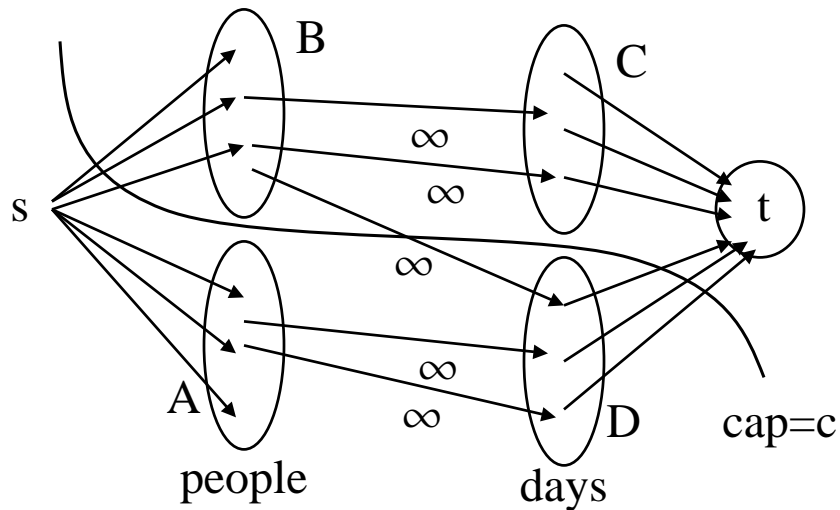


- An integer flow of value  $|f|$  in the network corresponds to a valid driving schedule where there is somebody driving on  $|f|$  out of  $d$  days.
- A valid driving schedule where every day someone is driving corresponds to a flow of value  $d$ .
- Thus assuming there is a valid driving schedule the max-flow in the network will correspond to it.
- To show that it exists (part (a)) we need to show that the value of the max-flow in the network is exactly  $d$  (not less).

## Proving (a)

- To show that the max-flow in the network is at least  $d$  we use the max-flow min-cut theorem.
- Suppose that the capacity of the network is  $c < d$ . Then there is a cut of capacity  $c$  in the network.

## The cut



- $c$  is finite, so there are no edges crossing from  $A$  to  $C$ . This means that only people from  $B$  go to work on the days from  $C$ :  $S_i \subset B$ .
- Hence the contribution of  $s \rightarrow B$  to the cut is

$$\sum_{p_j \in B} \lceil \Delta_j \rceil \geq \sum_{p_j \in B} \Delta_j = \sum_{p_j \in B} \sum_{i: p_j \in S_i} \frac{1}{|S_i|} = \sum_i \sum_{p_j \in S_i \cap B} \frac{1}{|S_i|} \leq$$

$S_i \subset B$ , so  $S_i \cap B = S_i$

restrict to  $i$ 's in  $C$

$$\sum_{i \in C} \sum_{p_j \in S_i \cap B} \frac{1}{|S_i|} = \sum_{i \in C} \sum_{p_j \in S_i} \frac{1}{|S_i|} = \sum_{i \in C} 1 = |C|$$

## Conclusion

- So, the capacity of the cut  $c$  is
- $c = \text{contribution of } s \rightarrow B + \text{contribution of } D \rightarrow t \geq |C| + \text{contribution of } D \rightarrow t = |C| + |D| = d$ .
- Contradiction, which shows that the max-flow in the network has value  $d$ .