

# 15-251

Great Theoretical Ideas  
in Computer Science

## Approximation and Online Algorithms

Lecture 28 (November 30, 2010)

In the previous lecture,  
we saw two problem classes:  
P and NP

### The Class P

We say a set  $L \subseteq \Sigma^*$  is in P if there is  
a program A and  
a polynomial  $p(\cdot)$

such that for any  $x$  in  $\Sigma^*$ ,

A(x) runs for at most  $p(|x|)$  time  
and answers question "is  $x$  in L?" correctly.

### The Class P

The class of all sets L that can be  
recognized in polynomial time.

The class of all decision problems that  
can be decided in polynomial time.

### P

contains many useful problems:

- graph connectivity
- minimum spanning tree
- matchings in graphs
- shortest paths
- solving linear systems  $Ax = b$
- linear programming
- maximum flows

Many of this we will (re)visit in 15-451.

## NP

A set  $L \in \text{NP}$

if there exists an algorithm  $V$  and a polynomial  $p(\ )$  such that

*verifier*

For all  $x \in L$

there exists  $y$  with  $|y| \leq p(|x|)$

such that  $V(x,y) = \text{YES}$

in  $p(|x|)$  time

"exists a quickly-verifiable proof"

For all  $x' \notin L$

For all  $y'$  with  $|y'| \leq p(|x'|)$

such that  $V(x',y') = \text{NO}$

in  $p(|x|)$  time

"all non-proofs rejected"

## The Class NP

The class of sets  $L$  for which there exist "short" proofs of membership (of polynomial length) that can be "quickly" verified (in polynomial time).

Recall:  $V$  doesn't have to find these proofs  $y$ ; it just needs to be able to verify that  $y$  is a "correct" proof.

## $P \subseteq \text{NP}$

For any  $L$  in  $P$ , we take the "proof"  $y$  to be the empty string, and  $V(x,y) = A(x)$  to satisfy the requirements.

Hence, every language in  $P$  is also in  $\text{NP}$ .

in this case, "the computation is the proof"

## Summary: P versus NP

Set  $L$  is in  $P$  if membership in  $L$  can be decided in poly-time.

Set  $L$  is in  $\text{NP}$  if each  $x$  in  $L$  has a short "proof of membership" that can be verified in poly-time.

Fact:  $P \subseteq \text{NP}$

Million (Billion) \$ question: Does  $\text{NP} \subseteq P$  ?

## NP-hard and NP-complete

$L$  is NP-hard if a polynomial-time algorithm for  $L$  implies a polynomial-time algorithm for all of  $\text{NP}$

to show NP-hardness for  $L$ , find another NP-hard problem (say SAT), and reduce SAT to  $L$ .

$L$  is NP-complete if  $L$  is NP-hard and  $L$  itself belongs to  $\text{NP}$ .

## NP Contains Lots of Problems We Don't Know to be in P

Classroom Scheduling  
Packing objects into bins  
Scheduling jobs on machines  
Finding cheap tours visiting a subset of cities  
Allocating variables to registers  
Finding good packet routings in networks  
Decryption  
...

What do we do now?

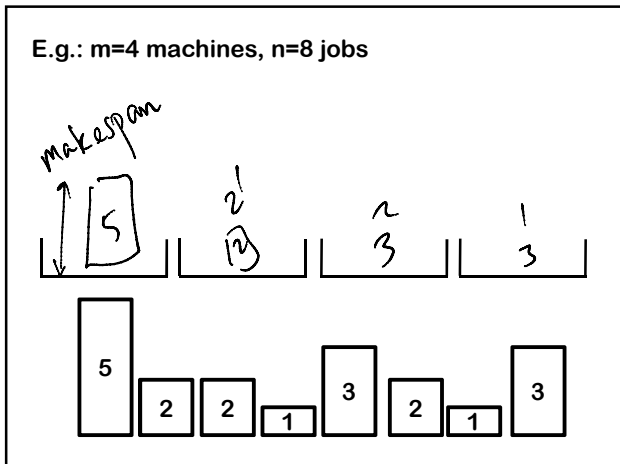
We'd really like to solve these problems.

But we don't know how to solve them in polynomial time...

A solution for some of these:  
Try to solve them "approximately"

E.g. Scheduling Jobs on Machines

Input:  
A set of  $n$  jobs, each job  $j$  has processing time  $p_j$   
A set of  $m$  identical machines



E.g. Scheduling Jobs on Machines

Input:  
A set of  $n$  jobs, each job  $j$  has processing time  $p_j$   
A set of  $m$  identical machines

Allocate these  $n$  jobs to these  $m$  machines to minimize the load of the worst-loaded machine.

(We call this objective function the "makespan")

We call this problem MinMakespan  
Is MinMakespan NP-hard?

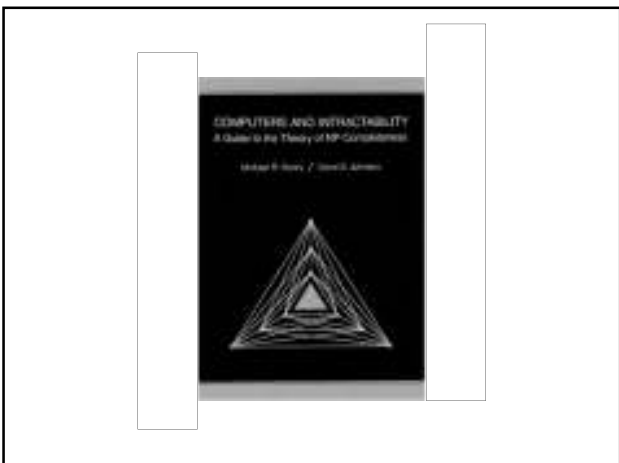
NP hardness proof

To prove NP hardness, find a problem such that

- that problem is itself NP hard
- if you can solve MinMakespan quickly, you can solve that problem quickly

*reduce that problem to MinMakespan*

Can you suggest such a problem?



**The (NP hard) Partition Problem**

Given a set  $A = \{a_1, a_2, \dots, a_n\}$  of  $n$  naturals which sum up to  $2B$  ( $B$  is a natural), does there exist a subset of these that sums to exactly  $B$ ?

Call this problem "Partition"

so we've found a problem Partition such that:

a) Partition is NP hard (we know - Johnson say so)

b) if you can solve MinMakespan quickly, you can solve Partition quickly

Take any instance  $(A = \{a_1, \dots, a_n\}, B)$  of Partition

Each natural number in  $A$  corresponds to a job.  
The processing time  $p_j = a_j$

We have  $m = 2$  machines.

Easy Theorem: there is a solution with makespan =  $B$  iff there is a partition of  $A$  into two equal parts.

$\Rightarrow$  if you solve Makespan fast, you solve Partition fast.

$\Rightarrow$  if Partition is hard, Makespan is hard.

**MinMakespan**

**Partition**

**NP-hard**

If MinMakespan in polynomial-time

then Partition in polynomial-time

we just took an instance of Partition and reduced it to MinMakespan

then MinMakespan is also hard

so, if Partition is hard

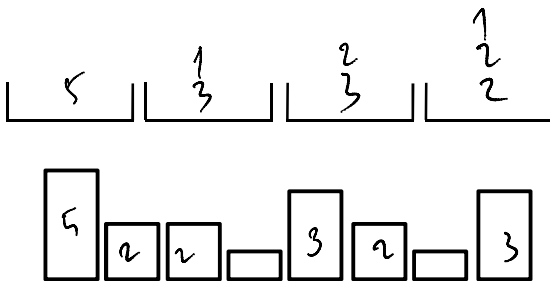
hence MinMakespan NP-hard

What do we do now???

Finding the best solution is hard

but can we find something that is not much worse than the best?

Can you suggest an algorithm?



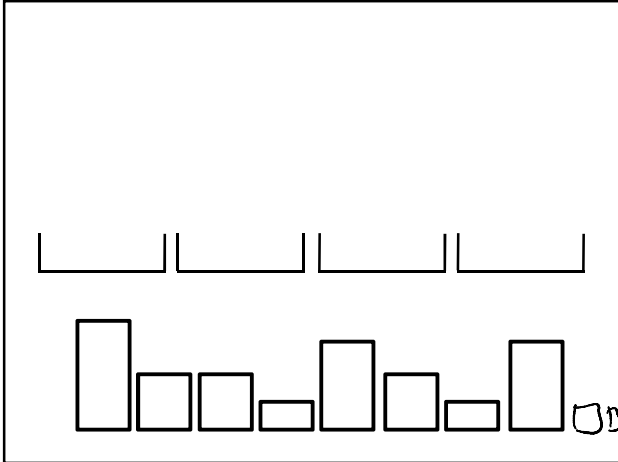
**Graham's Greedy Algorithm**

Order the jobs  $j_1, j_2, \dots, j_n$  in some order

Initially all the machines are empty

For  $t = 1$  to  $n$

Assign  $j_t$  to the least loaded machine so far



**Graham's Greedy Algorithm**

Order the jobs  $j_1, j_2, \dots, j_n$  in some order  
 Initially all the machines are empty  
 For  $t = 1$  to  $n$   
     Assign  $j_t$  to the least loaded machine so far

**Theorem: The  $\text{Makespan}_{\text{GGA}}$  of this algorithm is at most  $2 \times \text{OptimalMakespan}$**   
"OPT"

How do you argue what the optimal value is?

Suppose you could show that

1.  $\text{Makespan}_{\text{GGA}} \leq 2 \text{ Blah}$ , and
2.  $\text{Blah} \leq \text{OPT}$ ,

then you are done.

**Two candidates for "Blah"**

**Claim 1:  $p_{\max} \leq \text{OPT}$**   
**Proof: At least one machine gets the largest job.**

**Claim 2:  $(\sum_{t \leq n} p_t)/m \leq \text{OPT}$**   
**Proof: At least one machine must have at least the average load.**

OPT may be much larger than either of these two.

$p_{\max}$

$(\sum_{t \leq n} p_t)/m$

E.g.,  $n$  jobs of size 1  
 $\text{OPT} = n/m, p_{\max} = 1$

E.g., 1 job of size  $m$   
 $\text{OPT} = m, \text{average load} = 1$

Main insight: OPT cannot be simultaneously larger than both!

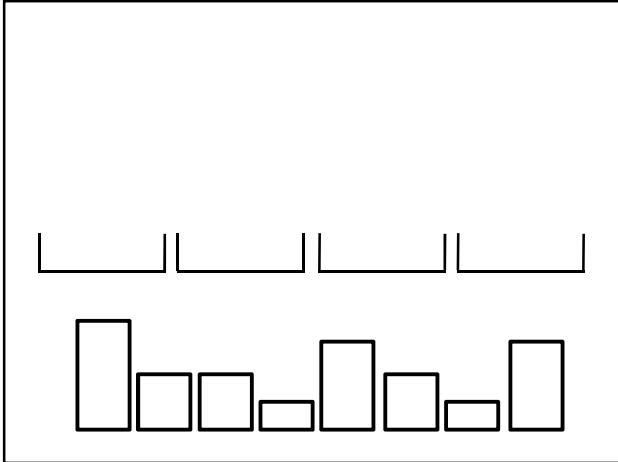
**Claim 3:  $\text{Makespan}_{\text{GGA}} \leq p_{\max} + (\sum_{t \leq n} p_t)/m$**

Look at the heaviest loaded machine.  
 Look at the last job  $j_k$  we scheduled on this machine.

The load of that machine (at that time) was at most the average  $(\sum_{t \leq k-1} p_t)/m \leq (\sum_{t \leq n} p_t)/m$

This last job contributes a load of  $p_k \leq p_{\max}$

Total load on this machine  $\leq p_{\max} + (\sum_{t \leq n} p_t)/m$



**To recap**

**Claim 1:**  $p_{\max} \leq \text{OPT}$

**Claim 2:**  $(\sum_{t \leq n} p_t)/m \leq \text{OPT}$

**Claim 3:**  $\text{Makespan}_{\text{GGA}} \leq p_{\max} + (\sum_{t \leq n} p_t)/m$   
 $\leq \text{OPT} + \text{OPT}$

**Theorem:** The  $\text{Makespan}_{\text{GGA}}$  of this algorithm is at most  $2 \times \text{OPT}$

**Two obvious questions**

Can we analyse this algorithm better?

Can we give a better algorithm?

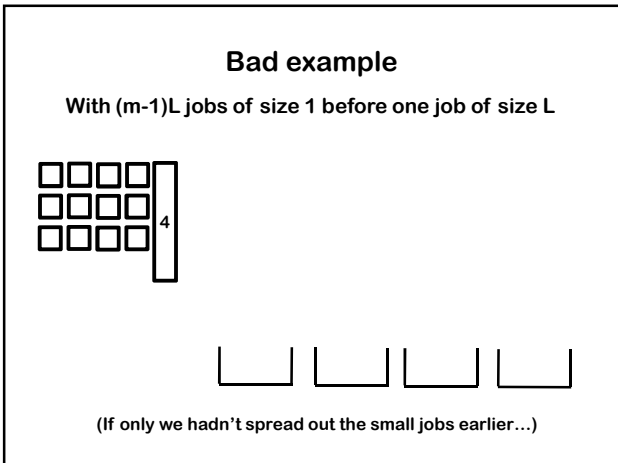
*m = # of machines*

**Is the Algorithm any better?**

Being slightly more careful a few slides back:

$\text{Makespan}_{\text{GGA}} \leq p_{\max}(1-1/m) + (\sum_{t \leq n} p_t)/m$   
 $\leq \text{OPT} (2-1/m)$

But we cannot do better with this algorithm



**Is the Algorithm any better?**

With  $(m-1)L$  jobs of size 1 before one job of size  $L$

$\text{Makespan}_{\text{GGA}} = L(m-1)/m + L = L(2 - 1/m)$   
 $\text{OPT} = L$

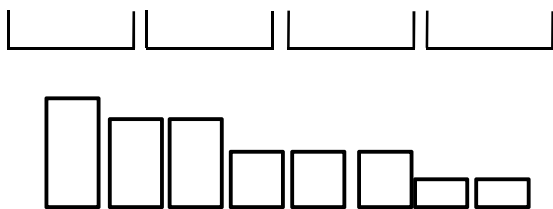
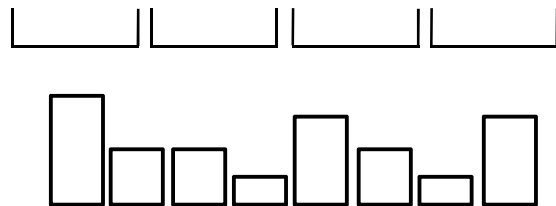
### Better Graham's Greedy Algorithm

Order the jobs  $j_1, j_2, \dots, j_n$  in ~~some order~~ <sup>decreasing order of size</sup>

Initially all the machines are empty

For  $t = 1$  to  $n$

Assign  $j_t$  to the least loaded machine so far



### Better Theorem: $\text{Makespan}_{\text{GBGA}} \leq 1.5 \text{ OPT}$

Suppose max-load machine has only one job  $j_t$

Then we are optimal. ( $p_t \leq p_{\text{max}} \leq \text{OPT}$ )

Else let  $j_t$  be last job GBGA scheduled on it.

Just before it, all machines have at least one job each.  
(since we allocated  $j_t$  to the least loaded machine)

So there are  $m+1$  jobs with length at least  $p_t$   
(by the non-increasing ordering of the jobs)

Hence  $2p_t \leq \text{OPT}$  (by pigeonhole)  
 $p_t \leq \text{OPT}/2$

$$\text{Makespan}_{\text{GBGA}} \leq p_t + (\sum_{k \leq n} p_k)/m$$

$p_t \leq \text{OPT}/2$                        $(\sum_{k \leq n} p_k)/m \leq \text{OPT}$

$$\Rightarrow \text{Makespan}_{\text{GBGA}} \leq 1.5 \text{ OPT}$$

In fact, it can be shown that  
 $\text{Makespan}_{\text{GBGA}} \leq (4/3 - 1/3m) \text{ OPT}$

The proof is not hard,  
but we'll not do it here.

You can show examples  
where GBGA is at least  $(4/3 - 1/3m) \text{ OPT}$

And there are algorithms which do better than  $4/3$   
but they are much more complicated...

This is the general template to safely handle problems you encounter that may be NP-hard

(as long as you cannot prove  $P = NP$ )

Let's skim over the steps with another problem as an example...

E.g. The Traveling Salesman Problem (TSP)

Input:

A set of  $n$  cities, with distance  $d(i,j)$  between each pair of cities  $i$  and  $j$ .

Assume that  $d(i,k) \leq d(i,j) + d(j,k)$  (triangle inequality)

Find the shortest tour that visits each city exactly once

NP hard!

Prove it is NP-hard

To prove NP hardness, find a problem such that

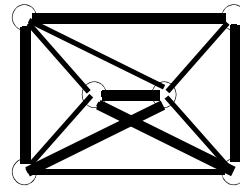
- a) that problem is itself NP hard
- b) if you can solve TSP quickly, you can solve that problem quickly

Can you suggest such a problem?

Hamilton Cycle

Hamilton Cycle (HAM)

Given a graph  $G = (V,E)$ , a cycle that visits all the nodes exactly once



- a) that problem is itself NP hard

- b) if you can solve TSP quickly, you can solve HAM quickly

Take any instance  $G$  of HAM with  $n$  nodes.

Each node is a city.

If  $(i,j)$  is an edge in  $G$ , set  $d(i,j) = 1$

If  $(i,j)$  is not an edge, set  $d(i,j) = 2$

Note:  $d(\dots)$  satisfies triangle inequality

Easy Theorem: any tour of length  $n$  is a Hamilton cycle in  $G$

$\Rightarrow$  if you solve TSP, you find a Hamilton cycle.

Since it is NP hard to solve exactly, try to find an approximately good solution

From Lecture 18:

Find a minimum spanning tree

Walk "around" the MST and take shortcuts if a node has already been visited

Shortcuts only decrease the cost, so

$\text{Cost}(\text{Greedy Tour}) \leq 2 \text{Cost}(\text{MST})$

$\leq 2 \text{Cost}(\text{Optimal Tour})$

this is a "2-approximation" to TSP



**And try to improve it...**

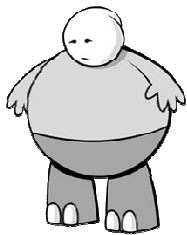
**[Christofides (CMU) 1976]: There is a simple 1.5-approximation algorithm for TSP.**

**Theorem: If you give a 1.001-approximation for TSP then  $P=NP$ .**

**What is the right answer?  
Still a major unsolved problem...**

**Many cool combinatorial problems that arise in analyzing algorithms.**

**The mathematical tools you've learnt in this course are extremely relevant for their analysis.**



**Here's What  
You Need to  
Know...**

**NP hardness**  
How to prove NP-hardness  
(via reductions from hard problems)

**Approximation Algorithms**  
If you can't solve it exactly  
try to solve it as best you can

**Scheduling Jobs on Parallel Machines**  
Graham's Greedy Algorithm  
Analysis