CS314 – Approximation algorithms

Note Title 11/6/2013

<u>Announcements</u>

- HW due Tuesday

Hard Problems

The world is full of them.

- some impossible
- some just slow

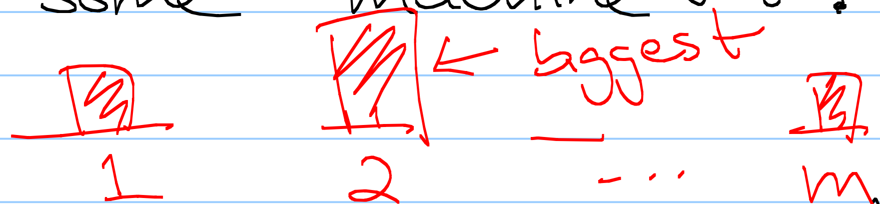
What to do?

Sometimes — faster solution.

Example: Load balancing

- n jobs, each with running time $T[1..n]$
- m machines to run them on

Goal: Compute an assignment $A[1..n]$ where each job j gets assigned to some machine i: $A[j] = i$



Makespan: max time any machine is busy

$$\text{makespan}(A) = \max_i \left(\sum_{j : A[j] = i} T[j] \right)$$

Note: Minimizing makespan is NP-Hard.

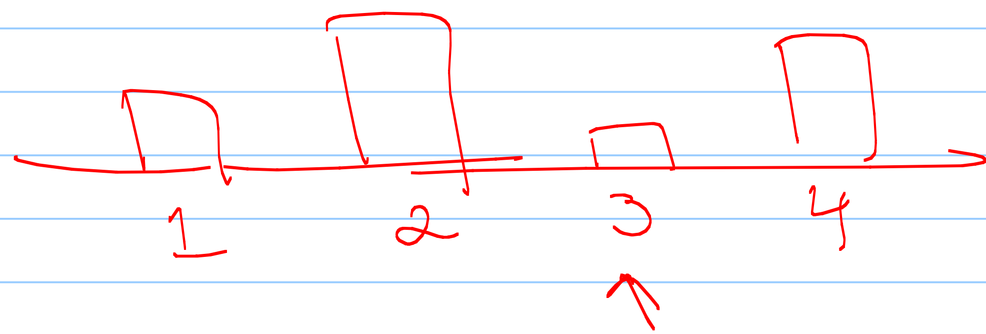
Reduction:

Reduce partition to makespan.

Approximating: think greedy

What seems like a decent strategy?

Given m spots, put job in current lowest.



Algorithm:

running times

#machines

$\textsc{GreedyLoadBalance}(T[1..n], m)$:

for $i \leftarrow 1$ to m
    $Total[i] \leftarrow 0$

for $j \leftarrow 1$ to n
    $mini \leftarrow \arg\min_i Total[i]$
    $A[j] \leftarrow mini$
    $Total[mini] \leftarrow Total[mini] + T[j]$

return $A[1..m]$

Claim: The makespan of this greedy algorithm is at most twice the optimal makespan.

pf: Start with 2 observations about OPT:

① For any job j, $T[j] \le OPT$.
because OPT must run j somewhere.

② $$\frac{\sum_j T[j]}{m} \le OPT$$

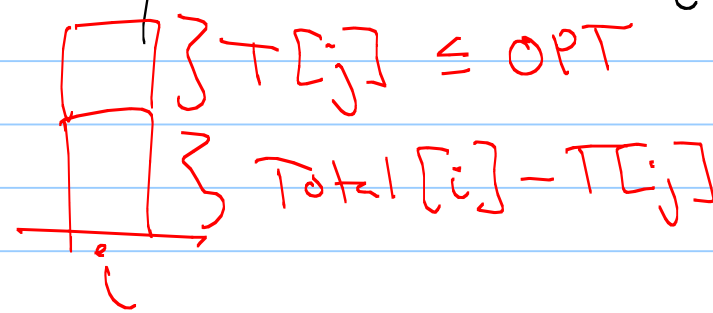
pf: cont.

Now consider machine with largest makespan in greedy $\to i$.

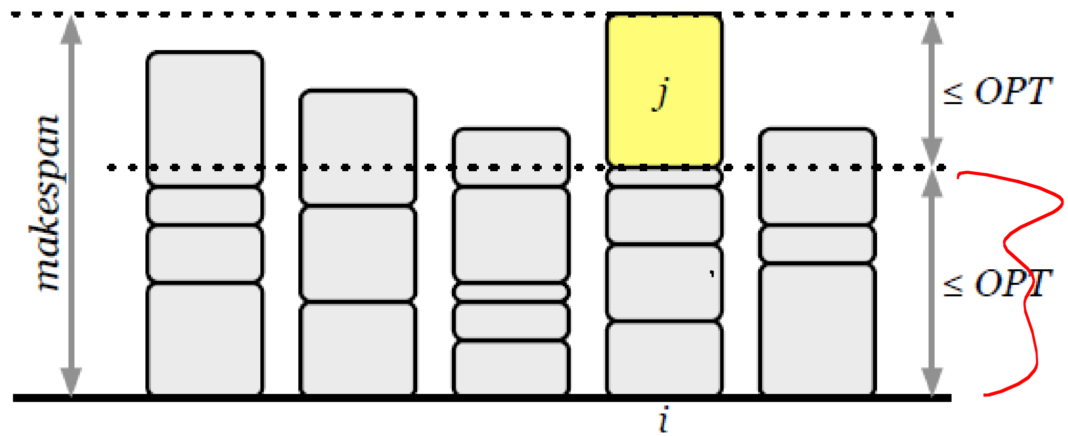
Let j be last job assigned.

Know $T[j] \le OPT$.

What can we say about $Total[i] - T[j]$?



Picture:



fact 2 $\Rightarrow \dfrac{\sum_j T[i]}{m} \leq OPT$

makespa $\leq 2 \cdot OPT$

Q:

Is this optimal?

(HW - no)

Note: This greedy algorithm is actually an online algorithm:

- doesn't need input ahead of time, but rather works when jobs are arriving one at a time!

Why useful?

processor allocation

Offline version: can improve!

SortedGreedyLoadBalance($T[1..n], m$):
  sort T in decreasing order
  return **GreedyLoadBalance**(T, m)

Claim: Makespan of above is $\leq \frac{3}{2} \cdot OPT$.

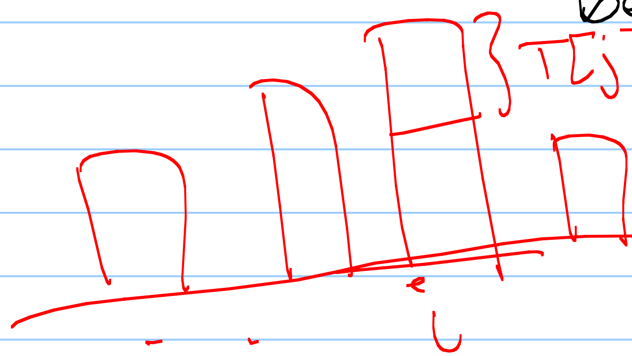
Claim: Makespan of above is $\le \frac{3}{2} \cdot OPT$.

pf:

Note:

- first m jobs go to different machines
- if $n \le m$, then $= OPT$ $\le \frac{3}{2} OPT$

Otherwise: Consider i & j as before:



$T[j]$ — last job assigned to biggest machine

Still have $\text{Total}[i] - T[j] \leq OPT$.
(by ②)

Now, in any schedule, some machine gets two of the first $m+1$ jobs:
(say $k \,\&\, \ell \leq m+1$)

$$\underbrace{T[k] + T[\ell] \leq OPT}$$

smaller

$1, 2, \ldots, k, \ldots, \ell, \ldots$

$$T[k] \leq \frac{OPT}{2} \text{ or } \underbrace{T[\ell] \leq \frac{OPT}{2}}$$

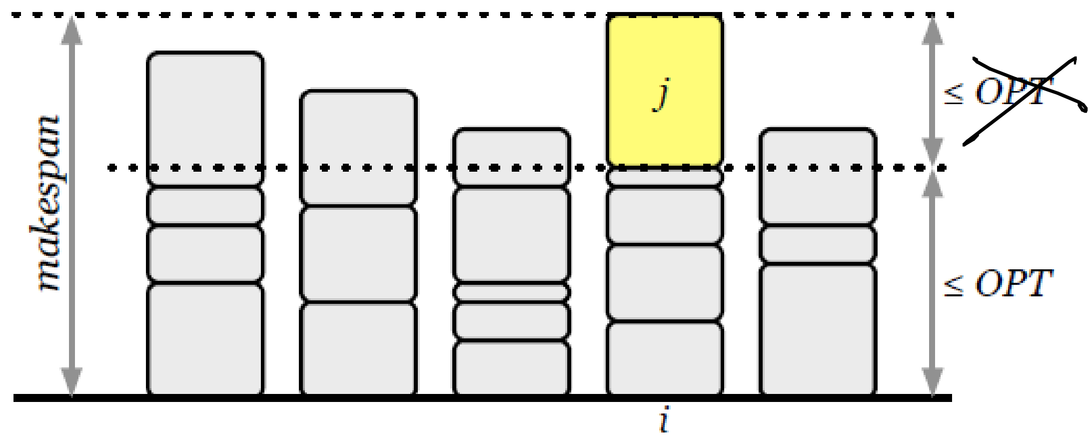
Jobs: $1, \ldots, k, \ldots, \ell, \ldots m+1, \ldots, j$

Now $m+1 \le j$ + jobs in decreasing order!

So $T[j] \le T[m+1] \le T[\max\{k,\ell\}]$

$\le \frac{1}{2} \cdot OPT$

So here:



$\frac{OPT}{2}$

Therefore, $Total[i] \leq \frac{3}{2} \cdot OPT$

Dfn: Approximation

- Let $OPT(x)$ = value of optimal solution
  $A(x)$ = value of solution computed by algorithm A.

A is an $\alpha(n)$-approximation algorithm if

$$\left.\begin{aligned} \frac{OPT(x)}{A(x)} &\le \alpha(n) \\ \text{and} \quad \frac{A(x)}{OPT(x)} &\le \alpha(n) \end{aligned}\right\}$$

So greedy load balancing (online):

$$\text{greedy}(x) \le 2\,\text{OPT}(x)$$

$$\frac{\text{greedy}(x)}{\text{OPT}(x)} \le 2 \qquad\qquad \frac{1}{2} \le \frac{\text{OPT}(x)}{\text{greedy}(x)}$$

greedy is a 2-approx

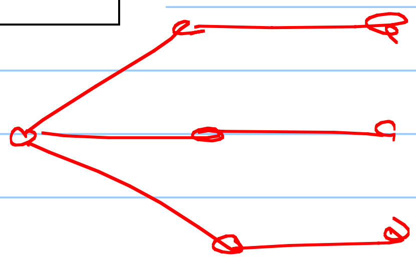
Vertex Cover

What was your greedy idea to find a vertex cover?

take max degree vertex, add it to cover

Algorithm:

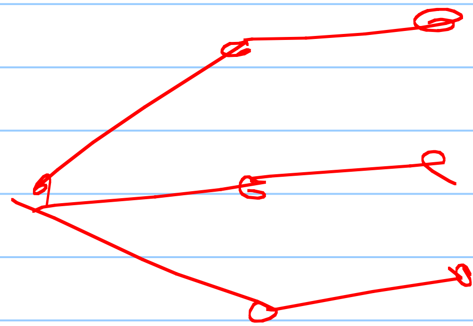
```
GREEDYVERTEXCOVER(G):
  C ← ∅
  while G has at least one edge
      v ← vertex in G with maximum degree
      G ← G \ v
      C ← C ∪ v
  return C
```



Optimal?

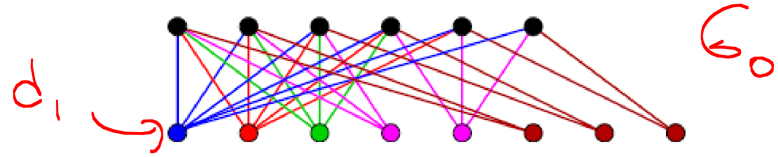
(find counter example)

Counterexample:



Q: Is it a 2-approx?

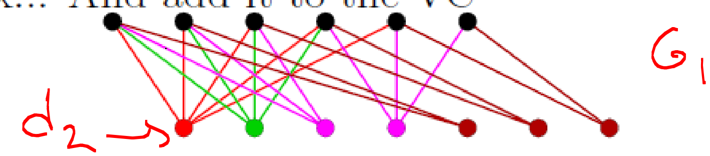
Ansser: No...



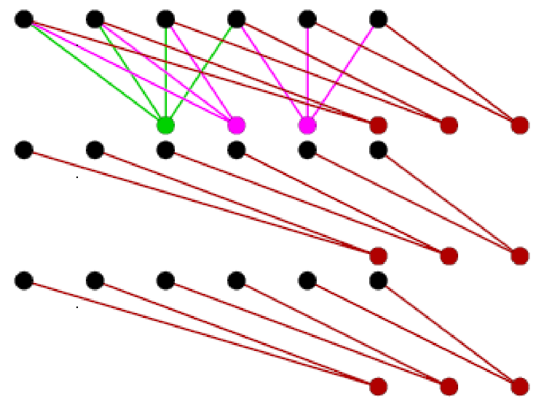
OPT: size 6

Greedy: 8

Remove the blue vertex... And add it to the VC



Remove red vertex



Greedy $\le O(\log n) \cdot OPT$

Thm: Greedy vertex cover is an $O(\log n)$-approximation.

pf: Let G_i = graph in i^{th} iteration

d_i = max degree in G_{i-1}

```
GreedyVertexCover(G):
  C ← ∅
  G_0 ← G
  i ← 0
  while G_i has at least one edge
      i ← i + 1
      v_i ← vertex in G_{i-1} with maximum degree
      d_i ← deg_{G_{i-1}}(v_i)
      G_i ← G_{i-1} \ v_i
      C ← C ∪ v_i
  return C
```

Also let: $|G_i|$ = # edges in G_i

C^* = OPT vertex cover

C^* is also vertex cover for G_{i-1}

$$\sum_{v \in C^*} \deg_{G_{i-1}}(v) \geq |G_i|$$

average degree in G_i of any $v \in C^*$ is $\geq \frac{|G_{i-1}|}{OPT}$

$G_0, G_1 \xrightarrow{\text{deleted}_2} G_2$

Also $|G_{i+1}| < |G_i|$

d_i = max degree in G_{i-1}, so

$$d_i \geq \frac{|G_{i-1}|}{OPT}$$

So:

$$\sum_{i=1}^{OPT} d_i \geq \sum_{i=1}^{OPT} \frac{|G_{i-1}|}{OPT} \geq \sum_{i=1}^{OPT} \frac{|G_{OPT}|}{OPT} = |G_{OPT}| = |G| - \sum_{i=1}^{OPT} d_i$$

$$\Rightarrow 2 \sum_{i=1}^{OPT} d_i \geq |G|$$

In other words, first OPT iterations of loop remove at least half edges in G.

$$\sum d_i \geq |G| - \sum d_i$$

$$2 \cdot \sum d_i \geq |G|$$

$$\Rightarrow \quad \sum_{i=1}^{OPT} d_i \geq \frac{|G|}{2}$$

So after $O(\log n)$ repetitions, all edges are gone.