CS300

Approximation

Announcements

- HW out, & oral grading next Friday

Hard Problems

Apparently, the world is full of them!

- some impossible

  ie

- others just slow

What to do?

- Approximate
- Randomization

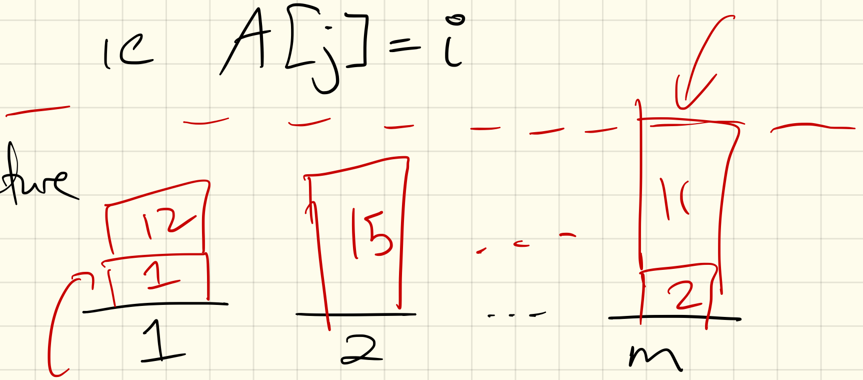
Example: Load Balancing

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

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

ie $A[j] = i$

Picture



$A[1]=1$
$A[12]=1$

__Natural Goal:__

Finish as early as possible!

Makespan: max time any machine is running jobs.

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

worst machine i

sum jobs on machine i

__Goal:__

Minimize makespan:

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

This is NP-Hard.

Why?

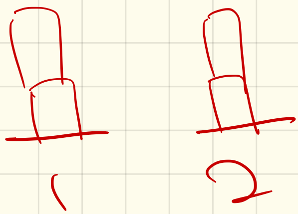
Reduce partition to this:

Given list $S = \{s_1, \dots, s_n\}$

$\hookrightarrow$ run n jobs with $T[j] = s_j$

+ set $m = 2$.

ask for makespan of value $\frac{\sum s_i}{2}$



Approximating

What seems a natural strategy?

Greed!

Possible heuristic:

A **heuristic technique** (/hjuːˈrɪstɪk/; Ancient Greek: εὑρίσκω, "find" or "discover"), often called simply a **heuristic**, is any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals. Where finding an optimal solution is impossible or impractical, heuristic methods can be used to speed up the process of finding a satisfactory solution. Heuristics can be mental shortcuts that ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, guesstimate, stereotyping, profiling, or common sense.

Consider jobs 1 at a time + assign to current "emptiest" machine.

Algorithm:

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 $A[1..m]$

initialize machines to 0

loop over jobs

find emptiest machine

assign j to emptiest machine

Runtime:

$$m + n(m+1)$$

$$= O(nm)$$

if you do Total[i]'s in a heap

$\hookrightarrow O(n \log m)$

"Correctness"

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

pf: Start w/ 2 observations:

① $$\underbrace{OPT}_{\text{optimal alg's makespan}} \geq \max_j (T[j])$$

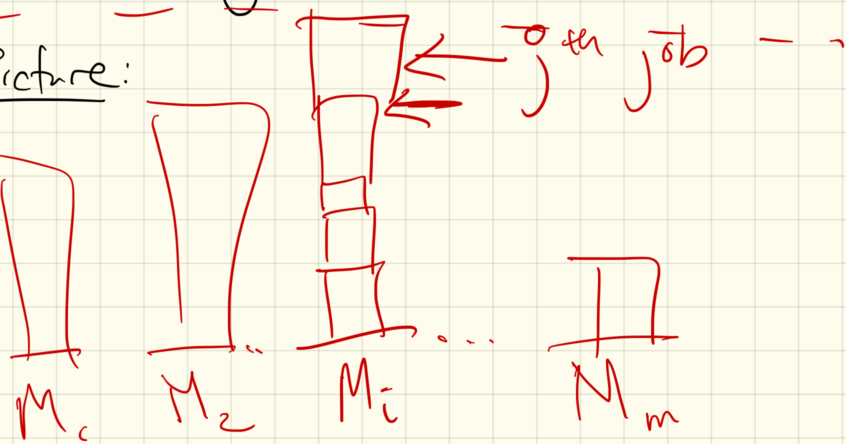
② $$OPT \geq \text{average job length} \quad \left(\frac{1}{m} \sum_{j=1}^{n} T[j] \right.$$

(pf cont)

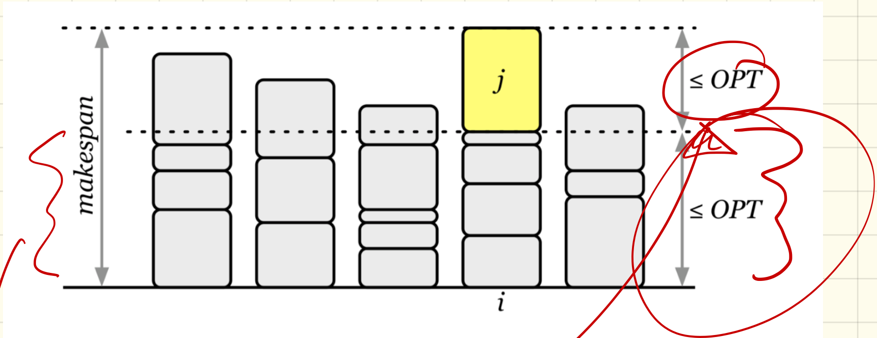
Now consider machine w/ largest makespan in greedy alg
↳ machine i.

Let j be last job assigned to machine i.

Picture:



Better picture:



Proof that GreedyLoadBalance is a 2-approximation algorithm

Obs 1 $\Rightarrow$

Goal:

Total[i] − T[j] ≤ OPT

↳ M_i's makespan w/ j removed

When j was assigned, M_i had lowest makespan

Total[i] − T[j] ≤ Total[k]

↳ had to be less than or equal to average!

by ②, OPT ≥ average

Q: Could this be optimal?

(Answer: **NO**!

Possibly on hw...)

Note: This is actually an *online* algorithm

↳ input is not specified ahead of time

Why might this be a useful observation?

Can we do better if given input offline?

Yes!

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

Runtime: $n(\log n + \cancel{\log m})$
m

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

pf:

2 cases:

$n \leq m$: (easy case)

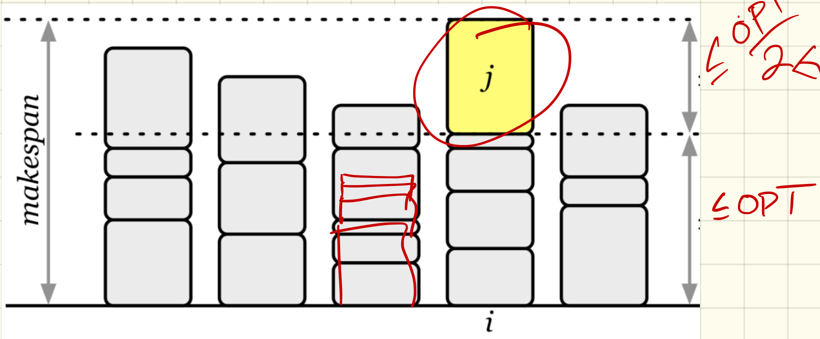
one per machine

$\Rightarrow$ greedy $= OPT$

$(\leq \frac{3}{2} OPT)$

Otherwise: $n > m$.

Consider i & j as before:



- Still have: $Total[i] - T[j] \le OPT$.

Now: in *any* schedule, some machine must have 2 of the first $m+1$ jobs.

$\rightarrow$ say $k + \ell \le m+1$

$T[k] + T[\ell] \le OPT$

So:

$$T[j] \le T[m+1] \le T[\max\{k, \ell\}] \le \frac{OPT}{2}$$

(since sorted)

Dfns for Approx:

Let $OPT(x)$ = value of optimal solution

$A(x)$ = value of solution computed by algorithm A

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

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

- $\alpha(n)$ is called the approximation factor.

So greedy load balancing:

$$A(x) \le 2\,OPT(x)$$

$$\frac{A(x)}{OPT(x)} \le 2$$

$$\frac{OPT(x)}{A(x)} \ge \frac{1}{2}$$

For this problem:

$$OPT(x) \le A(x)$$

Vertex Cover

NP-Hard.

Shall we try greedy again?

<u>How</u> should we be greedy?

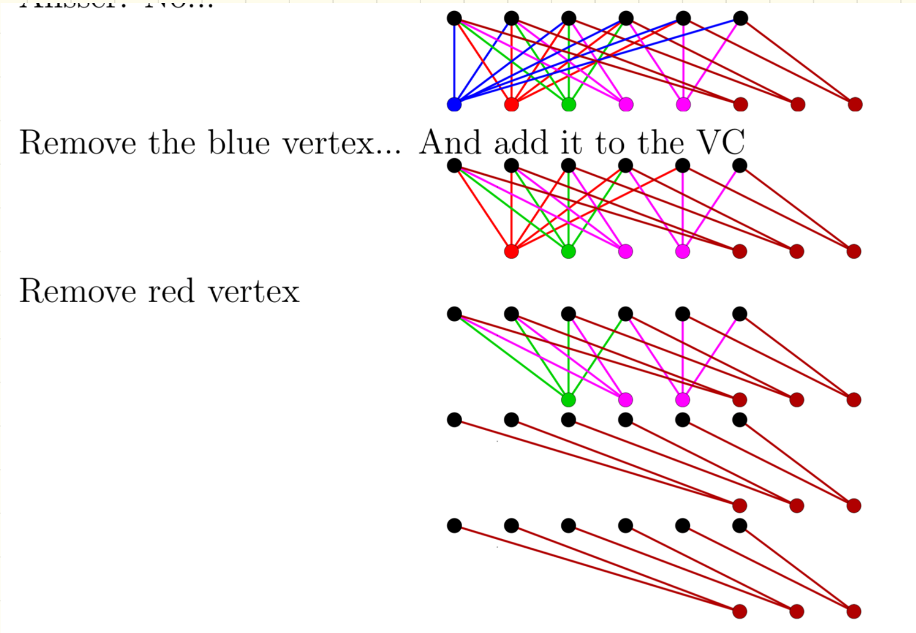
Algorithm:

```
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
```

Question: Is this ever optimal?

Q: Is it a 2-approx?

↳ NO:



OPT:

Greedy:

Thm: Greedy VC is an $O(\log n)$ approximation:

$$\text{Greedy} \leq O(\log n) \cdot OPT$$

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

Let d_i = max degree in G_i

```
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
```