Greedy Algs

To solve an optimization problem, incrementally build up sol’n at each step, choose what seems best “locally”

adv: simple, fast

disadv: may not be correct if correct, needs proof!

Ex1: Interval Scheduling

Given n intervals $[s_i, f_i], \ldots, [s_n, f_n]$ find largest subset of disjoint intervals

---

e.g. 

1 2 3 4 5 6

max 3

idea 1 (greedy) - pick shortest job (min $f_i - s_i$) fail!

idea 2 - pick job that starts (min $s_i$) earliest:
Counterex

Idea 3 - pick job that conflicts with fewest other jobs:
fail!

Counterex

Greedy 3 opt 4

Idea 4 - pick job that finishes earliest (min f_i):
bingo!

Greedy Alg'm:
1. repeat {
   1' pick (s_i, f_i) with smallest f_i
   2. remove [s_i, f_i] & all intervals intersecting it
   3. } until no intervals left
   4. return all intervals picked

Runtime: trivial $O(n^2)$ time
   better: sort & scan $O(n \log n)$ time

Correctness Pf:
   Let $I^*$ be an opt sol'n.
   Let $(s^*, f^*)$ be leftmost interval in $I^*$. 
Let \([s^*, f^*]\) be leftmost interval in \(I^*\).
Let \([s_i, f_i]\) be first interval chosen by above greedy algo.

\[ I^* \rightarrow s^* \rightarrow f^* \]

Know \(f_i \leq f^*\) by the way our greedy algo works.

\[ \Rightarrow \quad I^* - \{[s^*, f^*]\} \cup \{[s_i, f_i]\} \]

is a feasible sol'n with same \# intervals.
Reset \(I^* \leftarrow I^* - \{[s^*, f^*]\} \cup \{[s_i, f_i]\}\).

Remove \([s_i, f_i]\) & all intervals intersecting it.
& repeat argument (i.e. induction).

Rmk - does not extend to weighted case
(but can do DP)
- does not extend to 2D

Context: Scheduling to minimize avg wait time.
**Ex2: Scheduling to minimize avg wait time**

Given \( n \) jobs with process time \( p_1, \ldots, p_n \), find a re-ordering of the jobs to minimize total wait time.

\[
\text{Cost} = 0 + p_1 + (p_1 + p_2) + (p_1 + p_2 + p_3) + \ldots + (p_1 + \ldots + p_n)
\]

**Example:**

<table>
<thead>
<tr>
<th>jobs</th>
<th>1 2 3 4 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>3 4 1 8 2</td>
</tr>
</tbody>
</table>

*Wait time:*

\[
0 + 3 + (3+4) + (3+4+1) + (3+4+1+8) = 34
\]

*Better time:*

\[
0 + 3 + (3+4) + (3+4+1)+(3+4+1+2) = 28
\]

*Still better:*

\[
0 + 3 + (3+1) + (3+1+4) + (3+4+1+2) = 25
\]

Best:

\[
0 + 1 + (1+2) + (1+2+3) + (1+2+3+4) = 20
\]

**Greedy Algorithm:**

Just sort jobs in increasing \( p_i \).

**Correctness Pf:**

Let \( p_1^*, \ldots, p_n^* \) be opt order. Suppose it is not sorted.

Then \( p_i^* \neq p_i^* \), for some \( i \).
Then $p_i > p_{i+1}$ for some $i$.

Now swap $i$ and $i+1$.

```
Old cost: $p_1^* + \ldots + (p_i^* + \ldots + p_{i-1}^*) + (p_i + \ldots + p_{i+1}^*)$
  + $(p_{i+1}^* + \ldots + p_{i+m}^*) + \ldots + (p_{i+m}^* + p_{n-i}^*)$

New cost: $p_1^* + \ldots + (p_i^* + \ldots + p_{i-1}^*) + (p_i^* + \ldots + p_{i+m}^*)$
  + $(p_{i+1}^* + \ldots + p_{i+m}^*) + \ldots + (p_{i+m}^* + p_{n-i}^*)$
``` 

New cost - Old cost = $p_{i+1}^* - p_i^* < 0$

$\Rightarrow$ new sol'n is strictly better:

**Contradiction!**

**Rmk:** extends to weighted version

$\text{cost} = w_1 p_1 + w_2 p_2 + w_3 (p_1 + p_2) + \ldots + w_n (p_{i-1} + p_i)$

Correct greedy: Sort in increase $P_i / w_i$

```
Old cost: $\ldots + w_i^* (p_i + \ldots + p_{i-1}^*) + w_i^* (p_i^* + p_{i+1}^*)$

New cost: $\ldots + w_i^* (p_i + \ldots + p_{i-1}^*) + w_i^* (p_i^* + p_{i+1}^*)$
  + \ldots$
``` 

New cost - Old cost

$= w_i^* p_{i+1}^* - w_i^* p_i^*$

$< 0$ if $\frac{p_i^*}{w_i^*} > \frac{p_{i+1}^*}{w_{i+1}^*}$