Scheduling

Suppose you are trying to run many applications on a processor.

- Java
- Python
- Skype
- Itunes
- KOTR

What order is best?

Each job $i; i \in \{1, \ldots, n\}$
- Weight (importance) $w_i$
- Time $t_i$ to complete

Def: Completion time $C_j$ of job $j$ is sum of times required to complete all jobs run before $j$, plus $t_j$.

Q: Suppose there are 3 jobs, with $t_1 = 1, t_2 = 2, t_3 = 3$, and that they are run in reverse order. What is $C_1, C_2, C_3$?

A) 3, 2, 1  
B) 1, 2, 3  
C) 3, 6, 7  
D) 3, 5, 6
Scheduling Goal:

Minimize \( \sum_{j=1}^{n} w_j C_j \) \( \Rightarrow \) "Objective function"

Q: Why is this a good goal?
A: If important jobs left until end, \( w_j C_j \rightarrow \text{large} \)

\[ \uparrow \]
\[ \text{large large large} \]

Ex:

<table>
<thead>
<tr>
<th>Job</th>
<th>Time</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Order | A
---|---
123  | 132 213 231 312 321
Greedy algorithm: picks best job to schedule first

What is best? Create function \( f(w_i, t_i) \), find job with largest \( f \)-value. Put first. Repeat!

Q: Create a good \( f \):
A: Good jobs have large weight, short time
   \[ f_1 = \frac{w_i}{t_i}, \quad f_2 = w_i - t_i, \]
   \[ f = \frac{w_i}{t_i} + w_i - t_i \]

Which to choose? 😕

Try to find simple examples where behave differently!

<table>
<thead>
<tr>
<th>job</th>
<th>weight</th>
<th>time</th>
<th>( f_1 )</th>
<th>( f_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1/3</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2/5</td>
<td>-3</td>
</tr>
</tbody>
</table>

If follow \( f_1 \) → order \((2, 1)\)
If follow \( f_2 \) → order \((1, 2)\)

\( f_1 \) is better! In fact, \( f_1 \) is optimal!
Thm: Greedy algorithm choosing by $w_i/t_i$ is optimal.

Pf: EXCHANGE argument & Pf by contradiction

Assume
- $w_i/t_i$ are distinct $\forall i \in \{1, 2, \ldots, n\}$
- $w_1/t_1 > w_2/t_2 > \ldots > w_n/t_n$ (rename)

Let $\sigma$ be ordering using greedy
Let $\sigma^*$ be optimal ordering.

Suppose for contradiction, $\sigma^*$ is better than $\sigma$

Note: $\exists k, j \text{ s.t. } w_j/t_j > w_k/t_k$, but $j$ is immediately after $k$ in $\sigma^*$ ordering.

(Otherwise, $\sigma^* = \sigma$)

Let's create a new ordering $\sigma^{*'}$ that is same as $\sigma^*$, but with $k, j$ positions switched

Q: If $\sigma^*$ has objective value $A_{\sigma^*}$, and $\sigma^{*'}$ has objective value $A_{\sigma^{*'}}$, how are $A_{\sigma^*}, A_{\sigma^{*'}}$ related?

$$A_{\sigma^*} = A_{\sigma^{*'}} + \frac{x}{x}$$
\( T = \text{time to complete first set of jobs} \)

\[
A_{\sigma^*} = \sum w_i C_i + w_k (T + t_k) + w_j (T + t_k + t_j) + \sum w_r c_r
\]

\[
A_{\sigma^*'} = \sum w_i C_i + w_j (T + t_j) + w_k (T + t_j + t_k) + \sum w_r c_r
\]

\[
A_{\sigma^*} = B + w_j t_k
\]

\[
A_{\sigma^*'} = B + w_k t_j
\]

\[
\frac{w_j}{t_j} > \frac{w_k}{t_k} \quad \Rightarrow \quad w_j t_k > w_k t_j
\]

\[ A_{\sigma^*'} < A_{\sigma^*} \quad \Rightarrow \text{Contradiction!} \]

This is called an exchange argument. Show that a small alteration in a solution leads to a better solution.

What is run-time: \( O(n \log n) \) (need to sort \( n \) items)
Now let's get rid of the assumption $w_i/t_i$ distinct $\forall i \in \{1, \ldots, n\}$

Q: Why does the old proof fail?

A: There might not be a unique solution

B) Objective function might not decrease from $\sigma^*$ to $\sigma^{*'}$

C) We can't create an ordering such that $w_1/t_1 > w_2/t_2 > \ldots > w_n/t_n$

D) All of the above

Still use EXCHANGE argument, but now need more exchanges.

Choose some ordering s.t.

$$w_1/t_1 \geq w_2/t_2 \geq w_3/t_3 \geq \ldots \geq w_n/t_n$$

Let $\sigma$ be strategy using this ordering

Let $\sigma^*$ be some other strategy

We will show $A_{\sigma^*} \geq A_{\sigma}$

This procedure is bubble sort!
Conclusion: $A_{opt} \geq A_{\sigma}$, so greedy strategy is still optimal.

Q: What is run time?

A) $O(n)$  B) $O(n \log n)$  C) $O(n^2)$  D) $O(1)$

A: $O(n \log n)$. Need to sort $n$ items, requires $O(n \log n)$ time.