A greedy Algorithm for scheduling Jobs with Deadlines & Profits, on Multiple Processors

INPUT: $(d_1, g_1)\ (d_2, g_2) \ldots (d_n\ g_n)$, $m \leq n$

(d_i: deadline, g_i: profit)

SIMILAR TO PREVIOUS PROBLEM BUT NOW THERE ARE $m \leq n$ PROCESSORS (NOT ONLY ONE.)

A **schedule** $S: \{1, \ldots, m\} \times \{1, 2, \ldots, n\} \rightarrow \{0, 1, \ldots, n\}$
where $S(k,t)$ is the job scheduled on processor k in slot t. $S(k,t)=0$ means NO job scheduled on processor k at time slot t.

S is **feasible** if

(a) $S(k,t) = i > 0 \Rightarrow t \leq d_i$ (scheduled jobs meet deadlines)

(b) If $S(k,t) = S(k',t')$ then $k=k'$, $t=t'$
(each job scheduled at most once)

The **profit** of feasible S, $P(S) = \sum_{k,t} g_{S(k,t)}$

OUTPUT: A feasible schedule with optimal (maximum) profit.

Greedy Algorithm SAME AS BEFORE

Sort jobs $g_1 \geq \ldots \geq g_n$
For $i=1$ to n
schedule job i for last available time slot so as to meet deadline (on any processor) if possible.

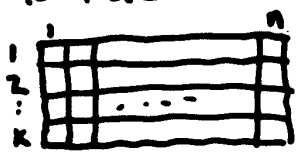
Lemma
The greedy algorithm finds the optimal schedule.

Proof
The proof is similar to the previous proof. We prove by induction on i, that the schedule after stage i, S_i, is promising; $P(i)$ holds.

$P(0)$ clearly holds.
IND STEP: ASSUME $P(i)$ And show $P(i+1)$
By assumption, there is an optimal schedule S_{opt} extending S_i and using jobs $\{i+1, .., n\}$.

Case 1 Job $i+1$ NOT scheduled by greedy algorithm
Then Job $i+1$ NOT scheduled in S_{opt}, so S_{opt} extends S_{i+1} using only jobs $\{i+2, .., n\}$

Case 2a Job $i+1$ scheduled at (k_0, t_0) by greedy algorithm, and Job $i+1$ scheduled at (k_1, t_1) by S_{opt}.
As before we know $t_1 < t_0$.
Also Without loss of generality, $k_0 = k_1$. This is true because if we view S_{opt} as a matrix



with n columns and k rows,
for any column, the values in that column can be permuted and the resulting matrix is still a feasible, optimal schedule.

$t_1 = t_0 \Rightarrow$ we are done.
$t_1 < t_0 \Rightarrow$ same argument as before.

Case 2b Job $i+1$ does NOT occur in S_{opt}.
same argument as before.

■

Activity Scheduling

An **activity** is a pair (s, f) $\quad s, f \geq 0,\ s < f$ where s is the start time of the activity and f is the finish time.

INPUT $(s_1, f_1) \ldots (s_n, f_n)$

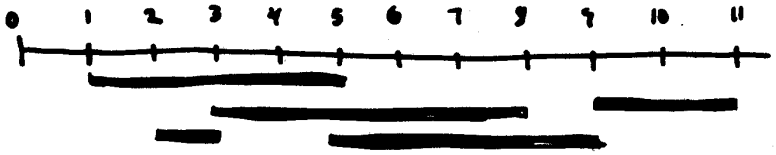
A **schedule** is a set $A \subseteq \{1, 2, \ldots, n\}$ such that for all $i, j \in A$ if $i \neq j$ then (s_i, f_i) does NOT overlap (s_j, f_j). (They do <u>not</u> overlap if $f \leq s'$ or $f' \leq s$.)

OUTPUT A schedule A of maximum size. ($|A|$ is the largest possible)

Example

$(1, 5)\ (3, 8)\ (2, 3)\ (9, 11)\ (5, 9)$

$A = \{(2, 3), (3, 8), (9, 11)\}$ is an optimal schedule



GREEDY ALGORITHM

1. SORT activities by Nondecreasing <u>finish</u> times $f_1 \leq f_2 \leq \ldots \leq f_n$
2. for $i = 1, \ldots, n$
   Schedule activity i if possible (if adding it does not lead to any overlaps)

Theorem The greedy algorithm outputs an optimal (largest possible) schedule.

Proof Let A_i be schedule after step i of loop.

Let e_i be finish time of last activity added to A up to step i.

Thus $A_i \subseteq \{1, 2, .., i\}$

$e_i = \max\{f_j \mid j \in A_i\}$ $(\max \phi = 0)$

Lemma For $0 \le i \le n$ A_i is promising after step i of loop.

That is, there is an optimal schedule A_{opt} such that

$A_i \subseteq A_{opt} \subseteq A_i \cup \{i+1, .., n\}$.

↑ This lemma completes the proof of the theorem.

Proof of Lemma

$i = 0$: clearly A_0 is promising after step 0

IND HYP: ASSUME A_i is promising after step i.

Let $A_i \subseteq A_{opt} \subseteq A_i \cup \{i+1, .., n\}$

$i = i+1$: show A_{i+1} is promising after $i+1$.

Clearly $f_{i+1} \ge e_i$.

Case 1 $s_{i+1} < e_i$ so job i NOT added.

Then job $i+1$ overlaps with some already scheduled activity so $A_i = A_{i+1} \subseteq A_{opt} \subseteq A_{i+1} \cup \{i+2, .., n\}$

Case 2a $s_{i+1} \ge e_i$ so job i is added, $A_{i+1} = A_i \cup \{i+1\}$ and $i+1 \in A_{opt}$. Then clearly A_{opt} extends A_{i+1} using only $\{i+2, .., n\}$

Case 2B $s_{i+1} \geq e_i$, $A_{i+1} = A_i \cup \{i+1\}$ but $i+1 \notin A_{opt}$

$s_{i+1} \geq e_i$ means that activity $i+1$ does not overlap with any activity in A_i.

A_{opt} cannot equal A_i since then $A_{opt} \cup \{i+1\}$ would be a larger schedule than A_{opt}

Thus let $u \geq i+2$ be the activity in $A_{opt} - A_i$ with smallest finish time.

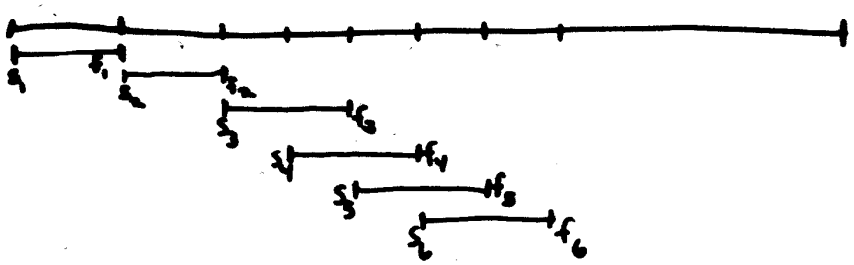
Let $A'_{opt} = (A_{opt} - \{u\}) \cup \{i+1\}$.

No activity in $A_{opt} - A_i - u$ overlaps with $i+1$ because each of these have start times larger than $f_u \geq f_{i+1}$

Thus A'_{opt} is a schedule of the same size as A_{opt}

and $A_{i+1} \subseteq A'_{opt} \subseteq A_{i+1} \cup \{i+2, \ldots, n\}$ ∎

Example



$A_3 = (s_1, f_1)(s_2, f_2)(s_3, f_3)$

$A_{opt} = (s_1, f_1)(s_2, f_2)(s_4, f_4)(s_6, f_6)$

Activity Scheduling on Multiple Processors

INPUT $(s_1, f_1)\ (s_2, f_2)\ \ldots\ (s_n, f_n)$, $m \leq n$

($s_1, f_1)\ldots(s_n, f_n)$: activities; m: number of processors

A **schedule** is a sequence $A = (A_1\ A_2\ \ldots\ A_m)$
where $A_k \subseteq \{1, 2, \ldots, n\}$ and such that:

(a) $\forall k\ \ 1 \leq k \leq m$, $\forall i, j \in A_k$, if $i \neq j$ then activity i does not overlap with activity j

(b) $\forall k_1, k_2\ \ 1 \leq k_1, k_2 \leq m$, if $k_1 \neq k_2$ then $A_{k_1} \cap A_{k_2} = \phi$
(each job scheduled at most once)

The **size** of A is $|A_1| + |A_2| + \cdots + |A_m|$.

OUTPUT A of optimal (maximum possible) size.

EXERCISE give a polynomial-time algorithm for this problem. Prove that your algorithm works.

WHEN DO GREEDY ALGORITHMS WORK?

BEWARE: OFTEN A GREEDY ALGORITHM WILL NOT PRODUCE A GOOD SOLUTION.

AND IT IS HARD TO DETERMINE WHEN THEY WILL AND WILL NOT WORK ON A GIVEN PROBLEM, EXCEPT BY EXPERIENCE AND INSIGHT.

KNAPSACK PROBLEM

INPUT: weights $w_1, w_2, .., w_d \in \mathbb{N}$
Knapsack weight $C \in \mathbb{N}$

FOR each $S \subseteq \{1, .., d\}$ let $K(S) = \sum_{i \in S} w_i$

OUTPUT

$$M = \max_{S \subseteq \{1, .., d\}} \{K(S) \mid K(S) \leq C\}$$

(Output a bunch of items of maximum wt that can be put in a knapsack of weight capacity C)

Greedy Algorithm For Knapsack

1. Sort weights $w_1 \geq w_2 \geq .. \geq w_d$
2. For $i = 1$ to d
   Add w_i as long as items so far do not exceed capacity

The greedy algorithm just described does NOT always output a correct answer.

Example

$w_1 = 501 \quad w_2 = 500 \quad w_3 = 500 \qquad C = 1000$

algorithm will output $M = 501$, $S = \{1\}$

but optimal answer is $M = 1000$, $S = \{2, 3\}$

Nonetheless, the greedy algorithm always outputs a number that is at least $\frac{1}{2}$ of the optimal answer.

Lemma For all instances of Knapsack, the answer, $\bar{M}$, returned by the greedy algorithm on that instance is at least $\frac{1}{2}M$, where M is the optimal value on that instance.

Proof idea

Show that if $\bar{M} \neq M$, then $\bar{M} \geq \frac{1}{2}C$.
Since $C \geq M$, the lemma follows.