HW 3 - Solutions

Problem 1 Weighted Interval Scheduling (Weighted Task Selection)

Given a set of jobs $J = \{1, \dots, n\}$.

the start & finish times of i^{th} job $= (s_i, f_i)$

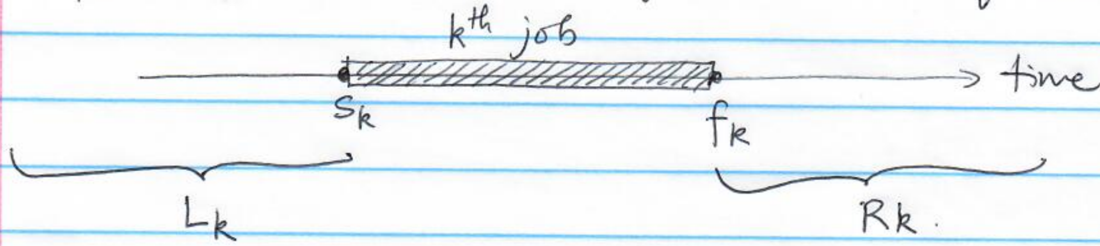
Value of i^{th} job $= v_i$.

Goal: Find the subset of mutually compatible jobs that have max. total value.

no two jobs overlap in time.

① Multi-stage decision problem:

Say, we pick the k^{th} job in the first decision stage.



Let L_k = subset of jobs in J that are mutually compatible to k^{th} job, and finish before s_k.

P.T.O.

$|||^{ly}$, let R_k = subset of jobs in J that start after f_k

$\Rightarrow$ these jobs are also mutually compatible with k^{th} job.

If $V(S')$ = optimal value for the subproblem with a subset of jobs S',

then, $$V(J) = \max_{k \in \{1, \ldots, n\}} v_k + V(L_k \cup R_k).$$

However, if the jobs in J are sorted in the increasing order of the finish times, i.e.,

$$f_1 \leq f_2 \leq \cdots \leq f_n,$$

then, $$V(J) = \max_{k \in \{1, \ldots, n\}} v_k + V(L_k) + V(R_k).$$

$$= \max_{k \in \{1, \ldots, n\}} v_k + V(\{1, \ldots, k_L\}) + V(\{k_U, \ldots, n\})$$

where k_L is the mutually compatible job to k with the largest finish time $\leq s_k$

and k_U is the mutually compatible job to k with smallest start time $\geq f_k$.

However, these Bellman equations are not very useful for our implementation.

For the sake of simple notation,

let $X = \{x_1, \dots, x_m\}$ ← Set of all time-markers relevant to this problem setting.

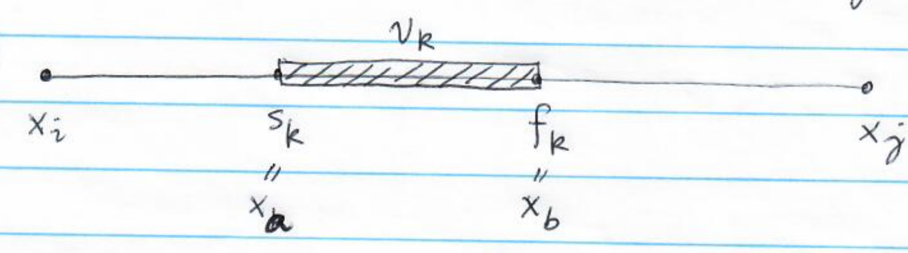
$= \{s_1, \dots, s_n\} \cup \{f_1, \dots, f_n\} \cup \{-\infty, \infty\}$.

$\Rightarrow \quad m \le 2n+2 \quad$ since $x_i < x_j \;\; \forall \; i < j$.

Let $M[i,j]$ = max. value of subset of jobs selected within the interval (x_i, x_j), $i<j$.

State variable : (i,j)

Decision variable : k.



then, $M[i,j] = \begin{cases} \max\limits_k \left\{ v_k + M[i,a] + M[b,j] \right\}, & \text{if } \exists \text{ job } k \text{ s.t. } x_i < s_k < f_k < x_j \\ 0 & \text{otherwise.} \end{cases}$

← Bellman equation.

Note : Matrix M =

$$\begin{array}{c|cccccc} & x_1=-\infty & x_2 & x_3 & \cdots & x_{m-1} & x_m=\infty \\ \hline x_1=-\infty & 0 & & & \cdots & & \\ x_2 & 0 & 0 & & \cdots & & \\ x_3 & 0 & 0 & 0 & \cdots & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m-1} & 0 & 0 & 0 & \cdots & 0 & \\ x_m=\infty & 0 & 0 & 0 & \cdots & 0 & 0 \end{array}$$

(2) Pseudocode:

```
DP-Task Selection (J)

        \\ Initialize time marker set by sorting jobs acc. to finish times
        x = {-∞, ∞}
        for i = 1 to n
                x = x ∪ {i.s, i.f}   \\ sorted time-markers.
        M = 0   \\ Initialize (m+1) × (m+1) all-zero matrix.
        for i = 1 to x.length
                for j = 1 to x.length
                        C = Compatible Jobs (J, i, j)
                        if C ≠ ϕ
                                for each job k ∈ C
                                        if M[i,j] ≤ k.v + M[i, k.s]
                                                        + M[k.f, j]
                                                M[i,j] = k.v + M[i, k.s] +
                                                        M[k.f, j]
        return M[1, x.length]
```

P.T.O.

where

Compatible Jobs (J, x, y)

\\ Find all compatible jobs between time markers x & y.

$G = \phi$

for $i = 1$ to n

if $x \leq i.s$ and $i.f \leq y$

$G = G \cup \{i\}$

return G.

③ <u>Greedy algorithm</u> (One possible solution)

Similar to Fractional Knapsack, let us define a normalized value $w_i = \dfrac{v_i}{f_i - s_i}$ $\forall$ $i = 1, \cdots, n$.

(value per unit time)

Greedy approach $\Rightarrow$ select the job with max. w_i such that it is compatible to previously selected jobs.

P.T.O.

Greedy-TaskSelection (J)

for $i = 1$ to n

$w_i = v_i / (f_i - s_i)$.

Sort all jobs in J in decreasing order of w.

$m = 0$.

while $J = \phi$

u = extract-max (J)

$m = m + u.v$

J = greedy-compatible(J, u)

return m

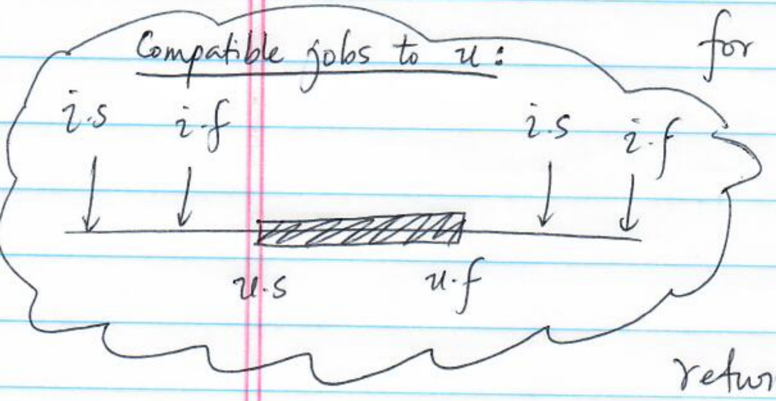
where greedy-compatible (J, u)

for each job i in J

if $(i.s > u.f)$ or $(i.f < u.s)$ is FALSE

$J = J - \{i\}$

return J.



Eliminating a dimension in DP — "Fast DP"

In the earlier DP solution, we picked an arbitrary job J_k and employed divide-and-conquer approach to solve L_k and R_k subproblems.

Question: Can we reduce this ~~problem~~ to a one-dimensional algorithm?

Answer: Yes.

Introduce an order on the subproblems.

Let $M[i]$ = max. weight solution to the subproblem

$$\{ [s, f] \in J \mid f < \underset{\uparrow}{x_i} \}$$

i^{th} time marker.

$$\Rightarrow M[i] = \begin{cases} 0 & \text{if } i = 1 \\ \max \begin{cases} M[i-1], \\ \max \left\{ M[j] + w([x_i, r]) \;\middle|\; \begin{array}{l} [x_j, f] \in J \text{ and} \\ f \in [x_{i-1}, x_i] \end{array} \right\} \end{cases} & \text{otherwise.} \end{cases}$$

Fast DP – Task Selection (J)

\\ Sort J according to finish times and create time markers.

$x = \{\ \}$

for $i = 1$ to n

$\quad x = x \cup \{ i.s,\ i.f \}$

for $i = 1$ to $x.\text{length}$

$\quad M[i] = 0$

for $i = 1$ to n

$\quad$ Search t such that $r_i \in [y_t ; y_{t+1})$

$\quad$ Search s such that $l_i = y_s$.

$$M[t] = \max \begin{Bmatrix} M[t-1], \\ M[s] + w[l_i, r_i] \end{Bmatrix}$$

Problem 2 String edit problem

(1) Given two strings $A = a_1 a_2 \ldots a_n$ and $B = b_1 \ldots b_m$, if our goal is to convert A into B using operations {Insert, Delete, Substitute}, we have the following possibilities —

(*) In order to match the character "b_1", we can evaluate all the possibilities

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}, \begin{pmatrix} a_1 & a_2 \\ b_1 & \end{pmatrix}, \ldots, \begin{pmatrix} a_1 & \cdots & a_n \\ b_1 & & \end{pmatrix}$$

(*) In order to match the ~~character~~ substring "$b_1 b_2$", we can evaluate the foll. possibilities:

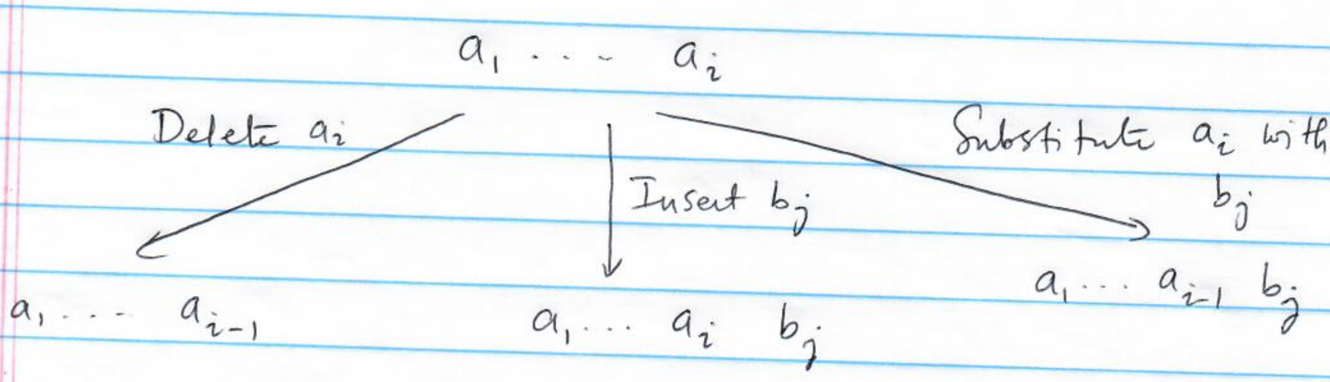
$$\begin{pmatrix} a_1 & \\ b_1 & b_2 \end{pmatrix}, \begin{pmatrix} a_1 & a_2 \\ b_1 & b_2 \end{pmatrix}, \ldots, \begin{pmatrix} a_1 & \cdots & a_n \\ b_1 & b_2 & \end{pmatrix}$$

$$\vdots$$

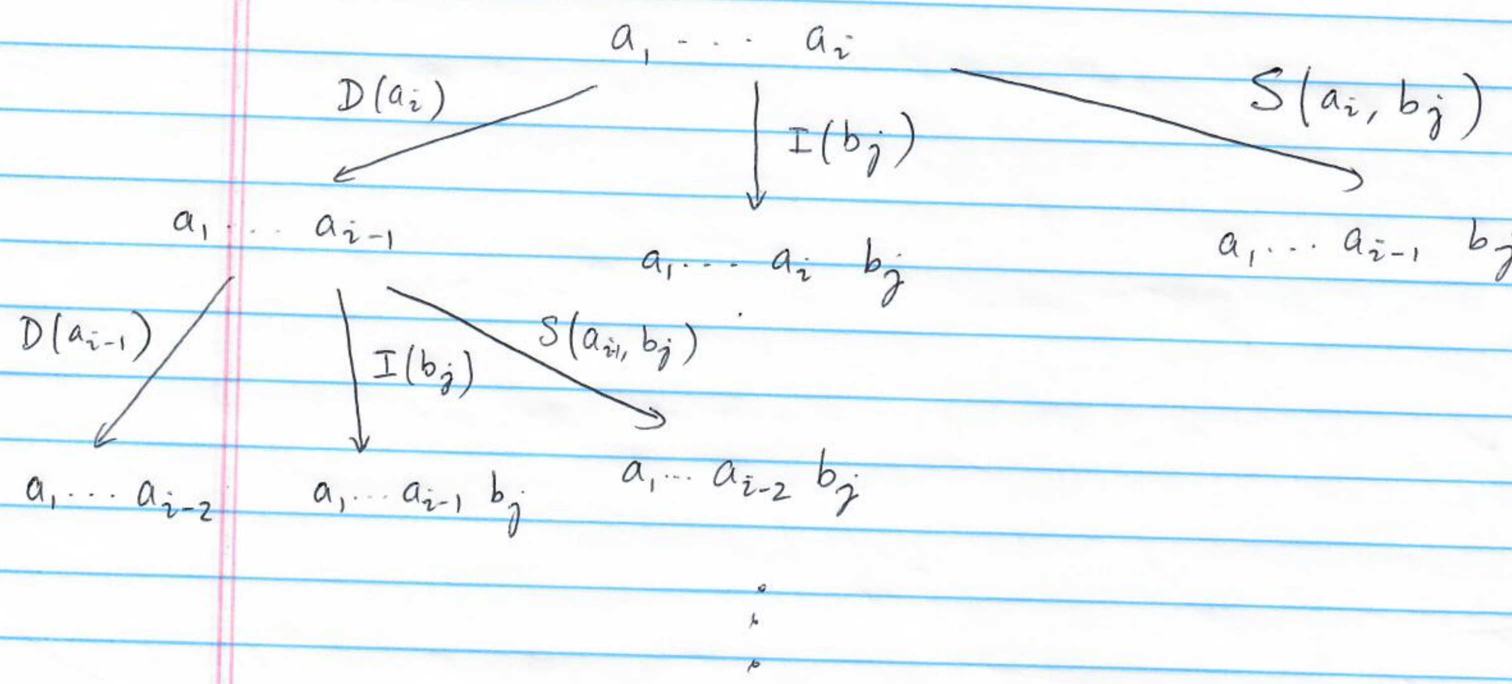
In other words, we need to evaluate all possible cases: $\begin{pmatrix} a_1 & \cdots & a_i \\ b_1 & \cdots & b_j \end{pmatrix}$ and identify a recursion.

Multi-stage decision problem:

~~Say~~ Consider the problem of converting $(a_1 \cdots a_i)$ into $(b_1 \cdots b_j)$. $\Rightarrow$ <u>State variable</u> $= (i, j)$



In general, if we were to build this tree,



Eventually, the goal is to reach the state where the entire sequence becomes $b_1, \ldots b_j$.

Say, $c[i,j]$ is the minimum cost of converting $(a_1 \dots a_i)$ into $(b_1 \dots b_j)$.

Since the first decision in the decision tree shown in Page 2.2 is as follows:

$$
\begin{tikzcd}[column sep=small]
 & \begin{pmatrix} a_1 \cdots a_i \\ b_1 \cdots b_j \end{pmatrix} \arrow[dl, "\text{Delete } a_i"'] \arrow[d, "\text{Insert } b_j"] \arrow[dr, "\text{Substitute } b_j"] & \\
\underbrace{\begin{pmatrix} a_1 \cdots a_{i-1} \\ b_1 \cdots b_j \end{pmatrix}}_{\text{Subproblem}} & \underbrace{\begin{pmatrix} a_1 \cdots a_i \\ b_1 \cdots b_{j-1} \end{pmatrix}}_{\text{Subproblem}} \underbrace{\boxed{\begin{matrix} b_j \\ b_j \end{matrix}}}_{\text{MATCH}} & \underbrace{\begin{pmatrix} a_1 \cdots a_{i-1} \\ b_1 \cdots b_{j-1} \end{pmatrix}}_{\text{Subproblem.}} \underbrace{\boxed{\begin{matrix} b_j \\ b_j \end{matrix}}}_{\text{MATCH}}
\end{tikzcd}
$$

The state evolution reduces to the following:

$$
\begin{tikzcd}
 & (i,j) \arrow[dl, "\text{Delete } a_i"'] \arrow[d, "\text{Insert } b_j"] \arrow[dr, "\text{Substitute } a_i \text{ with } b_j"] & \\
(i-1,j) & (i,j-1) & (i-1,j-1)
\end{tikzcd}
$$

$$
\Rightarrow \quad c[i,j] = \min_x \begin{cases} c[i-1,j] + D(a_i) \ ; & x = D \\ c[i,j-1] + I(b_j) \ ; & x = I \\ c[i-1,j-1] + S(a_i,b_j) \ ; & x = S \end{cases}
$$

"Bellman equation."

where x is the decision variable.

More formally, if $x = \begin{cases} -1 & ; \text{ Delete} \\ 0 & ; \text{ Insert} \\ 1 & ; \text{ Substitute,} \end{cases}$ we have

$$c[i,j] = \min_{x \in \{-1,0,1\}} \left[\frac{x(x-1)}{2} \left\{ c[i-1,j] + D(a_i) \right\} - \frac{(x-1)(x+1)}{2} \left\{ c[i,j-1] + I(b_j) \right\} + \frac{x(x+1)}{2} \left\{ c[i-1,j-1] + S(a_i, b_j) \right\} \right]$$

where $c[0,j] = \sum_{k=1}^{j} I(b_k)$ for all $j = 1, \cdots, m$

$c[i,0] = \sum_{k=1}^{i} D(a_k)$ for all $i = 1, \cdots, n$

and $c[0,0] = 0$.

Pseudocode :

Matrix $c \Longrightarrow$

	ϕ	$\{b_1\}$	$\cdots$	$\{b_1 \cdots b_m\}$
ϕ	0	$c[0,1]$	$\cdots$	$c[0,m]$
$\{a_1\}$	$c[1,0]$	$c[1,1]$	$\cdots$	$c[1,m]$
$\{a_1 a_2\}$	$c[2,0]$	$c[2,1]$	$\cdots$	$c[2,m]$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\{a_1, \cdots, a_n\}$	$c[n,0]$	$c[n,1]$	$\cdots$	$c[n,m]$

Pseudocode

Size (A) = n
Size (B) = m.

DP- StringEdit (A, B, I, D, S)

C = 0 \\ Initialize all-zero matrix of size (n+1) x (m+1).

for $i = 1$ to n
$C[i,0] = C[i-1,0] + D(a_i)$

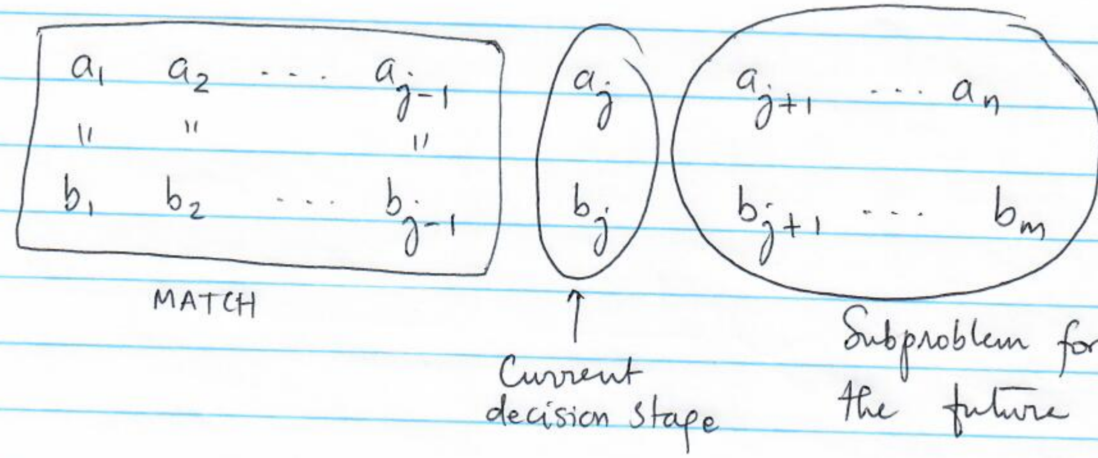
for $j = 1$ to m
$C[0,j] = C[0,j-1] + I(b_j)$

for $i = 1$ to n
for $j = 1$ to m

$$C[i,j] = \min \left\{ \begin{array}{l} C[i-1,j] + D(a_i), \\ C[i,j-1] + I(b_j), \\ C[i-1,j-1] + S(a_i, b_j) \end{array} \right\}$$

return $C[n,m]$

(2) <u>Greedy Algorithm</u> (One possible solution)



Greedy - String Edit (A, B, I, D, S)

$C = 0$

$j = 1$

while $(a_j = \phi)$ and $(b_j = \phi)$ is FALSE

if $a_j = \phi$ and $b_j \neq \phi$

$C = C + I(b_j)$

$j = j + 1$

else if $a_j \neq \phi$ and $b_j = \phi$

$C = C + D(a_j)$

$j = j + 1$

else

$\text{minCost} = \min \{ I(b_j), D(a_j), S(a_j, b_j) \}$

$C = C + mincost$

if $D(a_j) \neq mincost$

$j = j + 1$

return C.