Weighted Interval Scheduling

- No greedy algorithm known.
- Dynamic programming works.

Problem - Given n jobs with start and finish time

- $J_1 = [s_1, f_1]$, $J_2 = [s_2, f_2]$, $\ldots$, $J_n = [s_n, f_n]$.
- Job J_i has value $v_i \geq 0$.

Goal To find a set of pairwise non-overlapping jobs with maximum total value.

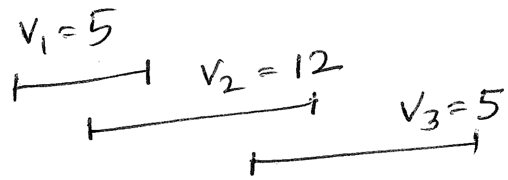
Recall The interval scheduling problem studied earlier corresponds to the case where all $v_i = 1$. The algorithm (greedy) sorts the jobs according to finish time, say,

$$f_1 \leq f_2 \leq f_3 \ldots \leq f_n$$ and

for $i = 1, 2, \ldots, n$, picks i^{th} job if it does not overlap with previously chosen jobs.

This does not work for general values.

Example



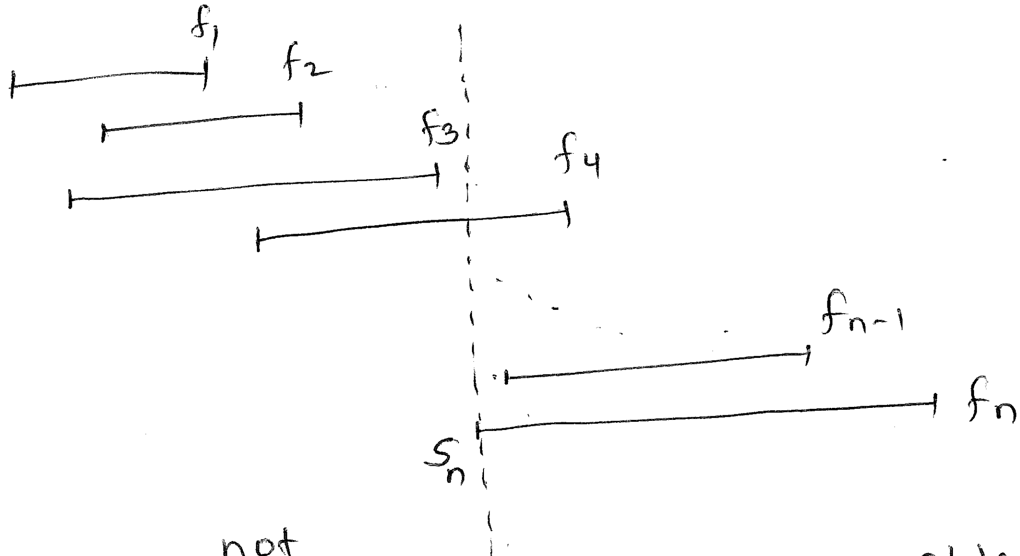
The greedy algorithm will pick $\{1,3\}$ but the optimal solution is $\{2\}$.

Dynamic Programming Algorithm

- Order jobs according to finish time, say

$$f_1 \le f_2 \le f_3 \dots \le f_n$$

Idea



- If J_n is not selected then the problem reduces to $\{J_1, J_2, \dots, J_{n-1}\}$.
- If J_n is selected, then problem reduces to $\{J_1, \dots, J_K\}$ s.t. k is largest index s.t. $f_K < S_n$. (K=3 in picture).

Subproblems

$OPT(J_1, J_2, \dots, J_t)$ = max value of non-overlapping set of jobs from $\{J_1, J_2, \dots, J_t\}$.

These are all "prefixes".

subproblems = n.

original problem = $\{J_1, \dots, J_t\}$.

Recursive formula

$$OPT(J_1, \dots, J_t) = \max \begin{cases} OPT(J_1, J_2, \dots, J_{t-1}) \\ v_t + OPT(J_1, \dots, J_k) & \text{where } k \text{ is largest index s.t. } f_k < s_t. \end{cases}$$

Order According to length of prefix.

Base case $OPT(J_1) = v_1$.

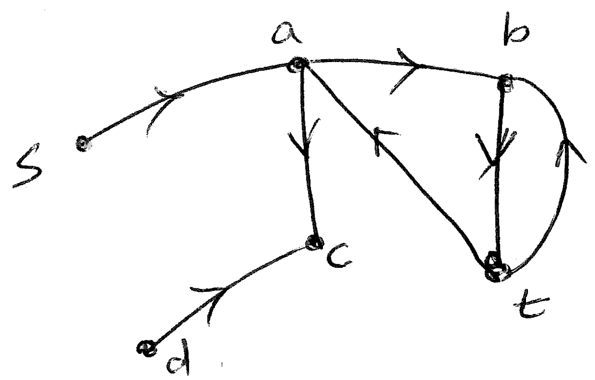
Shortest Path in Directed Graphs

Def A directed graph $G(V,E)$ consists of a set of vertices V and set of directed edges E.

$$E \subseteq \{(u,v) \mid u,v \in V,\ u \neq v\}$$

↑ ordered pair

Example



$V = \{a, b, c, d, s, t\}$

$E = \{(s,a), (a,c), (a,b), (b,t), (d,c), (t,a), (t,b)\}$.

Problem

- Given a dir. graph $G(V,E)$
  - each edge $(u,v) \in E$ has cost $c_{uv} \geq 0$.
- gives start node s and target node t.

Goal - to find the path from s to t that has least cost.

Bellman-Ford Algorithm

Obs For any two nodes u, v, the $u \rightsquigarrow v$ path with minimum cost has no cycles, so has $\leq n-1$ edges. $(|V| = n)$.

Subproblems

- $OPT(u, i)$ = minimum cost of a path from s to u that has $\leq i$ edges (∞ if no such path exists).

- original problem: $OPT(t, n-1)$.
- # subproblems: $O(n^2)$.

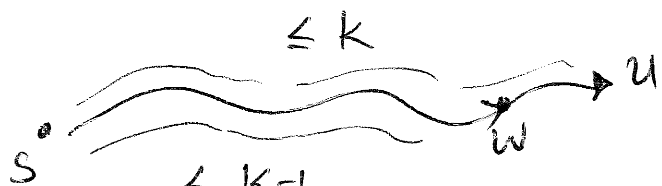
Base case $OPT(u,0) = \begin{cases} 0 & \text{if } u = s \\ \infty & \text{otherwise.} \end{cases}$

Recursive formula

$$OPT(u,k) = \min \begin{cases} OPT(u, k-1) \\ \min\limits_{w : (w,u) \in E} c_{wu} + OPT(w, k-1) \end{cases}$$

Order According to i (in $OPT(u,i)$).

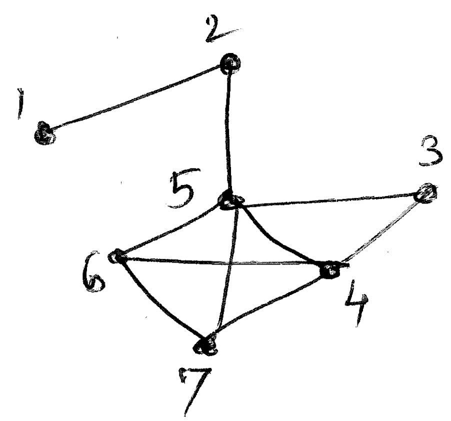
Observe that the recursive formula is based



on the following fact: min-cost ~~shortest~~ $s \rightsquigarrow u$ path with $\leq k$ edges has either $\leq k-1$ edges or has some w as the "last-but u" vertex and has min-cost $s \rightsquigarrow w$ path as prefix with $\leq k-1$ edges.

Maximum Independent Set in Trees

Def In an undirected graph $G(V,E)$, an independent set S is a subset of the vertex set V that contains no edge inside it, i.e. $\forall\, u,v \in S,\ \{u,v\} \notin E$.



Independent sets:

$\{1, 3, 6\}$.

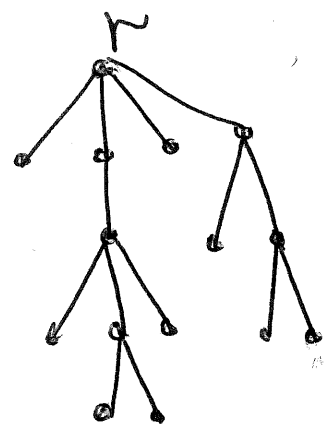
$\{2, 7, 3\}$.

$\{1, 5\}$.

Note. Finding independent sets in general graphs maximum-sized is NP-complete.

Problem Given a tree $T(V,E)$, find an independent set of the maximum size!

W.l.o.g. T is a rooted tree.



<u>Idea</u> Let r be the root. We consider two cases depending on whether r is included in an independent set or not.

Define:

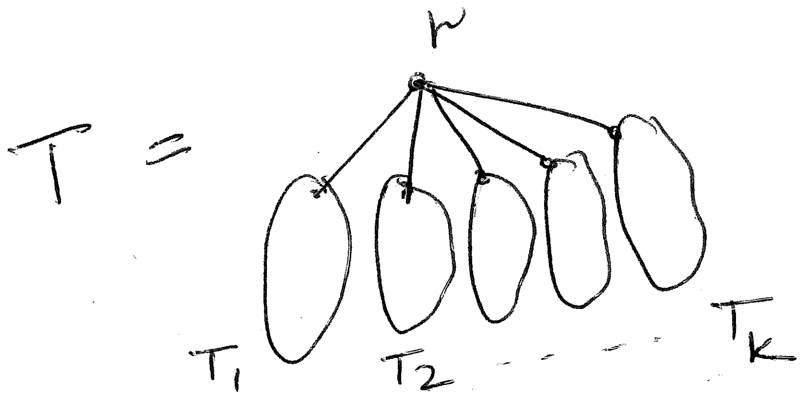
IS (T) = size of maximum independent set in T.

IS_with_root (T) = maximum size of an independent set in T that includes the root

IS_without_root (T) = max size of an indep. set in T that excludes the root.

Note

$$IS(T) = \max\{IS\_with\_root(T),\ IS\_without\_root(T)\}.$$



Recursive formula

$$IS\_without\_root(T) = \sum_{i=1}^{k} IS(T_i).$$

$$IS\_with\_root(T) = 1 + \sum_{i=1}^{k} IS\_without\_root(T_i).$$

$$IS(T) = \max\{IS\_with\_root(T),\ IS\_without\_root(T)\}.$$