Greedy Algorithms

1. Interval Scheduling Problem

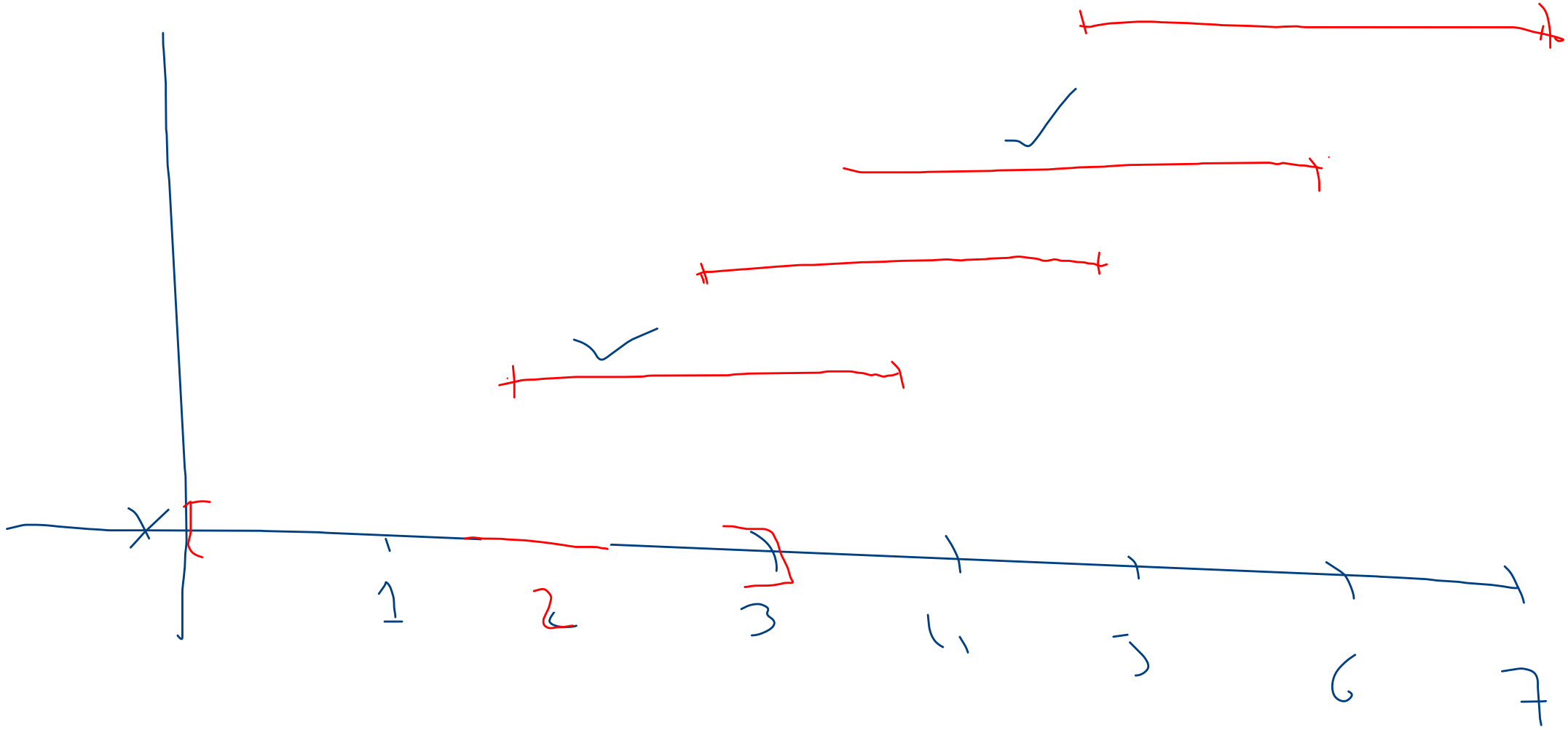
We have a set of n jobs or tasks for a machine. Each job j is associated with a start time s_j and an end time f_j. Thus each job is associated with an interval $[s_j, f_j]$

Clearly, if two intervals overlaps then the jobs cannot be scheduled. Such jobs are called 'incompatible'.

Our task is to schedule as many jobs as possible that are pairwise compatible.

Ex) $S = \{[1,3], [2,4], [3,5], [4,6], [5,7]\}$

The best solution is $\{[1,3), [3,5), [5,7]\}$.


1
2
3
4
5
6
7

Ex Call a job a candidate job if it is compatible with every already-scheduled jobs. If a candidate job is available then always add the candidate job with shortest finish time.

or Then always add the candidate job with earliest start time

or Then always add the candidate job with least no. of conflict with other candidate jobs.

Give counter example to show that in each case, it does not yield an optimal soln

We now show that early finish time yields an optimal soln

Algorithm EFT

Input A set $\mathcal{I} = \{I_1, \ldots, I_n\}$, where $I_j = [s_j, f_j]$, $s_j < f_j$ for $1 \le j \le n$

Output A maximum subset of pairwise compatible intervals.

1. Sort the input list $\mathcal{I}$ in increasing order of finish time.
2. $f = 0$
3. $S_0 = \phi$
4. for $i = 1$ to n do
5. if $f < s_i$ then
   $S_i \leftarrow S_{i-1} \cup \{I_i\}$
   $f \leftarrow f_i$
   end if
   end for

Thm Algorithm EFT gives an optimal soln in time $O(n \log n)$.

Pf. We shall prove by induction that, at every step

- if i jobs have already been scheduled by the algorithm, then these can be extended to an optimal soln without removing any of the already-scheduled jobs.

The base step is obvious

So assume that this invariant is true when i jobs have been scheduled by the algorithm.

Let S be an optimal set containing these i jobs.

Suppose the algorithm picks job j next

If $j \in S$ then we are done. So assume $j \notin S$.

Let j' be the job in S with earliest finish time which is not one of the i jobs.

Then clearly $\boxed{f_j \leq f_{j'}}$.

Also, j is compatible with the jobs in S that following j'

Hence $(S - \{j'\}) \cup \{j\}$ is an optimal soln that includes the i jobs as well as j

Hence this invariant is maintained throughout the alg & when it stops we must arrive at an optimal soln.

Remark. For each interval J_i intersecting Ω a vertex v_i

Give counter example to show that in each case, it does not yield an optimal soln

2 Minimum Spanning Tree

We shall give 2 greedy alg for MST; one due to Prim & the other due to Kruskal.

Defn Let $G = (V, E)$ be a connected graph. A subgraph (V, T) is called a spanning tree if T is a tree

Let w be a real-valued fn on E. T is a minimum spanning tree if $w(T) = \sum_{e \in T} w(e)$ is as small as possible.

Our strategy will be to pick an edge at each step & to ensure that the set of edges X already picked is part of a minimum spanning Tree. This is guaranteed by the following

Cut Property. Let $X \subseteq T$, where T is a MST. Let $S \subseteq V$. Suppose no edge in X crosses between S & $V-S$ i.e. there is no edge in X with one endpt in S & the other endpt in $V-S$

Of all the edges that cross between S & $V-S$, let e be the lightest edge. Then

$$X \cup \{e\} \subseteq T',$$ where T' is an MST.

Pf: If $e \in T$, then we are done. So assume $e \notin T$

Adding e to T creates a unique cycle.

Claim: There is an edge $e' \in T$ that crosses between S & $V-S$

As we traverse the cycle, in order to come back to the starting we need to cross S & $V-S$ again.

Remark: For each interval I_i introduce a vertex v_i. Two vertices are joined by an edge if the corr^{n} interval overlaps.

Computing a maximum subset of S consisting mutually consistent pair is the same as computing a maximum independent set in the graph.

Prim's Algorithm

In Prim's alg. the set X is a single tree & S is the set of vertices of the tree. Starting from an arb. root, at every step we add the lightest edge that crosses between S & V-S until we obtain an MST.

Let $T' = (T - \{e'\}) \cup \{e\}$

Clearly T' is a spanning tree

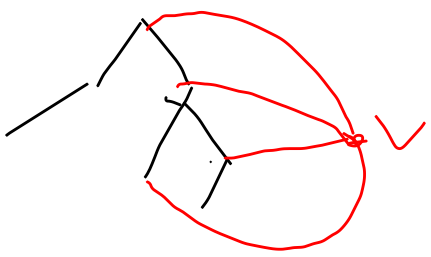
Also $w(e) \leq w(e')$

$w(T') \leq w(T)$ Since T is an MST

$w(T') = w(T)$ & $w(e) = w(e')$

Clearly $X \cup \{e\} \subseteq T'$

The key point in efficient implementation is to make easy the selection of the new to be added to X. The vertices of V−S reside in a priority queue with a key field. For each $v \in Q$, $k[v]$ is the min. wt of the edges joining v to some vertex in X. If there are no such edges, then $k[v] = \infty$.



$\pi(v)$ simply names the parent of v in the tree.

During the execution the set X is implicitly maintained as

$$X = \{ (v, \pi(v)) : v \in V - \{r\} - Q \}$$

When the alg terminates, $Q = \phi$ & the MST is

$$X = \{ (v, \pi(v)) : v \in V - \{r\} \}$$

Alg. Prim (G, w, r)

1. $Q \leftarrow V$
2. for each $u \in Q$
3. do $key[u] = \infty$
4. $key[r] = 0$
5. $\pi(r) = NIL$
6. while $Q \neq \phi$
7. — do $u \leftarrow$ EXTRACT-MIN(Q)
8. for each $v \in Adj(u)$
   do if $v \in Q$ & $w(u,v) < key[v]$
   then $\pi(v) = u$
   $key[v] = w(u,v)$