2 Interval Scheduling [KT 4.1]

We are given as n requests, where each request requires an interval of time. Say the i-th request needs to occupy (s_i, f_i) where s_i is the starting time, and f_i is the finishing time. We say to requests are compatible if their intervals do not overlap. Our goal is to accept as many compatible requests as possible. We first introduced a greedy framework where the high level idea is to myopically decide whether to accept each request using short-term evaluation metrics. Here, we make the "short-term" very short, i.e., as long as we can accept the request, we accept it.

General approach: Pick one, delete all the incompatible requests, and iterate. More preciously,

1. Sort the requests in some order (which will be decided later): $I_1, I_2, \ldots, I_n$
2. FOR $j = 1$ to n DO
3. IF I_j is compatible with the accepted requests THEN accept I_j

Failed greedy strategies:

(a) Select the earlier available request

(b) Select the shortest request

(c) Select the one with least #incompatible requests.

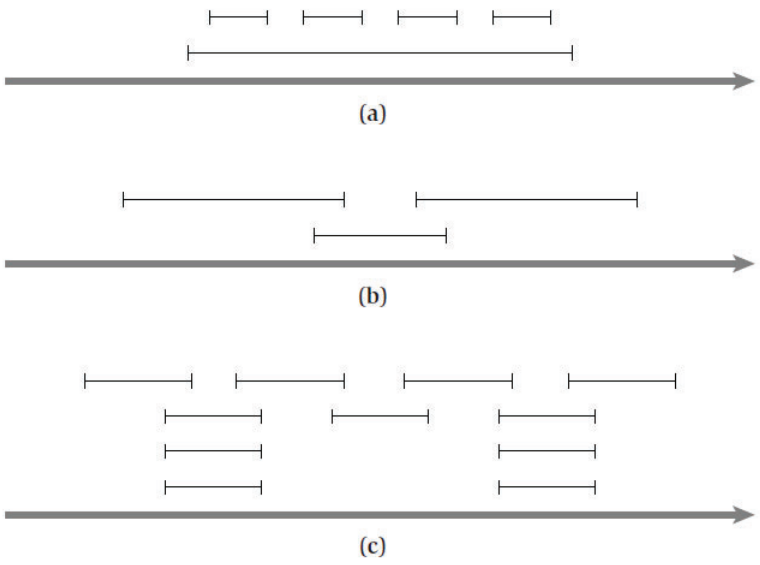

Figure 4.1 Some instances of the Interval Scheduling Problem on which natural greedy algorithms fail to find the optimal solution. In (a), it does not work to select the interval that starts earliest; in (b), it does not work to select the shortest interval; and in (c), it does not work to select the interval with the fewest conflicts.

Successful strategy: Select the first request that finishes first.

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

An example:

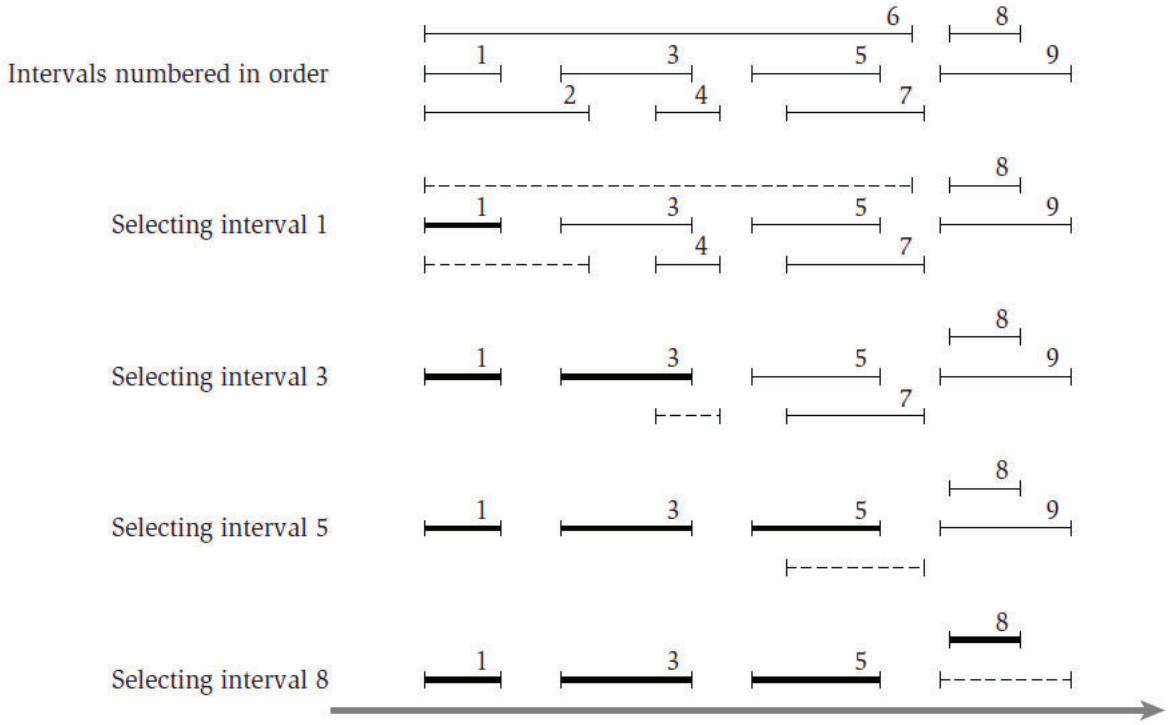

Figure 4.2 Sample run of the Interval Scheduling Algorithm. At each step the selected intervals are darker lines, and the intervals deleted at the corresponding step are indicated with dashed lines.

Proof of correctness: Cannot show that $A = O$ since there could be multiple O, but can show that $|A| = |O|$. Prove by contradiction+induction.

Proof: Suppose for contradiction that there exists an input instance where the greedy algorithm does not output the optimal answer. Let us assume that the greedy algorithm outputs intervals $i_1, \ldots, i_k$, and there exists an optimal solution with r requests where $r > k$.

Let us assume without loss of generality that the intervals $i_1, \ldots, i_k$ are sorted in increasing order with respect to time.

For each optimal solution $j_1, \ldots, j_r$ (also sorted in increasing order with respect to time), we know that there exists an index $q < k$ so that the first $q-1$ requests have the same finishing time as those of the greedy solution (i.e. $f_{i_1} = f_{j_1}, \ldots, f_{i_{q-1}} = f_{j_{q-1}}$), and $f_{i_q} \neq f_{j_q}$. Indeed, if such q does not exist, meaning that $f_{i_k} = f_{j_k}$, the greedy algorithm would have been able to select more requests than k (at least j_{k+1} is a choice).

Now among all optimal solutions, we pick the one $j_1, \ldots, j_r$ that such index q is the greatest. Now we have $f_{i_{q-1}} = f_{j_{q-1}}$ and $f_{i_q} \neq f_{j_q}$. Since the greedy algorithm always accepts a request with the smallest finishing time (while compatible with accepted ones), we know that $f_{i_q} < f_{j_q}$. We can verify that $j_1, \ldots, j_{q-1}, i_q, j_{q+1}, \ldots, j_r$ is also a valid optimal solution. However, in this optimal solution the first q requests have the same finishing time as the greedy solution. Therefore, the "q-index" of this new optimal solution must be greater than q. This contradicts to our assumption that the "q-index" of the optimal solution $j_1, \ldots, j_r$ was the greatest. □

Running time: $O(n \log n)$ for sorting. The rest can be done in $O(n)$ time by a single scan of the sorted intervals.

Homework 2 *If each request has weight, and we want to maximize the total weights of scheduled interval, can we use the aforementioned greedy algorithm? If yes give a proof; if no give an counterexample.*