CSE 331: Introduction to Algorithm Analysis and Design

Greedy Algorithms

1 Interval Scheduling: Maximize Intervals Scheduled

1.1 Problem

Input: A set of n intervals $I = \{i_0, i_1, \ldots, i_{n-1}\}$ such that each i_j is defined by its start time $s(i_j)$ and finish time $f(i_j)$ as the interval $[s(i_j), f(i_j))$. Note that the interval does not contain $f(i_j)$ so an interval can be scheduled with a start time equal to the previous tasks finish time.

Output: A schedule A of maximum size such that $A \subseteq I$ and no two intervals in A conflict.

1.2 Algorithm

1. Sort the intervals in I by increasing finish time
2. Initialize $A = \emptyset$
3. Iterate through the intervals in I
   (a) If the current interval does not conflict with any interval in A, add it to A
4. return A as the maximum set of scheduled intervals

1.3 Correctness

Greedy stays ahead: This is the first of two proofs techniques we will see for greedy algorithms. The idea is to prove that any point during execution, the algorithm has found the best possible solution on the sub-problem containing only the section of the input that has already been iterated over.

Proof. For this proof, we will assume we have an optimal solution $\mathcal{O} = \{j_0, j_1, \ldots, j_{m-1}\}$ and a solution from the algorithm $A = \{i_0, i_1, \ldots, i_{k-1}\}$. The optimal solution could be any arbitrary solution as long as it maximizes the number of intervals scheduled. We will prove that $|A| = |\mathcal{O}|$ which is equivalent to $k = m$.

We will assume that both A and $\mathcal{O}$ are sorted by the time of the intervals. Since there are no conflicts, sorting by start time or finish time will result in the same order. Now we want to prove that $f(i_r) \leq f(j_r)$ for all r by induction. This is the staying ahead part of the proof. We are proving that the greedy algorithm is always ahead (behind in this case) in terms of finish time. This can also be thought of as the greedy solution having more time remaining to schedule other tasks after the earliest $r + 1$ tasks are scheduled.

Base case: At $r = 0$, each algorithm has only a single task scheduled. Since the greedy algorithm chooses the task with the earliest finish time, we must have $f(i_0) \leq f(j_0)$. The interval i_0 must be

the interval with the earliest finish time so there is no possible interval that can be in $\mathscr{O}$ with an earlier finish time.

Induction step: We will assume $f(i_{r-1}) \leq f(j_{r-1})$ and prove $f(i_r) \leq f(j_r)$. We will assume $f(i_r) > f(j_r)$ and show that this leads to a contradiction. Since j_r is the next interval in the sorted schedule $\mathscr{O}$, we know that $s(j_r) \geq f(j_{r-1})$ and doesn't conflict any intervals in $\mathscr{O}$. Since $f(i_{r-1}) \leq f(j_{r-1})$, we know that j_r also doesn't conflict with any intervals already in the schedule A. Since j_r doesn't conflict with A and it has an earlier finish time than i_r, the algorithm would have chosen j_r over i_r. This means the algorithm didn't choose the earliest finish time interval with no conflicts which is a contradiction. Therefore, our assumption is false and $f(i_r) \leq f(j_r)$ as desired.

We have proven by induction that $f(i_r) \leq f(j_r)$ for all r, but this is not enough to prove the algorithms correctness. It is important to use this fact as a lemma to finish the proof. We have $|A| = k$ and $|\mathscr{O}| = m$ and we are proving that $k = m$. Since we assume that $\mathscr{O}$ is optimal, we can't have $k > m$, so we just need to show that $m \not> k$ which we will do by contradiction.

Assume that $m > k$. We know that at $r = k - 1$ (k intervals scheduled) that $f(i_{k-1}) \leq f(j_{k-1})$ and that $\mathscr{O}$ has at least one more interval in its schedule j_k. By the same reasoning as the induction step earlier in this proof, the greedy algorithm would add j_k to its schedule since it doesn't conflict with any previously scheduled algorithm. This implies that $|A| \neq k$ which is a contradiction. Our assumption must have been false and $k = m$. Since the greedy solution has the same number of intervals scheduled as an optimal solution, the greedy solution must be optimal.

□

1.4 Runtime

1. Sort the intervals in I by increasing finish time: Sorting n elements can be performed in $O(n \cdot log(n))$ time.
2. Initialize $A = \emptyset$: $O(1)$
3. Iterate through the intervals in I: This loop will iterate through all n intervals, visiting each exactly once for $O(n)$ total iterations.
   (a) If the current interval does not conflict with any interval in A, add it to A: By tracking the finish time of the most recently scheduled interval, this check can be done with one comparison of the previous finish time and the current intervals start time. This comparison takes $O(1)$ time.
4. return A as the maximum set of scheduled intervals: $O(1)$

This gives a total runtime of $O(n \cdot log(n)) + O(1) + O(n) \cdot O(1) + O(1)$ which is equivalent to $O(n \cdot log(n))$.