**CSE 331: Introduction to Algorithm Analysis and Design**

**Greedy Algorithms**

# 1 Weighted Interval Scheduling

## 1.1 Problem

**Input:** A set of $n$ intervals $I = \{i_1, i_2, \ldots, i_n\}$ such that each $i_j$ is defined by its start time $s(i_j)$, finish time $f(i_j)$ , and weight $v(i_j)$. 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 $A$ such that $A \subseteq I$, and no two intervals in $A$ conflict.

**Goal:** Maximize the sum of all interval weights in the output schedule $\sum_{i \in A} v(i)$.

## 1.2 Definitions

$p(j)$**:** The index of the interval with the latest end time that does not conflict with interval $i_j$ and has finish time earlier than $i_j$. Note that $p(j) < j$ since all intervals conflict with themselves. If no such interval exists, then $p(j) = 0$.

$OPT[j]$**:** The weight of the optimal schedule when only intervals $1, ..., j$ are considered.

## 1.3 Algorithm

1. Sort the intervals in $I$ by increasing finish time
2. compute $p(j)$ for all $j = 1, ..., n$
3. Initialize $OPT[0] = 0$
4. For $j = 1, ..., n$
   (a) Decide whether or not interval $j$ will be scheduled if only the first $j$ intervals are considered
   (b) $OPT[j] = max(v(i_j) + OPT[p(j)], OPT[j-1])$
5. Return the intervals scheduled to achieve $OPT[n]$

### 1.4 Correctness

We will prove the correctness by induction over $n$ to prove that this algorithm is correct for all instance of the weighted scheduling problem. The structure of this proof will mimic the structure of our algorithm.

*Proof.* We want to prove $OPT[n]$ is in fact the weight of the optimal schedule and therefore we've found the optimal schedule (which can be returned with some bookkeeping variables).

**Base case:** For $j = 0$ there are no intervals to consider. With an instance of the problem with no intervals there are no decisions to make and the optimal schedule has weight 0. Our algorithm correctly initializes $OPT[0] = 0$.

**Induction step:** We will use strong induction and assume that $OPT[j]$ is correct for all $0 \leq j \leq k$ and prove that our algorithm correctly computes $OPT[k+1]$. There are 2 different cases that can occur in any solution for $OPT[k+1]$: interval $i_{k+1}$ is in the schedule when only the first $k+1$ intervals are considered, or it is not.

If $i_{k+1}$ is in the schedule, then the schedule cannot contain any intervals $l > p(k+1)$ since they all conflict with $i_{k+1}$. So if $i_{k+1}$ is in the schedule we are left with a sub-problem where we only consider intervals $1, ..., p(k+1)$ the solution to which is $OPT[p(k+1)]$. Since $p(k+1) < k+1$ our algorithm is assumed to have computed the correct schedule for $OPT[p(k+1)]$. We then add this solution and $v(i_{k+1})$ to compute the largest possible weight schedule if only the first $k+1$ intervals are considered *and* interval $k+1$ is in the schedule.

If $i_{k+1}$ is not in the schedule the optimal schedule will be the same as it was when $i_{k+1}$ was not considered. That is $OPT[k+1] = OPT[k]$ and we assumed that our algorithm correctly computed $OPT[k]$.

We know what $OPT[k+1]$ would be if $i_{k+1}$ is in the schedule and if it's not. Since these are the only two options, the option that maximizes the weight of the schedule will be the correct value for $OPT[k+1]$. This is exactly what our algorithm computes and therefore our algorithm computes the correct value of $OPT[k+1]$

This proves by induction that our algorithm correctly computes $OPT[j]$ for all $j$, therefore it must compute the correct value for $OPT[n]$ which is the solution to the problem and is the return value of the algorithm.

□

### 1.5 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. compute $p(j)$ for all $j = 1, ..., n$: This can be computed in $O(n \cdot log(n))$ time by sorting the intervals by start time in a separate array and iterating through both arrays simultaneously (similar to merging in merge sort)

3. Initialize $OPT[0] = 0$: $O(1)$

4. For $j = 1, ..., n$: $O(n)$ iterations

   (a) Decide whether or not interval $j$ will be scheduled if only the first $j$ intervals are considered
   (b) $OPT[j] = max(v(i_j) + OPT[p(j)], OPT[j-1])$: $O(1)$

5. Return the intervals scheduled to achieve $OPT[n]$: $O(n)$ with some extra bookkeeping variables. Create an array of $n$ boolean values that remembers if interval $j$ was added or not when it was considered. Starting with $j = n$, if interval $j$ was added add it to the solution and next consider $p(j)$. If $j$ was not added, next consider $j - 1$.

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