1 A Greedy Algorithm for Scheduling Jobs with Deadlines and Profits

The setting is that we have \( n \) jobs, each of which takes unit time, and a processor on which we would like to schedule them in as profitable a manner as possible. Each job has a profit associated with it, as well as a deadline; if the job is not scheduled by the deadline, then we don’t get the profit. Because each job takes the same amount of time, we will think of a Schedule \( S \) as consisting of a sequence of job “slots” 1, 2, 3, \ldots where \( S(t) \) is the job scheduled in slot \( t \). (If one wishes, one can think of a job scheduled in slot \( t \) as beginning at time \( t – 1 \) and ending at time \( t \), but this is not really necessary.)

More formally, the input is a sequence \((d_1, g_1), (d_2, g_2), \ldots, (d_n, g_n)\) where \( g_i \) is a nonnegative real number representing the profit obtainable from job \( i \), and \( d_i \in \mathbb{N} \) is the deadline for job \( i \); it doesn’t hurt to assume that \( 1 \leq d_i \leq n \). (The reason why we can assume that every deadline is less than or equal to \( n \) is because even if some deadlines were bigger, every feasible schedule could be “contracted” so that no job was placed in a slot bigger than \( n \).)

Definition 1 A schedule \( S \) is an array: \( S(1), S(2), \ldots, S(n) \) where \( S(t) \in \{0, 1, 2, \ldots, n\} \) for each \( t \in \{1, 2, \ldots, n\} \).

The intuition is that \( S(t) \) is the job scheduled by \( S \) in slot \( t \); if \( S(t) = 0 \), this means that no job is scheduled in slot \( t \).

Definition 2 \( S \) is feasible if

(a) If \( S(t) = i > 0 \), then \( t \leq d_i \). (Every scheduled job meets its deadline)

(b) If \( t_1 \neq t_2 \) and \( S(t_1) \neq 0 \), then \( S(t_1) \neq S(t_2) \). (Each job is scheduled at most once.)

We define the profit of a feasible schedule \( S \) by \( P(S) = g_{S(1)} + g_{S(2)} + \ldots + g_{S(n)} \), where \( g_0 = 0 \) by definition.

Goal: Find a feasible schedule \( S \) whose profit \( P(S) \) is as large as possible; we call such a schedule optimal.

We shall consider the following greedy algorithm. This algorithm begins by sorting the jobs in order of decreasing (actually nonincreasing) profits. Then, starting with the empty schedule, it considers the jobs one at a time; if a job can be (feasibly) added, then it is added to the schedule in the latest possible (feasible) slot.
Greedy:
Sort the jobs so that: \( g_1 \geq g_2 \geq \ldots \geq g_n \)
for \( t : 1..n \)
\[
S(t) \leftarrow 0 \{\text{Initialize array } S(1), S(2), \ldots, S(n)\}
\]
end for
for \( i : 1..n \)
    Schedule job \( i \) in the latest possible free slot meeting its deadline;
    if there is no such slot, do not schedule \( i \).
end for

Example. Input of Greedy:

<table>
<thead>
<tr>
<th>Job ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline ( d_i )</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>(when job must finish by)</td>
</tr>
<tr>
<td>Profit ( g_i )</td>
<td>9</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>(already sorted in order of profits)</td>
</tr>
</tbody>
</table>

Initialize \( S(t) \):

<table>
<thead>
<tr>
<th>( t )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S(t) )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Apply Greedy: Job 1 is the most profitable, and we consider it first. After 4 iterations:

<table>
<thead>
<tr>
<th>( t )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S(t) )</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Job 3 is scheduled in slot 1 because its deadline \( t = 3 \), as well as slot \( t = 2 \), has already been filled.

\( P(S) = g_3 + g_2 + g_1 = 7 + 7 + 9 = 23 \).

**Theorem 1** The schedule output by the greedy algorithm is optimal, that is, it is feasible and the profit is as large as possible among all feasible solutions.

We will prove this using our standard method for proving correctness of greedy algorithms. We say feasible schedule \( S' \) extends feasible schedule \( S \) iff for all \( t \) \( 1 \leq t \leq n \), if \( S(t) \neq 0 \) then \( S'(t) = S(t) \).

**Definition 3** A feasible schedule is promising after stage \( i \) if it can be extended to an optimal feasible schedule by adding only jobs from \( \{i + 1, \ldots, n\} \).

**Lemma 1** For \( 0 \leq i \leq n \), let \( S_i \) be the value of \( S \) after \( i \) stages of the greedy algorithm, that is, after examining jobs \( 1, \ldots, i \). Then the following predicate \( P(i) \) holds for every \( i, 0 \leq i \leq n \):

\[
P(i) : S_i \text{ is promising after stage } i.
\]

This Lemma implies that the result of Greedy is optimal. This is because \( P(n) \) tells us that the result of Greedy can be extended to an optimal schedule using only jobs from \( \emptyset \). Therefore the result of Greedy must be an optimal schedule.
**Proof of Lemma:** To see that $P(0)$ holds, consider any optimal schedule $S_{opt}$. Clearly $S_{opt}$ extends the empty schedule, using only jobs from $\{1, \cdots, n\}$.

So let $0 \leq i < n$ and assume $P(i)$. We want to show $P(i+1)$. By assumption, $S_i$ can be extended to some optimal schedule $S_{opt}$ using only jobs from $\{i+1, \cdots, n\}$.

**Case 1:** Job $i+1$ cannot be scheduled, so $S_{i+1} = S_i$.
Since $S_{opt}$ extends $S_i$, we know that $S_{opt}$ does not schedule job $i+1$. So $S_{opt}$ extends $S_{i+1}$ using only jobs from $\{i+2, \cdots, n\}$.

**Case 2:** Job $i+1$ is scheduled by the algorithm, say at time $t_0$ (so $S_{i+1}(t_0) = i+1$ and $t_0$ is the latest free slot in $S_i$ that is $\leq d_{i+1}$).

**Subcase 2A:** Job $i+1$ occurs in $S_{opt}$ at some time $t_1$ (where $t_1$ may or may not be equal to $t_0$).
Then $t_1 \leq t_0$ (because $S_{opt}$ extends $S_i$ and $t_0$ is as large as possible) and $S_{opt}(t_1) = i+1 = S_{i+1}(t_0)$.

If $t_0 = t_1$ we are finished with this case, since then $S_{opt}$ extends $S_{i+1}$ using only jobs from $\{i+2, \cdots, n\}$. Otherwise, we have $t_1 < t_0$. Say that $S_{opt}(t_0) = j \neq i+1$. Form $S'_{opt}$ by interchanging the values in slots $t_1$ and $t_0$ in $S_{opt}$. Thus $S'_{opt}(t_1) = S_{opt}(t_0) = j$ and $S'_{opt}(t_0) = S_{opt}(t_1) = i+1$.

The new schedule $S'_{opt}$ is feasible (since if $j \neq 0$, we have moved job $j$ to an earlier slot), and $S'_{opt}$ extends $S_{i+1}$ using only jobs from $\{i+2, \cdots, n\}$. We also have $P(S_{opt}) = P(S'_{opt})$, and therefore $S'_{opt}$ is also optimal.

**Subcase 2B:** Job $i+1$ does not occur in $S_{opt}$.
Define a new schedule $S'_{opt}$ to be the same as $S_{opt}$ except for time $t_0$, where we define $S'_{opt}(t_0) = i+1$. Then $S'_{opt}$ is feasible and extends $S_{i+1}$ using only jobs from $\{i+2, \cdots, n\}$.

To finish the proof for this case, we must show that $S'_{opt}$ is optimal. If $S_{opt}(t_0) = 0$, then we have $P(S'_{opt}) = P(S_{opt}) + g_{i+1} \geq P(S_{opt})$. Since $S_{opt}$ is optimal, we must have $P(S'_{opt}) = P(S_{opt})$ and $S'_{opt}$ is optimal. So say that $S_{opt}(t_0) = j$, $j > 0$, $j \neq i+1$. Recall that $S_{opt}$ extends $S_i$ using only jobs from $\{i+1, \cdots, n\}$. So $j > i+1$, so $g_j \leq g_{i+1}$. We have $P(S'_{opt}) = P(S_{opt}) + g_{i+1} - g_j \geq P(S_{opt})$.

As above, this implies that $S'_{opt}$ is optimal. $\Box$

We still have to discuss the running time of the algorithm. The initial sorting can be done in time $O(n \log n)$, and the first loop takes time $O(n)$. It is not hard to implement each body of the second loop in time $O(n)$, so the total loop takes time $O(n^2)$. So the total algorithm runs in time $O(n^2)$.

Using a more sophisticated data structure one can reduce this running time to $O(n \log n)$, but in any case it is a polynomial-time algorithm.