CSC 373 H 1 Y — Summer 2007

University of Toronto — St. George Campus

Lecture Summary for Week 4

This summary is not a replacement for the lecture. If you miss a class, please arrange with a friend to take note for you.

2.5 Weighted Interval Scheduling [Section 6.1]

There are n jobs where job i has starting time $s(i)$, finishing time $f(i)$ and profit/weight $w(i)$ (for $1 \leq i \leq n$). Here $s(i), f(i), w(i) \in \mathbb{N}$ and $s(i) < f(i)$. There is a processor that can process only one job at a time. The problem is to select (or schedule) a subset of the jobs to execute in order to maximize the total profit. The selected job can not overlap.

Input n triples $(s(i), f(i), w(i))$ where $s(i), f(i), w(i) \in \mathbb{N}$, $s(i) < f(i)$, for $1 \leq i \leq n$.
Output A subset $S \subseteq \{1, 2, \ldots, n\}$ so that for any $i, j \in S$:

$$\text{either } f(i) \leq s(j) \qquad \text{or} \qquad f(j) \leq s(i)$$

and $\sum_{i \in S} w(i)$ is maximum.

(The name "Interval Scheduling" comes from the fact that each job i can be considered as an interval $[s(i), f(i)]$.)

The first attempt might be to consider the subproblems of scheduling the first i jobs in a time interval $[t_1, t_2]$. For this approach, we can define an array A (of size $n \times t \times t$ where $t = max\{f(1), f(2), \ldots, f(n)\} - \min\{s(1), s(2), \ldots, s(n)\}$) where $A[i, t_1, t_2]$ is the best profit of scheduling the jobs $\{1, 2, \ldots, i\}$ in the time interval $[t_1, t_2]$. If job i can be scheduled in $[t_1, t_2]$ (i.e., $t_1 \leq s(i)$ and $f(i) \leq t_2$), then

$$A[i, t_1, t_2] = max\{w(i) + A[i-1, t_1, s(i)] + A[i-1, f(i), t_2], A[i-1, t_1, t_2]\} \tag{1}$$

Otherwise, $A[i, t_1, t_2] = A[i-1, t_1, t_2]$. We can go on and write the program that computes A and a program to compute an optimal solution from A.

We can do better by noticing that for a given set of jobs $\{1, 2, \ldots, i\}$, we already know that they can be scheduled only in the time interval between $\min\{s(1), s(2), \ldots, s(i)\}$ and $max\{f(1), f(2), \ldots, f(i)\}$. As a result, suppose that

$$f(i) = max\{f(1), f(2), \ldots, f(i)\}$$

Then the term $A[i-1, f(i), t_2]$ in (1) is 0. The parameters t_1, t_2 used in this approach for defining subproblems are actually redundant.

Thus we will first sort the jobs in non-decreasing order of their finishing time:

$$f(1) \leq f(2) \leq \ldots \leq f(n)$$

Then, let $M[i]$ be the maximum profit of scheduling the first i jobs $\{1, 2, \ldots, i\}$ (for $1 \leq i \leq n$). The recurrence for $M[i]$ is as follows: If i is not in an optimal set for i jobs then

$$M[i] = M[i-1]$$

otherwise

$$M[i] = w(i) + M[j]$$

where j is the largest index so that $f(j) \leq s(i)$ (i.e., all jobs $j+1, j+2, \ldots, i-1$ overlap with job i).

For each i, we need to compute such value $j = p[i]$. This can be done by binary search, which take time $\mathcal{O}(\log(n))$ for each i, and thus time $\mathcal{O}(n\log(n))$ in total. (THE PROGRAM GIVEN IN CLASS FOR COMPUTING $p[1], p[2], \ldots, p[n]$ RUNS IN TIME $\mathcal{O}(n^2)$.)

The initial value and recurrence for M are:

$$M[0] = 0, \qquad M[i] = max\{M[i-1], w(i) + M[p[i]]\} \text{ for } 1 \leq i \leq n$$

The program for computing M:

1. $M[0] \leftarrow 0$
2. For $i = 1$ to n do
3. $\quad M[i] \leftarrow max\{M[i-1], w(i) + M[p[i]]\}$
4. End For

Finally, an optimal subset S can be computed from M as follows:

1. $S \leftarrow \varnothing$ % solution
2. $i \leftarrow n$
3. While $i \leq 1$ do
4. $\quad$ If $M[i] = M[i-1]$ then $i \leftarrow i - 1$
5. $\quad$ Else
6. $\qquad S \leftarrow S \cup \{i\}$
7. $\qquad i \leftarrow p[i]$
8. $\quad$ End If
9. End While
10. Return S.

Running time: Sorting the jobs and computing $p[1], p[2], \ldots, p[n]$ take time $\mathcal{O}(n\log(n))$ each. Computing M and computing an optimal solution from M both take time $\mathcal{O}(n)$. So the above algorithm runs in time $\mathcal{O}(n\log(n))$.

2.6 Job Scheduling with Deadlines, Durations and Profits

This problem is slightly different from the previous: the jobs have a deadline and duration instead of the starting time and finishing time. More precisely, each job i has deadline $d(i)$, duration $\ell(i)$ and profit $w(i)$ ($d(i), \ell(i), w(i) \in \mathbb{N}$). We want a schedule with maximum total profit. Here a schedule S is an array of length n, where

$$S[i] = \begin{cases} -1 & \text{if job } i \text{ is not scheduled} \\ t \geq 0 & \text{if job } i \text{ is scheduled to run at time } t \end{cases}$$

A schedule S is feasible if all jobs that are scheduled meet their deadlines, and there are no overlapping jobs:

- For $1 \leq i \leq n$: if $S[i] \geq 0$ then $S[i] + \ell(i) \leq d(i)$

- For $1 \le i < j \le n$: if $S[i] \ge 0$ and $S[j] \ge 0$, then

$$\text{either } S[i] + \ell(i) \le S[j] \qquad \text{or} \qquad S[j] + \ell(j) \le S[i]$$

Input n triples $(d(i), \ell(i), w(i))$ for $1 \le i \le n$, where $d(i), \ell(i), w(i) \in \mathbb{N}$.
Output A feasible schedule with maximum total profit.

Here we will sort the jobs in the non-decreasing order of their deadlines:

$$d(1) \le d(2) \le \ldots \le d(n)$$

A difficulty in solving this problem recursively is that a job i can start any time as long as it meets its deadline. The following lemma is useful: It shows that when the jobs are sorted by their deadlines (in non-decreasing order), then in an optimal schedule we can choose to run the scheduled jobs as late as possible.

Lemma: Suppose that

$$d(1) \le d(2) \le \ldots \le d(i)$$

and S is a feasible schedule for the jobs $\{1, 2, \ldots, i\}$ where i is scheduled. Suppose that all jobs finish by time $t \le d(i)$. Then there is a feasible schedule S' that schedules the same jobs as S, where job i is the last to run and finishes by time t.

Proof: Simply modify S by moving job i to start at time $t - \ell(i)$, and schedule all jobs in S that start after job i $\ell(i)$ earlier. □

Corollary: Suppose that

$$d(1) \le d(2) \le \ldots \le d(n)$$

Then there is an optimal schedule OPT that schedules the jobs in the order of their numbers (i.e., if $i < j$ are two jobs in OPT, then i start befores j).

Proof: The proof can be done by induction. Alternatively, we can proceed as follows.

Let OPT be an optimal schedule with the smallest number of "inversions", i.e., pairs i, j where $i < j$ and j starts before i. We show that there must be no inversions in OPT. The proof is by contradiction.

Suppose by way of contradiction that there is an inversion in OPT. Let m be the largest job in OPT that involves in an inversion. So there is a job j in OPT such that $j < m$ and j starts after m. Suppose that j finishes at time t.

Notice that if i is a job in OPT that finishes before t, then $i < m$ (otherwise $i > m$ and i involves in an inversion i, j, contradicts the choice of m). Therefore by the Lemma we can modify OPT so that m starts after j and m finishes by time t. The result is also an optimal schedule, but with less number of inversions. Contradiction to the choice of OPT. □

Our algorithm below will only look for an optimal schedule that satisfies the conclusion of the Corollary. The four steps of the dynamic programming algorithm is as follows:

1. Let A be an $n \times d(n)$ array, $A[i, d]$ is the optimal total profit of scheduling the jobs $\{1, 2, \ldots, i\}$ to finish before time d. Here $0 \le i \le n, 0 \le d \le d(n)$.

2. Initialization:

$$A[0, d] = 0 \qquad \text{for } 0 \le d \le d(n)$$

For the recursion, if it is possible to schedule i (i.e., $d \le \ell(i)$) then the optimal profit of schedule job i is

$$w(i) + A[i-1, \min\{d, d(i)\} - \ell(i)]$$

In other words,

$$A[i, d] = \begin{cases} A[i-1, d] & \text{if } d < \ell(i) \\ max\{A[i-1, d], w(i) + A[i-1, \min\{d, d(i)\} - \ell(i)] & \text{otherwise} \end{cases}$$

3. Program for computing A:

```
For d = 0 to d(n)   do   A[0, d] ← 0   End For
For i = 1 to n do
    For d = 0 to d(n) do
        If d < t(i) then   A[i, d] ← A[i − 1, d]
        Else A[i, d] ← max{A[i − 1, d], w(i) + A[i − 1, min{d, d(i)} − ℓ(i)]
        End If
    End For
End For
```

- Computing an optimal schedule S. Recall that if job i is not scheduled then $S[i] = -1$, otherwise $S[i]$ is the starting time for job i.

```
S: an array of length n.
i ← n,  d ← d(n)
While e > 0 do
    If A[i, d] ≠ A[i − 1, d]
        S[i] ← min{d, d(i)} − ℓ(i)
        d ← min{d, d(i)} − ℓ(i)
    Else S[i] ← −1
    End If
    i ← i − 1
End While
```