Greedy Algorithms

ECE 1762 Algorithms and Data Structures
Fall Semester, 2007 — U of Toronto

1 Scheduling with Deadlines

We have a set of n jobs to execute, each of which takes unit time. At any time $t = 1, 2, 3, \ldots$ we can execute exactly one job. Job i, $1 \leq i \leq n$, earns us a profit g_i , if and only if it is executed no later than time d_i. For example, with $n = 4$ and the following values:

i	1	2	3	4
g_i	50	10	15	30
d_i	2	1	2	1

the schedules to consider and the corresponding profits are:

Sequence	profit	
1	50	
2	10	
3	15	
4	30	
1,3	65	
2,1	60	
2,3	25	
3,1	65	
4,1	80	←(optimum)
4,3	45	

The sequence $(3, 2)$, for instance, is not considered because job 2 would be executed at time $t = 2$, that is, after its deadline $d_2 = 1$. To maximize our profit in this example, we should execute the schedule (4,1).

A set of jobs is *feasible* if there exists at least one sequence (also called feasible) that allows all the jobs in set to be executed in time for their respective deadlines. An obvious *greedy algorithm* consists of constructing the schedule step by step, adding at each step the job with the highest value of g_i among those not yet considered, provided that the chosen set of jobs remains feasible. In the proceeding example we first choose job 1. Next, we choose job 4: the set $\{1, 4\}$ is feasible because it can be executed in the order $(4, 1)$. Next we try the set $\{1, 3, 4\}$, which turns out not to be feasible; job 3 is therefore rejected. Finally we try $\{1, 2, 4\}$, which is also infeasible; so job 2 is also rejected. Our solution, optimal in this case, is therefore to execute the set of jobs $\{1, 4\}$, which in fact can only be done in the order $(4, 1)$. It remains to prove that this algorithm always finds an optimal schedule and to find an efficient way of implementing it.

Let J be a set of k jobs. At first glance it seems we might have to try all the $k!$ possible permutations of these jobs to see whether J is feasible. Happily, this in not the case:

Lemma: Let J be a set of k jobs, and let $S = (s_1, s_2, s_3, s_4, \ldots, s_k)$ be a permutation of these jobs such that $d_{s1} \leq d_{s2} \leq \ldots \leq d_{sk}$. Then the set J is feasible if and only if the sequence of S is feasible.

Proof: The "if" is obvious. For the "only if" : If J is feasible, there exists at least one sequence of jobs $R = (r_1, r_2, \ldots, r_k)$ such that $d_{r_i} \geq$ i, 1≤ i ≤k. Suppose S $\neq R$. let a be the smallest index such that $s_a \neq r_a$, and let b be defined by $r_b = s_a$; It is clear that $b > a$. Also $d_{r_a} \geq d_{s_a}$ (by the construction of S and the minimality of a) $= d_{r_b}$, (by the definition of b).

The job r_a could therefore be executed later, at the time when r_b is at present scheduled. Since r_b can certainly be executed earlier than planned, we can interchanged the item r_a and r_b in R. The result is a new feasible sequence, which is the same as S as least in positions $1, 2, \ldots, a$. Continuing this, we obtain a series of feasible sequences, each having at least one more position in agreement with S. Finally after a maximum of $K - 1$ steps of this type, we obtain S itself, which is therefore feasible.

This shows that it suffices to check a single sequence, in order of increasing deadlines, to know whether a set of jobs is or is not feasible. We now prove that the greedy algorithm always finds an optimal schedule.

Proof of optimality: suppose that the greedy algorithm chooses to execute a set of jobs I whereas in fact the set $J \neq I$ is optimal. Consider two feasible sequences S_I and S_J, for the two sets of jobs in question. By making appropriate interchanges of jobs in S_I and S_J, we can obtain two feasible sequences $S_I^{'}$ and $S_J^{'}$ such that every job common to both I and J is scheduled for execution at the same time in the two sequences. (We may have to leave gaps in the schedule.) The necessary interchanges are easily found if we scan the two sequences S_I and S_J from right to left (Figure 1). $S_I^{'}$ and $S_J^{'}$ are distinct sequences since I $\neq$ J. Let us consider an arbitrary time when the task scheduled in $S_I^{'}$ is different from that scheduled in $S_J^{'}$.

- If some task a is scheduled in $S_I^{'}$ whereas there is a gap in $S_J^{'}$ (and therefore task a doesn't belong to J), the set $J \bigcup \{a\}$ is feasible and would be more profitable that J. This is not possible since J is optimal by assumption.
- If some task b is scheduled in $S_J^{'}$ whereas there is a gap in $S_I^{'}$, the set $I \bigcup \{b\}$ is feasible hence the greedy algorithm should have included b in I. This is also impossible since it did not do so.

  The only remaining possibility is that some task a is schedules in $S_I^{'}$ whereas different task b is scheduled in $S_J^{'}$. Again this implies that a doesn't appear in J and the b doesn't appear in I.

  - If $g_a > g_b$, one could substitute a for b in J and improve it. This goes against the optimality of J.
  - If $g_a < g_b$, the greedy algorithm should have chosen b before even considering a since $(I \backslash \{a\}) \bigcup \{b\}$ would be feasible. This is not possible either since it didn't include b in I.
  - The only remaining possibility is therefore that $g_a = g_b$.

In conclusion , for each time slot, sequences $S_I^{'}$ and $S_J^{'}$ either schedule no tasks, the same task, or two distinct tasks yielding the same profit. This implies that the total worth of I is identical with that of the optimal set J, and thus I is optimal as well.

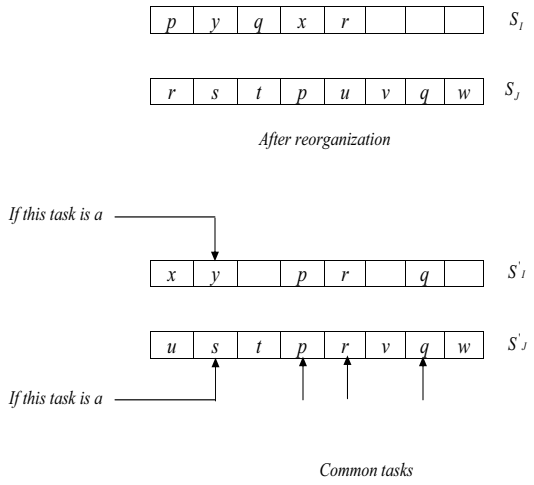

Figure 1: Rearranging schedules to bring identical tasks together