More Greed

Greedy Algorithm for Activity Selection

Suppose we have a set of activities to choose from, but some of them overlap. We want to choose the largest possible set of non-overlapping activities. In class I set up a scenario involving the imaginary School of Computing International Film Festival, with movies starting and ending at odd times. I'll spare you the details here and just refer to generic "activities".

A greedy algorithm always starts by sorting the objects. In class we experimented with different criteria for sorting:

1. Sort the activities by start time. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

This algorithm fails on this example

Task	1	2	3
Start Time	8:00	8:01	8:05
Finish Time	9:00	8:02	8:06

2. Sort the activities by length, shortest first. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

This algorithm fails on this example

Task	1	2	3
Start Time	8:58	8:55	9:01
Finish Time	9:02	8:59	9:05

3. Sort the activities by length, longest first. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen. (I didn't even mention this idea in class – it seems silly – and it is.)

This algorithm fails on this example

Task	1	2	3
Start Time	8:00	8:01	8:05
Finish Time	9:00	8:02	8:06

None of these work ... but undaunted by our repeated failures, we finally hit upon another alternative:

4. Sort the activities by **finish time**, earliest first. Then repeatedly choose the next activity if it doesn't overlap with the ones already chosen.

Proof of correctness:

We will use proof by induction, applying induction to n, the number of activities.

Base case: Let n = 1. Clearly the algorithm will choose the only activity, thus finding the optimal solution.

Assume the algorithm always finds the optimal solution when there are $\leq$ n activities, where n is some integer ≥ 1.

Let the number of activities be n+1. Apply the algorithm (sort by finish time, and select as above). Let the Greedy Algorithm solution be $A = \{a_1, a_2, \ldots a_t\}$. Note that a_1 must have the earliest of all finish times, due to the sort.

Step 1: We need to show that there is an optimal solution containing a_1. Let O be any optimal solution, with its elements sorted by finish time. Let O = $\{o_1, o_2, \ldots o_s\}$ We know that start_time(o_2) $\geq$ finish_time(o_1) $\geq$ finish_time(a_1) because a_1 has the earliest finish time of all the activities. Thus O* = $\{a_1, o_2, \ldots o_s\}$ is a feasible solution, and |O*| = |O|, so O* is an

optimal solution that contains a_1.

Step 2: Now we need to show that A is optimal. By the inductive assumption, $\{a_2, a_3, \ldots a_t\}$ is an optimal (ie largest) solution to the problem of choosing activities that start no earlier than finish_time(a_1).

Let O* be the optimal solution we constructed above, so O* = $\{a_1, o_2, \ldots o_s\}$.

Observe that $\{o_2, o_3, \ldots o_s\}$ is also a solution to the problem of choosing activities that start no earlier than finish_time(a_1).

Because $\{a_2, a_3, \ldots a_t\}$ is an **optimal** solution to this reduced problem, we know

$|\{a_2, a_3, \ldots a_t\}| \geq |\{o_2, o_3, \ldots o_s\}|$

Thus $|A| \geq |O^*|$

Since O* is an optimal solution, |A| > |O*| is not possible, so $|A| = |O^*|$. Thus A is an optimal solution, and the Greedy Algorithm finds the optimal solution whenever there are n+1 activities.

Therefore the Greedy Algorithm finds the optimal solution to the Activity Selection Problem for all sets of activities.

In class I stressed the point that even though in our proof of correctness for Greedy Algorithms we say "Let O be an optimal solution", our proof does not need to actually construct O – we just make use of properties that all optimal solutions must have in common.

Of course the algorithm simply constructs the set A – it never constructs some other solution O and swaps elements of A and O – that is just part of the proof that A is an optimal solution.

Here's another idea for sorting the activities: sort the activities in order of the number of "overlaps" (so an activity that overlaps with two others would be sorted in front of one that overlaps with three others). Now apply the Greedy algorithm exactly as described above (really, all Greedy algorithms look alike): work through the sorted list and add each activity to the chosen set iff it doesn't conflict with the ones already chosen.

Question: Can you prove that this sorted order always leads to an optimal solution, or can you find a set of activities where this approach fails to find an optimal solution?