Solutions -- Topic 13, Greedy Algorithms

Activity Scheduling

In the activity scheduling problem, we are given the start and finish times s_i and f_i of a set S of candidate activities, and possibly other data about them. We know all the activities in advance: this is batch scheduling, not real time. We need to **select (schedule) a subset A of the activities that are *compatible* (do not overlap with each other)**. There are different versions of the problem depending on what we maximize.

Maximizing Count

In this version of the problem we ***find the largest possible set (maximum <u>count</u>,*** not duration) *of activities that do not overlap with each other*. CLRS shows that this problem has the ***greedy choice property***: a globally optimal solution can be assembled from locally optimal choices. They show it by example with this *greedy strategy*: Always select the remaining compatible activity that **ends first**, and then solve the subproblem of scheduling the activities that start after this activity.

There are other greedy strategies, but some work and some don't. In the following two problems (and the extra credit problem) you determine whether alternative locally greedy strategies lead to globally optimal solutions. For each strategy below,

- **either show that the strategy leads to an optimal solution** by outlining the algorithm or approach (be sure to show it works on *any* input, not just the example you drew),
- **or give a counterexample**: write down a set of activities (defined by their start and finish times) that the strategy fails on, and show what goes wrong.

1. *Greedy strategy*: Always select the remaining compatible activity with the **earliest start time**. *Proposed Rationale:* Don’t waste any time getting started.

Solution (1 pt): Counterexample:

```
|---------------------------|
     |---|   |---|   |---|
```

Or as one group of students put it:

```
|-----------------------------------Reddit---------------------------------------|
   |-----homework-----|     |----class----|       |---exercise---| |-----work-----| |-sleep-|
```

2. *Greedy strategy*: Always select the remaining compatible activity that has the **latest start time**. *Proposed Rationale:* Leave the most time remaining at the beginning for other activities.

Solution (2 pts, divided depending on approach taken, e.g., 1 for basic insight that it is the reverse form of the CLRS example and 1 for some demonstration that/how it works): This is the same as the greedy approach “ends first”, but chronologically reversed in decreasing start times. Therefore we can modify the proof and the algorithm to work from the upper end of the activities' times sorted in decreasing order by start times, rather than from the lower end sorted in increasing order by finish times. Note: the inequality needs to be changed as well.

Algorithm recursive version: changes in yellow. We make a fictional activity a_0 with $s[0] = \infty$.

```
GreedyActivitySelect(s,f)
  SortByDecreasingStartTimes(s,f)
  n = s.length
  return RecursiveActivitySelector(s,f,0,n)

RecursiveActivitySelector(s,f,k,n)
  m = k + 1
// find the latest starting activity in Sk
  while (m ≤ n) and (f[m] > s[k]) // finishes too late
     m = m + 1
  if m <= n
     return {$a_m$} ∪ RecursiveActivitySelector(s,f,m,n)
   else return ∅
```

Algorithm iterative version:

```
GreedyActivitySelect(s,f)
  SortByDecreasingStartTimes(s,f)
  n = s.length
  A = {$a_1$}
  k = 1
  for m = 2 to n
     if f[m] ≤ s[k] // finishes on time
```

```
            A ∪ { $a_m$ }
            k = m
    return A
```

Different tables may take different approaches in attempting this: we'll give credit for showing they had the basic insight <u>and</u> tried to show it works.

Students are not required to do this, but a proof can be constructed following the proof in CLRS or Topic 13 by changing "earliest finish" to "latest start" and changing the comparison:

> ***Theorem:*** If S_k is nonempty and a_m has the ~~earliest finish~~ latest start time in S_k, then a_m is included in some optimal solution.
>
> *Proof:* Let A_k be an optimal solution to S_k, and let $a_j \in A_k$ have the ~~earliest finish~~ latest start time in A_k. If $a_j = a_m$ we are done. Otherwise, let $A'_k = (A_k - \{a_j\}) \cup \{a_m\}$ (substitute a_m for a_j).
>
>> *Claim:* Activities in A'_k are disjoint.
>>
>> *Proof of Claim:* Activities in A_k are disjoint because it was a solution.
>>
>> Since a_j is the ~~first~~ last activity in A_k to ~~finish~~ start, and ~~$f_m \le f_j$~~ $s_m \ge s_j$ (a_m is the ~~earliest~~ latest in S_k), a_m cannot overlap with any other activities in A'_k
>>
>> No other changes were made to A_k, so A'_k must consist of disjoint activities.
>
> Since $|A'_k| = |A_k|$ we can conclude that A'_k is also an optimal solution to S_k, and it includes a_m. QED

Maximizing Value

We now consider a variation of the problem. Suppose that different activities earn different amounts of revenue. In addition to their start and finish times s_i and f_i, each activity a_i has <u>revenue</u> r_i, and our objective is now to **maximize the total revenue:**

$$\sum_{a_i \in A} r_i$$

3. Can you identify a greedy strategy that shows that this problem exhibits the greedy choice property? If so, show how it works to maximize value. If not, show counterexamples for the strategies you considered and identify an alternative problem solving strategy that would work on this problem.

Solution (2 points: 1 for what won't work and 1 for what will): a greedy algorithm won't work. Counterexamples for the above strategies all apply (they all ignore the value). Counter examples also exist for value-optimizing methods:

Choose highest value first:

```
            |--100--|
|-------99-------|     |-------99-------|
```

Choose highest density of value per unit time first:

Above is also counterexample for this

This is similar to the 0-1 knapsack problem! **Greedy won't work.**
⇒ Use dynamic programming instead. (Students need not develop the solution.)

Challenge Problems: More Strategies for Maximizing Count

As you did in #1 and #2,

- either show that strategy below leads to an optimal solution
- or give a counterexample.

4. *Greedy strategy*: Always select the remaining compatible activity that has the **least duration**. *Proposed Rationale:* Leave the most time remaining for other activities.

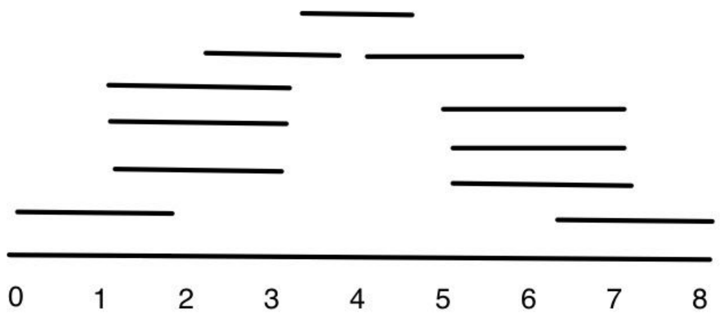
Counterexample (1 pt):

```
           |-------|
|----------------|    |----------------|
```

Perhaps that did not work because we don't care about duration, we care about how many other options are eliminated. So let's try …

5. *Greedy strategy*: Always select the remaining compatible activity that **overlaps with the fewest number of remaining activities**. *Proposed Rationale:* Eliminate the fewest number of remaining activities from consideration.

Counterexample (1 pt):



Middle activity will be chosen first, forcing choice of only one activity on each side. However, it is possible to get a sequence of 4 activities.