Recitation 5 - Greedy Algorithms

Sebastian Laudenschlager

sebastian.laudenschlager@colorado.edu

February 16, 2018

Greed ... is good?

- When facing an optimization problem, we usually have local choices to make.
- Greedy algorithms will make locally optimal choices in order to form a globally optimal solution.
- Greedy algorithms are usually simple algorithms.
  - Choose local optimum without regard to global optimum.
  - Greediness is simple: always choose what looks best at the moment.
- Important: Greediness doesn't always work.
  - Sometimes greedy algorithms lead to suboptimal solutions.

Sample problem

- Consider scheduling competing activities exclusively using some resource.
- Have a set of n possible activities $S = \{a_1, a_2, \ldots, a_n\}$.
- The resource can only serve one activity at a time, e.g. a lecture hall.
- Goal: maximize the number of (nonoverlapping) activities.
- Assume we have a sorted list of activity finishing times, i.e. $f_1 \leq f_2 \leq \ldots \leq f_n$.
- A particular activity a_i needs the resource during a time interval $[s_i, f_i)$. Here s_i and f_i are the start and finish times of activity a_i, respectively.

Example

- Solution to this problem may not be unique.
- Let's get greedy. How?
- Since we want to maximize the number of activities, intuitively, at any given point, we would want to choose an activity that leaves the resource available to as many other activities as possible.
- So if we pick the first activity to be the one that finishes first, that should leave the resource available to as many activities as possible.
- Since we ordered the activities by finishing time, the first choice should be a_1.

Subproblems

- If we choose a_1 first, that leaves us with the subproblem involving activities which start after a_1 finishes.
- Let $S_k = \{a_i \in S : s_i \geq f_k\}$ represent the activities that start after activity a_k finishes.
- So our choice of a_1 leaves us with solving the subproblem S_1.
- We want this greedy choice to form subproblems with optimal substructure, i.e. optimally solving all the subproblems optimally solves the entire problem.

Optimal Substructure

- Let S_{ij} be the set of activities that start after a_i finishes and that finish before a_j starts.
- Want to find a maximum set of (mutually compatible) activities in S_{ij}, call it A_{ij}.
- Let $a_k \in A_{ij}$ be some activity.
- Since a_k is part of the optimal solution, we are left with two subproblems, namely S_{ik} and S_{kj}.
- Let $A_{ik} = A_{ij} \cap S_{ik}$ be the activities in A_{ij} that finish before a_k starts, and let $A_{kj} = A_{ij} \cap S_{kj}$ be the activities in A_{ij} that start after a_k finishes.
- Then write $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$.

Optimal Substructure, cont.

- Two things now become apparent:
  - $|A_{ij}| = |A_{ik}| + 1 + |A_{kj}|$.
  - The optimal solution A_{ij} contains the optimal solutions to the subproblems S_{ik} and S_{kj}.
- Suppose we could find a more optimal solution than A_{ik} in the subproblem S_{ik}, call it A'_{ik}, such that $|A'_{ik}| > |A_{ik}|$.
- Then our solution set would be of size $|A'_{ik}| + 1 + |A_{kj}| > |A_{ik}| + 1 + |A_{kj}| = |A_{ij}|$.
- Contradiction, since A_{ij} was assumed to be the optimal solution.
- Copypasta for subproblem S_{kj}.
- So our greedy choice results in optimal substructure.

Optimality of greediness?

- Go back to our original first greedy choice of a_1, which left us with the subproblem S_1.
- Now that we know that our problem exhibits optimal substructure, we can conclude that if a_1 is in the optimal solution, then the entire optimal solution consists of a_1 and the solution to S_1.
- Great. But how do we know that a_1 is part of the optimal solution?

Greedy choice becomes optimal

- If S_k is any nonempty subproblem and a_m is the activity in S_k with the earliest finish time, then a_m is included in the optimal solution (maximum size subset of mutually compatible activities) of S_k.

Proof

- Let A_k be an optimal solution for S_k, and let a_j be the activity in A_k with the earliest finish time.
- Two cases:
  - If $a_j = a_m$ (greedy choice), then a_m is part of the optimal solution A_k.
  - If $a_j \neq a_m$ (optimal solution does not start with greedy choice), let $A'_k = (A_k \setminus \{a_j\}) \cup \{a_m\}$ be A_k where a_m replaces a_j.
- Want to show that A'_k is another optimal solution, starting with a_m.
- Note that the activities in A'_k are disjoint, since activities in A_k are disjoint (by definition), and $f_m \leq f_j$.
- Since activities in A'_k are disjoint, $|A'_k| = |A_k|$, which means A'_k is also optimal, and contains a_m.

Pseudocode

```
def greedyActivitySelector(s, f):
    n = len(s)
    A = {a_1}
    k = 1
    for m = 2:n
        if s[m] >= f[k]:
            A = A ∪ {a_m}
            k = m
    return A
```

Pseudocode explanation

- s and f are arrays containing start and finish times, respectively.
  - Remember arrays are sorted by finish time.
- k represents the index of the most recently added activity.
- Initialize A to contain the first activity.
- Loop through activities a_m and add the first activity that is compatible with the previously selected one.
- This will be the first one to finish, due to sorted finishing order.
- Set k to m to denote that a_m was added to A.
- Easy to see that this algorithm runs in $\Theta(n)$ time.