CS 255, Spring 2014, SJSU

Greedy algorithms

Fernando Lobo

Greedy Algorithms

- Another algorithm design technique.
- An algorithm is *greedy* when at each step of its execution it takes what seems to be the best option according to some local criterion.
- Used to solve many optimization problems (when we need to minimize or maximize something).

Greedy Algorithms

- The greedy strategy does not always work.
- It's possible to have several greedy algorithms for the same problem.
- It's easy to come up with a greedy algorithm.
- The difficult thing it to get a greedy algorithm that solves the problem to optimality.

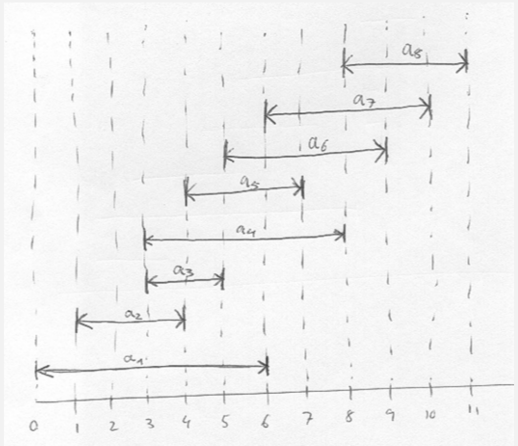
Example: Activity Selection / Interval Scheduling

Problem definition:

- Input: n activities (a_1, a_2, ..., a_n). Each a_i has a start time s_i and a finish time f_i.
- Output: a subset of the n activities with the largest possible size such that no two activities overlap in time with each other.
- If activities = classes, the problem consists of attending the maximum number of classes.

Example

a_i	s_i	f_i
a_1	0	6
a_2	1	4
a_3	3	5
a_4	3	8
a_5	4	7
a_6	5	9
a_7	6	10
a_8	8	11



- What's the best solution?
- Answer: $\{a_2, a_5, a_8\}$
- Algorithm?

Generic algorithm

1. Consider the intervals in some order.
2. Pick the first interval from the sequence.
3. Remove from the sequence all intervals which are not compatible with the interval chosen in step 2.
4. Repeat steps 2 and 3 until the sequence is empty.

Strategies for step 1

- Several strategies for step 1.
- Any idea?

Choose the interval that starts the earliest first

- A possible strategy: choose the interval that starts the earliest first.
- In our example, we would get the following sequence:
  a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
- Algorithm's execution:

  iter 0: a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
  iter 1: pick a_1, remove a_2 a_3 a_4 a_5 a_6
  a_1 a_7 a_8
  iter 2: pick a_7, remove a_8
  a_1 a_7
  fim: Final solution: $\{a_1, a_7\}$. Not optimal.

Another strategy: shortest interval first

a_i	s_i	f_i	d_i
a_1	0	6	6
a_2	1	4	3
a_3	3	5	2
a_4	3	8	5
a_5	4	7	3
a_6	5	9	4
a_7	6	10	4
a_8	8	11	3

iter 0: sort by d_i (duration)
a_3 a_2 a_5 a_8 a_6 a_7 a_4 a_1

iter 1: a_3, remove a_2 a_5 a_4 a_1
a_3 a_8 a_6 a_7

iter 2: a_8, remove a_6 a_7
a_3 a_8

fim: Final solution: $\{a_3, a_8\}$.
Not optimal.

Another strategy: earliest finish time first

a_i	s_i	f_i
a_1	0	6
a_2	1	4
a_3	3	5
a_4	3	8
a_5	4	7
a_6	5	9
a_7	6	10
a_8	8	11

iter 0: sort on f_i
a_2 a_3 a_1 a_5 a_4 a_6 a_7 a_8

iter 1: a_2, remove a_3 a_1 a_4
a_2 a_5 a_6 a_7 a_8

iter 2: a_5, remove a_6 a_7
a_2 a_5 a_8

fim: We got the optimal solution!
Does it always work? Yes, let's prove it.

Proof

- The idea is to show that at any step of the execution, out algorithm is as good as any other algorithm.
- We will use induction.
- Need some notation.

Proof

- Let O be the set of intervals returned by the greedy algorithm (earliest finish time first).
- Let O^* be an optimal set of intervals.
- We could try proving that $O = O^*$, but that's asking too much. There can be several optimal solutions.
- Let's simply try to prove that $|O| = |O^*|$. In other words, we will try to prove that O is also optimal.

Proof (cont.)

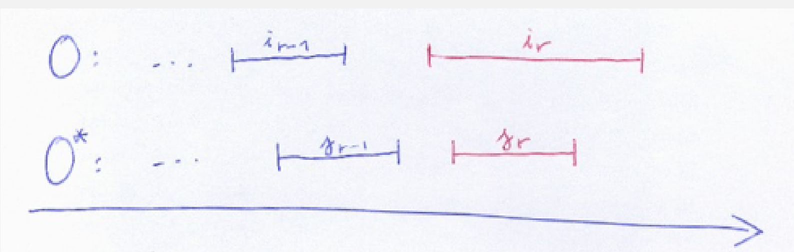
- Let i_1, i_2, ..., i_k, be the ser of intervals in O in the same order in which they were chosen by the greedy algorithm.
- Likewise, let j_1, j_2, ..., j_m, be the set of intervals in O^*.
- $\Rightarrow$ We must prove that $k = m$.
- Note: The intervals in O^* do not intercept each other. Otherwise they couldn't belong to the final solution.

Proof (cont.)

- Let us prove that $\forall_{r \leq k}\ f(i_r) \leq f(j_r)$.
- In words, we will prove that the r^{th} interval in O doesn't end later that the r^{th} interval in O^*.
- Proof by induction.
- <u>Base case</u>: $r = 1$
  - $f(i_1) \leq f(j_1)$
  - Why?
  - Because the greedy algorithm always chooses the interval with the least end time.

Proof (cont.)

- <u>Inductive hypothesis</u>: Assume $f(i_{r-1}) \leq f(j_{r-1})$.
- <u>Inductive step</u>: Prove that $f(i_r) \leq f(j_r)$.



- Is it possible that $f(i_r) > f(j_r)$?
- Answer: No. If that was the case, it would imply that the greedy algorithm was not choosing the interval that ends the earliest.

Proof (cont.)

- Note that j_r is available to be chosen by the greedy algorithm (i.e., j_r doesn't intercept any interval in O).
- $f(i_{r-1}) \leq f(j_{r-1}) \leq s(j_r)$
  The first inequality is justified by the inductive hypothesis, the second inequality because the intervals cannot intercept each other.
- If $f(j_r) < f(i_r)$ then the greedy algorithm would choose j_r instead of i_r.
- Therefore $f(i_r) \leq f(j_r)$ $\square$

Proof (cont.)

- We only need to prove that $k = m$.
- We can do it by contradiction.
- Suppose O is not optimal.
  - $\implies |O^*| > |O| \Leftrightarrow m > k$.

Proof (cont.)

- We already proved by induction that $f(i_k) \leq f(j_k)$, $\forall_{r \leq k}$.
- If $m > k$, then there must be some interval j_{k+1} in O^*. Such an interval has to start after j_k's end time, and therefore, after i_k's end time.
- This implies that interval j_{k+1} would be available to be chosen by the greedy algorithm.
- But the greedy algorithm stops with i_k which is a contradiction. Therefore O is optimal. $\square$

Complexity

- $\Theta(n \lg n)$ to sort on f_i.
- $\Theta(n)$ for the rest of the algorithm.
- Total: $\Theta(n \lg n)$

Summary

- We have seen several greedy strategies for the activity selection problem, and we could have thought of many others.
- Not all strategies result in optimal solutions.

Properties

Problems that can be solved to optimality by a greedy algorithm, have two important properties:

1. The greedy-choice property.
2. Optimal substructure.

Properties

- Greedy-choice property: An optimal solution can be constructed by making a sequence of locally optimal (greedy) choices.
- Optimal substructure: An optimal solution for a problem contains within it optimal solutions to subproblems.

Our greedy algorithm has optimal substructure

- Can be seen from the induction proof.
- $O' = O - \{i_k\}$ is the optimal solution for the subproblem confined to the intervals that start before i_{k-1} ends.