Algorithms 06

CS201

Kaustuv Nag

Activity-selection Problem

- Suppose we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
- Each activity a_i has a start time s_i and a finish time f_i, where $0 \leq s_i < f_i < \infty$.
- If selected, activity a_i takes place during the half-open time interval $[s_i, f_i)$.
- Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. That is, a_i and a_j are compatible if $s_i \geq f_j$ or $s_j \geq f_i$. In the activity-selection problem, we wish to select a maximum-size subset of mutually compatible activities.

Activity-selection Problem

An Example

- Assume that the activities are sorted in monotonically increasing order of finish time:

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{n-1} \leq f_n \tag{1}$$

- consider the following set S of activities:

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- The subset $\{a_3, a_9, a_{11}\}$ consists of mutually compatible activities.
- $\{a_1, a_4, a_8, a_{11}\}$ is a largest subset of mutually compatible activities; another largest subset is $\{a_2, a_4, a_9, a_{11}\}$.

Activity-selection Problem

Optimal Substructure

- Let S_{ij} be the set of activities that start after activity a_i finishes and that finish before activity a_j starts.
- A maximum set of mutually compatible activities in S_{ij} is A_{ij}, which includes some activity a_k.
- By including a_k in an optimal solution, we are left with two subproblems: finding mutually compatible activities in the set S_{ik} and in the set S_{kj}.
- Let $A_{ik} = A_{ij} \cap S_{ik}$ and $A_{kj} = A_{ij} \cap S_{kj}$, so that A_{ik} contains the activities in A_{ij} that finish before a_k starts and A_{kj} contains the activities in A_{ij} that start after a_k finishes.
- Thus $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$, and so the maximum-size set A_{ij} of mutually compatible activities in S_{ij} consists of $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$ activities.

Activity-selection Problem

Optimal Substructure

- If we could find a set A'_{kj} of mutually compatible activities in S_{kj} where $A'_{kj} > A_{kj}$, then we could use A'_{kj}, rather than A_{kj}, in a solution to the subproblem for S_{ij}.
- We would have constructed a set of $|A'_{ik}|+|A_{kj}|+1 > |A_{ik}|+|A_{kj}|+1 = |A_{ij}|$ mutually compatible activities, which contradicts the assumption that A_{ij} is an optimal solution. A symmetric argument applies to the activities in S_{ik}.

Activity-selection Problem

Recurrence Relation

- If we denote the size of an optimal solution for the set S_{ij} by $c[i,j]$, then we would have the recurrence

$$c[i,j] = c[i,k] + c[k,j] + 1$$

- To find a_k, we need to check all activities in S_{ij}. Thus, we obtain

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset\,, \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset\,. \end{cases}$$

- Now, we may use dynamic programming to solve this problem.

Greedy Algorithms

- Need to make a sequence of choices to achieve a global optimum.
- At rach stage, make the next choice based on some local criterion.
  - Drastically reduces space to search for solutions.
- Never go back and revise an earlier decision.
- How to prove that local choices achieve global optimum?

Greedy Algorithms

Activity Selection Problem

- Pick the next activity to allot based on a local strategy.
- Remove all activities that overlap with this slot.
- Argue that this sequence will maximize the number of the subset of mutually compatible activities.

Activity Selection Problem

Greedy Approach - I

- Choose the activity whose start time is the earliest.
- Counterexmaple:

Activity Selection Problem

Greedy Approach - II

- Choose the activity whose interval is the shortest.
- Counterexmaple:

Activity Selection Problem

Greedy Approach - III

- Choose the activity that overlaps with the minimum number of activities.
- Counterexmaple:

Activity Selection Problem

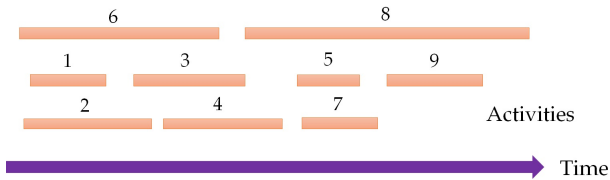
Greedy Approach - IV

- Choose the activity whose filnish time is earliest.
- Counterexamples?
- Proof?

Activity Selection Problem

Greedy Approach - IV: Algorithm

- Let S be the set of activites.
- Let A be the set of selected activities, initially empty.
- While S is not empty repeat the following:
  - Select $a \in S$ with the smallest finish time.
  - Add a to A.
  - Remove all activities from S that overlaps with a.



Activity Selection Problem

Greedy Approach - IV: Correctness

- Our algorithm produces a solution A.
- Let O be any optimal solution.
- A and O need not be identical.
  - There may be multiple optimal solutions of the same size.
- We need to show that $|A| = |O|$.

Activity Selection Problem

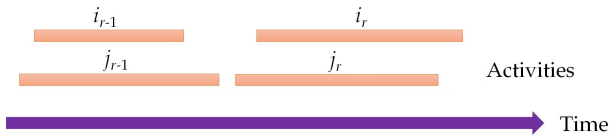
Greedy Approach - IV: Correctness

- Let $A = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$.
  - Assume sorted: $f_{i_1} \leq s_{i_2}, f_{i_2} \leq s_{i_3}$.
- Let $O = \{a_{j_1}, a_{j_2}, \ldots, a_{j_m}\}$.
  - Assume sorted: $f_{j_1} \leq s_{j_2}, f_{j_2} \leq s_{j_3}$.
- We need to show $k = m$.

Activity Selection Problem

Greedy Approach - IV: Correctness

- Claim: For each $r \leq k, f_{i_r} \leq f_{j_r}$
  - The greedy solution "stays ahead" of O.
- Proof: By induction on r
  - $r = 1$: Our approach chooses activity a_{i_1} with earliest overall finish time.
  - $r > 1$: Assume by induction that $f_{i_{r-1}} \leq f_{j_{r-1}}$.
    - Then, it must be the case $f_{i_r} \leq f_{j_r}$.
    - If not, algorithm would choose j_r rather than i_r.



Activity Selection Problem

Greedy Approach - IV: Correctness

- Suppose $m > k$
- We know that $f_{i_k} \leq f_{j_k}$
- Consider activity $a_{j_{k+1}} \in O$
  - Greedy algorithm terminates when S is empty.
  - Since $f_{i_k} \leq f_{j_k} \leq s_{j_{k+1}}$, this activity is compatible with $A = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$.
  - After selecting i_k, S still contains $a_{j_{k+1}}$.
    - This is a contradiction.

Greedy Algorithms

Procedure for Designing a Greedy Algorithm

- Identify optimal substructure.
- Cast the problem as a greedy algorithm with the greedy choice property.
- Write a simple iterative algorithm.

Greedy Algorithms: Activity Selection Problem

Greedy Choice Property

- Let S_k be a nonempty subproblem containing the set of activities that finish after activity a_k.
- Let a_m be an activity in S_k with the earliest finish time.
- Then a_m is included in some maximum-size subset of mutually compatible activities of S_k.
- **Proof:**
  - Let A_k be a maximum-size subset of mutually compatible activities in S_k and a_j be the activity in A_k with the earliest finish time.
  - If $a_j = a_m$, we are done, since we have shown that a_m is in some maximum-size subset of mutually compatible activities of S_k.
  - If $a_j \neq a_m$, let the set $A'_k = A_k - \{a_j\} \cup \{a_m\}$.
  - The activities in A'_k are disjoint, because (i) the activities in A_k are disjoint, (ii) a_j is the first activity in A_k to finish, and (iii) $f_m \leq f_j$.
  - Since $|A'_k| = |A_k|$, we conclude that A'_k is a maximum-size subset of mutually compatible activities of S_k, and it includes a_m.

Activity Selection Problem

Greedy Approach: Recursive

RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)

1 $m = k + 1$
2 **while** $m \leq n$ and $s[m] < f[k]$ // find the first activity in S_k to finish
3 $\quad m = m + 1$
4 **if** $m \leq n$
5 $\quad$ **return** $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
6 **else return** $\emptyset$

Activity Selection Problem

Greedy Approach: Iterative

```
GREEDY-ACTIVITY-SELECTOR(s, f)
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5       if s[m] ≥ f[k]
6            A = A ∪ {a_m}
7            k = m
8  return A
```

Knapsack Problem

- Given a knapsack which holds W pounds of stuff and a list of n items:

Item (i)	1	2	$\cdots$	n
Value (v_i)	v_1	v_2	$\cdots$	v_n
Weight (w_i)	w_1	w_2	$\cdots$	w_n
$\left(\frac{v_i}{w_i}\right)$	$\left(\frac{v_1}{w_1}\right)$	$\left(\frac{v_2}{w_2}\right)$	$\cdots$	$\left(\frac{v_n}{w_n}\right)$

  fill the knapsack with the most profitable items.
- Two variants:
  - *Integral knapsack:* Take an item or leave it.
    Fractional knapsack: Can take a fraction of an item (infinitely divisible).
- Both fractional and integral knapsack have optimal substructure.
- Only fractional knapsack has the greedy choice property.

Greedy Algorithms: Fractional Knapsack

Greedy Choice Property

- Let j be the item with maximum v_i/w_i. Then there exists an optimal solution in which you take as much of item j as possible.
- **Proof:**
  - Suppose, there exists an optimal solution that didn't take as much of item j as possible.
  - If the knapsack is not full, add some more of item j, and you have a higher value solution. **[Contradiction]**.
  - If the knapsack is full, there must exist some item $k \neq j$ with $v_k/w_k < v_j/w_j$ that is in the knapsack.
  - We also must have that not all of j is in the knapsack.
  - We can therefore take a piece of k, with ε weight, out of the knapsack, and put a piece of j with ε weight in.
  - This increases the knapsacks value by

$$\varepsilon\left(\frac{v_j}{w_j}\right) - \varepsilon\left(\frac{v_k}{w_k}\right) > 0$$

This is a contradition to the original solution being optimal.

Greedy Algorithms: Fractional Knapsack

Greedy Algorithm

- Sort items by $\left(\frac{v_j}{w_j}\right)$, renumber.
- **For** $i = 1$ to n
  - Add as much of item i as possible.

White Board

White Board