Greedy Algorithms

CSCI 3100

Overview

Like dynamic programming, used to solve optimization problems.

Problems exhibit optimal substructure (like DP).

Problems also exhibit the **greedy-choice** property.

- When we have a choice to make, make the one that looks best *right now*.
- Make a **locally optimal choice** in hope of getting a **globally optimal solution**.

Greedy Strategy

The choice that seems best at the moment is the one we go with.

- Prove that when there is a choice to make, one of the optimal choices is the greedy choice. Therefore, it's always safe to make the greedy choice.
- Show that all but one of the subproblems resulting from the greedy choice are empty.

Activity-selection Problem

Input: Set S of n activities, $a_1, a_2, \ldots, a_n$.

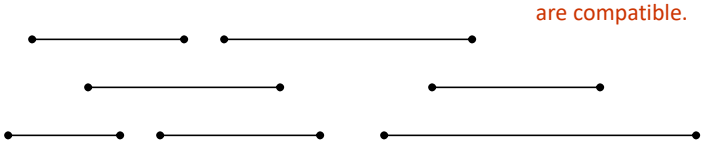
- s_i = start time of activity i.
- f_i = finish time of activity i.

Output: Subset A of maximum number of compatible activities.

- Two activities are compatible, if their intervals don't overlap.

Example:

Activities in each line are compatible.



Optimal Substructure

Assume activities are sorted by finishing times.

- $f_1 \le f_2 \le \ldots \le f_n$.

Suppose an optimal solution includes activity a_k.

- This generates two subproblems.
- Selecting from $a_1, \ldots, a_{k-1}$, activities compatible with one another, and that finish before a_k starts (compatible with a_k).
- Selecting from $a_{k+1}, \ldots, a_n$, activities compatible with one another, and that start after a_k finishes.
- The solutions to the two subproblems must be optimal.
  - Prove using the cut-and-paste approach.

Recursive Solution

Let S_{ij} = subset of activities in S that start after a_i finishes and finish before a_j starts.

Subproblems: Selecting maximum number of mutually compatible activities from S_{ij}.

Let $c[i, j]$ = size of maximum-size subset of mutually compatible activities in S_{ij}.

Recursive Solution:

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \phi \\ \max_{i<k<j} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \phi \end{cases}$$

Greedy Algorithm for Scheduling

Let T be the set of tasks, construct a set of independent tasks I, A is the rule determining the greedy algorithm

I = { }

While (T is not empty)

Select a task t from T by a rule A

Add t to I

Remove t and all tasks incompatible with t from T

Greedy Choices

- Schedule a task with earliest starting time
- Schedule a task with fewest conflicts
- Schedule a task with shortest duration
- Schedule a task with earliest finish time

Earliest finish time greedy property

Solution based on earliest finish time is optimal

- There is an optimal solution to the subproblem S_{ij}, that includes the activity with the smallest finish time in set S_{ij}.
- Can be proved easily.

a_m – activity with earliest finish time

Given set S of activities, let $A \subseteq S$ be the optimal solution. Prove that $a_m \in A$.

Proof: Let $a_k \in A$ be an activity with earliest finish time in A.

Case 1: $a_m = a_k$.

Case 2: $a_m \neq a_k$. Construct $A' = A - \{a_k\} \cup \{a_m\}$.

A' is a set of compatible tasks because a_m finishes before a_k. $|A'| = |A|$ ☒

Exploiting the greedy property

Hence, there is an optimal solution to S that includes a_1.

Therefore, make this greedy choice without solving subproblems first and evaluating them.

Solve the subproblem that ensues as a result of making this greedy choice.

Combine the greedy choice and the solution to the subproblem.

Recursive Algorithm

Recursive-Activity-Selector (*s, f, i, j*)

1. $m \leftarrow i+1$
2. **while** $m < j$ and $s_m < f_i$
3. **do** $m \leftarrow m+1$
4. **if** $m < j$
5. **then return** $\{a_m\} \cup$
6. Recursive-Activity-Selector(s, f, m, j)
6. else return ϕ

Straightforward to convert the algorithm to an iterative one. See the text.

remove incompatible solutions with a_i

Initial Call: Recursive-Activity-Selector (s, f, 0, n+1)

Complexity: $\Theta(n)$

Typical Steps

Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.

Prove that there's always an optimal solution that makes the greedy choice, so that the greedy choice is always safe.

Show that greedy choice and optimal solution to subproblem $\Rightarrow$ optimal solution to the problem.

Make the greedy choice and **solve top-down**.

May have to preprocess input to put it into greedy order.

- Example: Sorting activities by finish time.

Elements of Greedy Algorithms

Greedy-choice Property.

- A globally optimal solution can be arrived at by making a locally optimal (greedy) choice.

Optimal Substructure.