Lecture 6
Greedy algorithms: interval scheduling

COMP 523: *Advanced Algorithmic Techniques*
Lecturer: *Dariusz Kowalski*
Overview

Previous lectures:
• Algorithms based on recursion - call to the same procedure solving smaller-size input

This lecture:
• Greedy algorithms
• Interval scheduling
Greedy algorithm’s paradigm

Algorithm is **greedy** if :
- it builds up a solution in *small steps*
- it chooses a decision at each step myopically to *optimize* some *underlying criterion*

Analyzing **optimal greedy** algorithms by showing that:
- in every step it is not worse than any other algorithm, or
- every algorithm can be gradually transformed to the greedy one without hurting its quality
Interval scheduling

Input: set of intervals on the line, represented by pairs of points (ends of intervals)

Output: finding the largest set of intervals such that none two of them overlap

Greedy algorithm:

• Select intervals one after another using some rule
Rule 1

Select the interval which starts earliest (but not overlapping the already chosen intervals)

Underestimated solution!
Rule 2

Select the interval which is shortest (but not overlapping the already chosen intervals)

Underestimated solution!
Rule 3

Select the interval with the fewest conflicts with other remaining intervals (but still not overlapping the already chosen intervals)

Underestimated solution!
Rule 4

Select the interval which ends first (but still not overlapping the already chosen intervals)

Hurray! Exact solution!

________   _______   _______   _______   _______
________________________________________________________________________

_________   __________________

_________

_________   _______   _______   _______   _______   _______   _______

_________   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______   _______

Lecture 6: Greedy algorithms
Analysis - exact solution

Algorithm gives non-overlapping intervals:
obvious, since we always choose an interval which does
not overlap the previously chosen intervals

The solution is exact:
Let $A$ be the set of intervals obtained by the
algorithm, and $\text{Opt}$ be the largest set of pairwise
non-overlapping intervals. We show that $A$ must
be as large as $\text{Opt}$. 
Let $A = \{A_1, \ldots, A_k\}$ and $Opt = \{B_1, \ldots, B_m\}$ be sorted. By definition of $Opt$ we have $k \leq m$.

**Fact:** for every $i \leq k$, $A_i$ finishes not later than $B_i$.

**Proof:** by induction.

For $i = 1$ by definition of a step in the algorithm.

Suppose that $A_{i-1}$ finishes not later than $B_{i-1}$. From the definition of a step in the algorithm we get that $A_i$ is the first interval that finishes after $A_{i-1}$ and does not overlap it. If $B_i$ finished before $A_i$ then it would overlap some of the previous $A_1, \ldots, A_{i-1}$ and consequently - by the inductive assumption - it would overlap $B_{i-1}$, which would be a contradiction.
Analysis - exact solution \textit{cont.}

**Theorem:** A is the exact solution.

**Proof:** we show that $k = m$.

Suppose to the contrary that $k < m$. We have that $A_k$ finishes not later than $B_k$. Hence we could add $B_{k+1}$ to $A$ and obtain bigger solution by the algorithm - contradiction.
Time complexity

Implementation:

• Sorting intervals according to the right-most ends
• For every consecutive interval:
  – If the left-most end is after the right-most end of the last selected interval then we select this interval
  – Otherwise we skip it and go to the next interval

Time complexity: \( O(n \log n + n) = O(n \log n) \)
Conclusions

- Greedy algorithms: algorithms constructing solutions step after step using a local rule
- Exact greedy algorithm for interval selection problem - in time $O(n \log n)$ illustrating “greedy stays ahead” rule
Textbook and Exercises

- Chapter 4 “Greedy Algorithms”
- All Interval Sorting problem from Chapter 4