6.046 - Design & Analysis of Algorithms
Prof. Piotr Indyk, Ronitt Rubinfeld, & Ron Rivest
stellar.mit.edu/S/course/6/fa11/6.046

Outline:
- Course overview & admin
- Some problems related to interval scheduling

Admin:
- Course info handout (all else on web)
- Pre-reqs: 6.006 & (6.042 v18.310)
- Recitations: Fridays (as by Registrar)
- Psets: 6 of them
  - exponential penalty for missed problems
  - collaboration good; copying/plagiarism bad

Exams:
- Quiz 1: in class Tue 10/18
- Quiz 2: take-home
  - Nov 15 → Nov 22
  - (no collaboration)
- Final
Outline:

- # theory
- randomization
- data structures
- graphs
- optimization
- complexity
- advanced topics
Today's theme: Very similar problems can have very different solutions & complexity.

Recall: $P =$ class of problems solvable in polynomial time
e.g. multiplying two $n$-bit #'s in time $O(n^2)$

$NP =$ class of problems whose solution can be verified in polynomial time
e.g. is given boolean formula satisfiable? (SAT)
e.g. is given integer composite (not prime)

We know $P \subseteq NP$, but don't know if $P = NP$

The NP-complete problems are not in $P$, if $P \neq NP$.
(e.g. SAT, Travelling Salesman, ... 1000's ...)

[More in lectures 17-18]
Scheduling: Fascinating problem area, with many variants. (We'll look at three.)

- Suppose we have a valuable resource (classroom, Hubble telescope) that can be used by at most one party at a time. \( \Rightarrow \) Its use must be scheduled.

- Assume we are given a set \( R \) of \( n \) requests
  \[ R = \{ R_1, R_2, \ldots, R_n \} \]
  for use of this resource.

- We want to find a schedule \( S \subseteq R \), a set of compatible requests, that optimizes some given criterion.

- Simplest case: each request \( R_i \) specifies
  - a start time \( s(i) \)
  - a finish time \( f(i) \)

- \( R_i \) and \( R_j \) are compatible if they don't overlap:
  \[ f(i) \leq s(j) \text{ or } f(j) \leq s(i) \]
First Problem: **Interval Scheduling**
Given $R$, find schedule $S$ that maximizes $|S|$.

**Example:**

\[
\begin{align*}
\text{Optimal schedule} & = \{ R_4, R_5, R_6 \} \quad \text{(or \{ R_4, R_5, R_3 \})} \\
\text{Brute force:} & \quad \text{There are } 2^n \text{ subsets of } R. \\
& \quad \text{Testing a subset for compatibility takes time } O(n^3). \\
& \quad \text{Overall time is } O(n^3 \cdot 2^n) \\
& \quad \text{Not polynomial time 😞} \\
\text{We solve Interval Scheduling in time } O(n \log n) \\
& \quad \text{using a greedy method.}
\end{align*}
\]
Greedy method for Interval Scheduling:

\[ S = \emptyset \]

while \( \mathcal{R} \neq \emptyset \):

- Pick \( R_i \) from \( \mathcal{R} \) according to some (greedy, myopic) rule.

- \( S = S \cup \{ R_i \} \)

- Remove from \( \mathcal{R} \) the request \( R_i \) and all requests incompatible with \( R_i \).

return \( S \)

With right rule, produces optimal schedule (in polynomial time).
Which rule?

1. Pick $R_i$ that starts first (minimizes $s(i)$)

2. Pick $R_i$ that is smallest (minimizes $f(i) - s(i)$)

3. Pick $R_i$ with fewest incompatibilities

4. Pick $R_i$ that finishes first (minimizes $f(i)$)

(intuition: leaves maximum amount of time available for other requests)
Theorem: Greedy method (with first-to-finish rule) gives an optimum schedule.

Proof:  • Suppose greedy alg returns \( S = \{ R_{i_1}, R_{i_2}, ..., R_{i_p} \} \) so \( |S| = p \) & results "in order" \( (R_{i_i} \text{ finishes first}) \)

  • Let \( S^* = \{ R_{j_1}, R_{j_2}, ..., R_{j_q} \} \) be some optimal schedule (in order). (This maximizes \( |S^*| = q \).)

  • There exists an optimal schedule \( S^{**} \) starting with \( R_{i_1} \).
    (Replace \( R_{j_1} \) with \( R_{i_1} \) in \( S^* \); OK since \( f(i_i) \leq f(j_1) \).)

  • Let \( \mathcal{R}^{x} = \{ R_i \in \mathcal{R} \mid s(i) > x \} \)

  • Then \( \{ R_{j_2}, ..., R_{j_q} \} \) must be optimal for \( \mathcal{R}^{x} f(i) \)
    (otherwise we could replace it with better solution, improving \( S^{**} \).

  • By induction, greedy method gives optimal solution.

  • Note: \( \text{opt}(\mathcal{R}) = 1 + \text{opt}(\mathcal{R}^{f(i)}) \)
    where \( \text{opt}(\mathcal{R}) = \text{size of optimal solution} \)
    \( i_i = i \) s.t. \( R_i \) finishes first
Running time: $O(n \lg n)$

- sort requests in order of increasing $f(i)$
- consider each in turn, adding it to $S$
  if it is compatible with last interval added to $S$. (or if it is 1st interval)
Second Problem: Weighted Interval Scheduling

Same as before, but now each request $R_i$ also has a "weight" $w(i)$. We seek schedule $S \subseteq R$ of maximum total weight.

E.g. $w(i) = \text{bid}$ (max revenue)

$w(i) = f(i) - s(i)$ (max utilization)

$w(i) = \# \text{students}$ (max student 😊)

Note: Greedy no longer works

First to finish rule $\Rightarrow 1 + 2 = 3$

max weight rule $\Rightarrow 4$

optimum $= 3 + 2 = 5$
Dynamic Programming works on Weighted Interval Scheduling.

Subproblems are $\mathcal{R}^x = \{ R_i \in \mathcal{R} \mid s(i) \geq x \}$

Only $n$ different subproblems; one for each $x = f(i)$.

Only need to solve each subproblem once (save soln for later use).

Since greedy doesn't work, we don't have rule telling us which interval to schedule first.

So, we try each $R_i$ in turn as possible "first".

Given $R_i$ as tentative first, remaining is $\mathcal{R}^{3f(i)}$.

Program: $\text{opt}(\mathcal{R}) = \max_{1 \leq i \leq n} (w_i + \text{opt}(\mathcal{R}^{3f(i)}))$

Running time is $O(n^3)$

Exercise: show how to use sorting initially & reduce running time to $O(n \log n)$

Note: Actual optimum schedule can be obtained if we record $i$ that yields max for each subproblem.
Third Problem: "Flexible" interval scheduling (unit weights, unweighted)

Each request $R_i$ specifies:

$s(i) =$ earliest possible start time (11 am)

$f(i) =$ latest possible finishing time (5 pm)

$l(i) =$ length requested ($\leq f(i) - s(i)$) (2 hours)

Want schedule of maximum size.

(Solution must specify when each scheduled interval actually starts.)
"Flexibility" makes problem harder!

**Thm:** Flexible interval scheduling is NP-complete.

**Thm:** Greedy algorithm gives good approximate soln. (within factor of 2 of optimal)

**Thm:** Fancier approximation algorithm achieves factor $e/(e-1) = 1.582...$

[Chuzhoy, Ostrovsky, Rabani 2006]

**Thm:** ($\exists \epsilon$) Getting within $(1+\epsilon)$ of optimum is NP-hard. [Spieksma 1999]
Summary:

Very similar problems can have very different solutions & complexity.

- Interval scheduling is in P (greedy method).
- Weighted interval scheduling is in P (dynamic programming).
- Flexible interval scheduling is NP-complete
  (Good approximation algorithms exist, but getting very good approximations is NP-hard.)

Good algorithms make a difference!
So does knowing when to stop looking for one!