LECTURE 1
Introduction

Prof. Piotr Indyk
Plan for today

• Part I: Administrativia
  http://stellar.mit.edu/S/course/6/fa13/6.046/
  See also the handout

• Part II: Scheduling Algorithms
  (a.k.a. 6.046 preview)
6.046 Lineup

• Lecturers:
  
  Piotr Indyk
  Shafi Goldwasser
  Ronitt Rubinfeld

• TAs: Nathan Arce, Adam Gleitman, Bonny Jain, Christina Lee, Hayden Metsky, Casey O'Brien
Prerequisites

• 6.006 *Introduction to Algorithms* (or 6.001 *Structure and Interpretation of Computer Programs*) and

• Either 6.042J/18.062J *Mathematics for Computer Science* or 18.310 *Principles of Applied Mathematics*

• We will assume a good understanding of that material (proofs, probability, dynamic programming, etc…)

• If you did not take the pre-reqs, talk to one of the lecturers
Lectures etc

• Lectures: Tue/Thu
• Recitations: Fri (bldg. 36 or 26)
  – Check Stellar for the assignment
• Book: CLRS, 3rd edition

9/7/2013
Grading

• Exams: Quiz1 (October 22), Quiz2 (November 14), Final
• 6 problem sets
• Score breakdown:
  – Problem sets 25%
  – Quiz 1 20%
  – Quiz 2 25%
  – Final 30%
• But you need to at least attempt to solve each problem, or your grade is reduced (see course information for details)
Collaboration policy

• OK to work in a group but:
  – You must write up each problem solution by yourself without assistance, and be able to explain orally if needed
  – You must acknowledge collaborators and sources
  – No collaboration whatsoever is permitted on quizzes or exams

• Plagiarism and other dishonest behavior will not be tolerated
Part II: Course Overview/Scheduling

Algorithms

- Old friends: Divide and Conquer, Dynamic Programming, Shortest Paths
- New acquaintances:
  - Greedy algorithms
  - Randomization
  - Approximation
  - Reduction
  - Amortization
  - Complexity theory
- Topics: Linear programming, Cryptography, Distributed Algorithms, Sub-linear algorithms, String matching, etc
Interval Scheduling

• Given: n requests $R = r_1, \ldots, r_n$ to use a single resource for a time interval
  – Each request $r_i = [a_i, b_i]$
  – Two requests $r_i, r_j$ are compatible if the intervals do not overlap

• Goal: find compatible subset of requests of maximum size (denoted by $OPT(R)$)
Greedy approach

- Greedy algorithm: repeatedly makes locally best choices, ignoring the future
  “Think globally, act locally”
- Greedy interval scheduling:
  - Use some simple rule to pick the “best” interval $r_i$
  - Remove all intervals incompatible with $r_i$
  - Repeat
- What rule to pick?
Possible greedy rules

Quiz 0: which of the following rules work?

a. Select interval that starts first
b. Select shortest interval
c. Select interval that ends first

Answer: c. See Appendix for counter-examples
Select interval that ends first (SITEF)

- Theorem: greedy with the SITEF rule always computes *optimal* solution
- Proof: by induction on $n$.
  - Let $r_{OPT} = \text{leftmost interval in } \text{OPT}(R)$
  - Let $r_G = \text{interval that ends first in } R$
  - Let $R' = R - \text{intervals overlapping with } r_G$
  - Claim: $r_G$ is compatible with $\text{OPT}(R) - \{r_{OPT}\}$
    (because whatever comes after $r_{OPT}$ also comes after $r_G$)
  - Therefore $\text{OPT}(R) - \{r_{OPT}\}$ is included in $R'$
  - By induction, our algorithm will find at least $|\text{OPT}(R)| - 1$ compatible intervals in $R'$
  - Together with $r_G$, it will return $|\text{OPT}(R)|$ compatible intervals
Running time of SITEF

- Sort requests $r_i = [a_i, b_i]$ by $b_i$ \( \text{O}(n \log n) \)
- Scan requests from left to right, maintaining the current finish time $f$, and skipping requests that start before $f$ \( \text{O}(n) \)

Total time: \( \text{O}(n \log n) \)
Variant 2: Weighted Interval Scheduling

• Given: \( n \) requests \( r_1, \ldots, r_n \) to use a single resource for a time interval, as before
  – Each request \( r_i = [a_i, b_i] \) has a weight \( w_i \)
• Goal: find compatible subset of requests of maximum weight
• Greedy ?
  – Is short but light interval better than long and heavy one?

\[
\begin{array}{c}
4 \\
2 & 3
\end{array}
\]
Dynamic Programming

• DP=recursive formula+overlapping subproblems

• Algorithm:
  – Sort all $r_i$ by the start time
  – Consider $r_1$:
    • If $r_1$ not in OPT(R), then the optimal cost is
      $\text{weight}(\text{OPT}(R)) = \text{weight}(\text{OPT}(R-r_1))$
    • If $r_1$ in OPT(R), then the optimal cost is
      $\text{weight}(\text{OPT}(R)) = w_1 + \text{weight}(\text{OPT}(R-r_1-r_2-\ldots-r_i))$, where $r_2\ldots r_i$ are incompatible with $r_1$
  – Recurrence:
    $\text{weight}((\text{OPT}(R)) = \max[w_1 + \text{weight}(\text{OPT}(R-r_1-r_2-\ldots-r_i)), \text{weight}(\text{OPT}(R-r_1))]$
Analysis

• Recurrence:
  \[ \text{weight((OPT(R))=max[w_1+weight(OPT(R-r_1-r_2-\ldots-r_i)), weight(OPT(R-r_1))]} \]

• Number of sub-problems:
  \[ n \]

• Time per sub-problem:
  \[ O(n) \]

• Total: \[ O(n^2) \]

• Can be improved to \[ O(n \log n) \]

9/7/2013
Variant 3: Flexible job scheduling

- Each request $r_i$ consists of
  - $a_i$: earliest possible starting time
  - $b_i$: latest possible ending time
  - $l_i$: length (which is at most $b_i - a_i$)

- Goal: find a maximum subset of compatible requests (with actual starting/ending times)
  Note: interval scheduling is a special case where $l_i = b_i - a_i$

- Algorithm?
Flexible job scheduling, ctd

• Unfortunately, the problem is NP-complete
  – I.e., unlikely to have a polynomial time algorithm
• In fact, it is NP-complete to find a solution within a factor of 1.00001 of the optimal
• Good news: there is a greedy algorithm that finds a solution within a factor of 2 of the optimal (or even 1.582…)

9/7/2013
Appendix: Counterexamples

• Select interval that starts first

• Select shortest interval

• Select interval with fewest incompatibilities

9/7/2013