Introduction to Algorithms

Lecture 24
Prof. Piotr Indyk
Dealing with Hard Problems

• What to do if:
  – Divide and conquer
  – Dynamic programming
  – Greedy
  – Linear Programming/Network Flows
  – …

does not give a polynomial time algorithm?
Dealing with Hard Problems

• Solution I: Ignore the problem
  – Can’t do it! There are **thousands** of problems for which we do not know polynomial time algorithms
  – For example:
    • Traveling Salesman Problem (TSP)
    • Set Cover
Traveling Salesman Problem

- Traveling Salesman Problem (TSP)
  - Input: undirected graph with lengths on edges
  - Output: shortest cycle that visits each vertex exactly once
- Best known algorithm: $O(n \ 2^n)$ time.
Set Covering

- Set Cover:
  - Input: subsets $S_1 \ldots S_n$ of $X$, $\bigcup_i S_i = X$, $|X| = m$
  - Output: $C \subseteq \{1 \ldots n\}$, such that $\bigcup_{i \in C} S_i = X$, and $|C|$ minimal

- Best known algorithm: $O(2^n m)$ time(?)

Bank robbery problem:
- $X = \{\text{plan, shoot, safe, drive, scary}\}$
- Sets:
  - $S_{\text{Joe}} = \{\text{plan, safe}\}$
  - $S_{\text{Jim}} = \{\text{shoot, scary, drive}\}$
  - 
  - 
  - 
  - 
  -
Dealing with Hard Problems

- Exponential time algorithms for small inputs. E.g., \((100/99)^n\) time is not bad for \(n < 1000\).
- Polynomial time algorithms for some (e.g., average-case) inputs
- Polynomial time algorithms for all inputs, but which return approximate solutions
Approximation Algorithms

• An algorithm $A$ is $\rho$-approximate, if, on any input of size $n$:
  – The cost $C_A$ of the solution produced by the algorithm, and
  – The cost $C_{OPT}$ of the optimal solution are such that $C_A \leq \rho \cdot C_{OPT}$

• We will see:
  – $2$-approximation algorithm for TSP in the plane
  – $\ln(m)$-approximation algorithm for Set Cover
Comments on Approximation

- "$C_A \leq \rho \ C_{\text{OPT}}$" makes sense only for minimization problems
- For maximization problems, replace by "$C_A \geq 1/\rho \ C_{\text{OPT}}$"
- Additive approximation "$C_A \leq \rho + C_{\text{OPT}}$" also makes sense, although difficult to achieve
2-approximation for TSP

- Compute MST $T$
  - An edge between any pair of points
  - Weight = distance between endpoints
- Compute a tree-walk $W$ of $T$
  - Each edge visited twice
- Convert $W$ into a cycle $C$ using shortcuts
2-approximation: Proof

- Let $C_{\text{OPT}}$ be the optimal cycle
- $\text{Cost}(T) \leq \text{Cost}(C_{\text{OPT}})$
  - Removing an edge from $C$ gives a spanning tree, $T$ is a spanning tree of minimum cost
- $\text{Cost}(W) = 2 \ \text{Cost}(T)$
  - Each edge visited twice
- $\text{Cost}(C) \leq \text{Cost}(W)$
  - Triangle inequality

$\Rightarrow \ \text{Cost}(C) \leq 2 \ \text{Cost}(C_{\text{OPT}})$
Approximation for Set Cover

Greedy algorithm:

- Initialize $C = \emptyset$
- Repeat until all elements are covered:
  - Choose $S_i$ which contains largest number of yet-not-covered elements
  - Add $i$ to $C$
  - Mark all elements in $S_i$ as covered
Greedy Algorithm: Example

- $X\{1,2,3,4,5,6\}$
- Sets:
  - $S_1\{1,2\}$
  - $S_2\{3,4\}$
  - $S_3\{5,6\}$
  - $S_4\{1,3,5\}$
- Algorithm picks $C\{4,1,2,3\}$
- Not optimal!
\textbf{ln(m)-approximation}

- Notation:
  - $C_{\text{OPT}}$ = optimal cover
  - $k = |C_{\text{OPT}}|$
- Fact: At any iteration of the algorithm, there exists $S_j$ which contains at $\geq 1/k$ fraction of yet-not-covered elements
- Proof: by contradiction.
  - If all sets cover $<1/k$ fraction of yet-not-covered elements, there is no way to cover them using $k$ sets
  - But $C_{\text{OPT}}$ does that!
- Therefore, at each iteration greedy covers $\geq 1/k$ fraction of yet-not-covered elements
ln(m)-approximation

• Let $u_i$ be the number of yet-not-covered elements at the end of step $i=0,1,2,...$
• We have
  
  $$u_{i+1} \leq u_i (1-1/k)$$
  $$u_0 = m$$
• Therefore, after $t=k \ln m$ steps, we have
  $$u_t \leq u_0 (1-1/k)^t \leq m (1-1/k)^{k \ln m} < m 1/e^{\ln m} = 1$$
• I.e., all elements are covered by the $k \ln m$ sets chosen by greedy algorithm
• Opt size is $k \Rightarrow$ greedy is $\ln(m)$-approximate
Approximation Algorithms

• Very rich area
  – Algorithms use greedy, linear programming, dynamic programming
    • E.g., 1.01-approximate TSP in the plane
  – Sometimes can show that approximating a problem is as hard as finding exact solution!
    • E.g., $0.99 \ln(m)$-approximate Set Cover